For office use only	For office use only
T1 ________________	F1 ________________
T2 ________________	F2 ________________
T3 ________________	F3 ________________
T4 ________________	F4 ________________

2015

18th Annual High School Mathematical Contest in Modeling (HiMCM) Summary Sheet

Team Control Number: 5813

Problem Chosen: A

Traffic jam has become a significant problem for big cities in the past few years, because serious road rage occurs in lane closure situations. Thus, we researched on the effect of various strategies to control the traffic in lane closure situations.

The Fluid Dynamics Traffic model is marvel in that it maps the discrete reality into a continuous model. We consider three factors: traffic flux, speed and traffic density. By deducing their relationships through the law of traffic conservation, we are able to obtain the key differential equation. Solving the partial differential equation enables us to achieve any traffic density at any position and at any time on the road quickly.

Although Fluid Dynamics Traffic model has access to car densities quickly, because of its continuity, such a model is not sufficient for a real-life simulation. We adopted Car-following model to do the simulation. Car-following model describes the interaction between each car and the car in front of it. In our model, each car determines its speed according to a set of equalities considering the properties of the preceding cars. We keep updating the speed of each car until it exit the system. Also, we set up inequalities to determine whether it's safe for a car to merge into another lane.

To begin with, we suggested three strategies to control the traffic: all cars merge as soon as possible; all cars merge according to its order; and all cars only merge according to speed of surrounding cars.

Then, we judged the fairness and efficiency of each situation. We define that, fair strategies have small variance of waiting time, and efficient strategies have small average waiting time. Therefore, we obtained the most efficient and the fairest strategies of drivers in different situations by experiments. When two lanes merge to a single lane, merging last is always the fairest and the most efficient strategy. When the number of lanes increases, merging in order is almost the best strategy, but sometimes less efficient than merging according to speed. When the speed limitation halves, the efficiency of the three strategies stays at a similar level, while merging in order and merging according to speed have slightly better fairness than merging first.

Table of Content

1 Problem Restatement

On a highway, a two-lane road is reduced to a one-lane road. We are required to design a model to simulate real world road system and to analyze the behaviors of drivers. Then, according to the data provided by our model, we are required to determine the efficiency and fairness of each strategy, in order to control the traffic flow. Then, the generality of the problem is improved with three-lane road and roads with lower speed limitations. Finally, we are required to provide guidelines for drivers, as well as a letter for the Department of Transportation to urge the guidelines to be carried out.

2 Variable Definitions

Variables	***Definitions***
$\boldsymbol{L}$	Road length.
$\boldsymbol{\lambda}$	Expected number of cars entering the system per minute.
$\boldsymbol{\Delta t}$	Time interval of simulation.
$\boldsymbol{l}$	Length of cars.
$\boldsymbol{v^{max}}$	Maximum speed allowed on the road.
$\boldsymbol{a^{max}}$	Maximum acceleration allowed by the car.
$\boldsymbol{b^{max}}$	Maximum deceleration allowed by the car.
$\boldsymbol{T}$	Human reaction time.
$\boldsymbol{k}$	Minimum distance between stationary cars.
$\boldsymbol{C_n^p}$	Car $\#n$ on lane p. (l represents left lane, r represents right lane)
$\boldsymbol{x_n^p(t)}$	Displacement of car $\#n$ on lane p at time t. (l represents left lane, r represents right lane)
$\boldsymbol{v_n^p(t)}$	Speed of car $\#n$ on lane p at time t. (l represents left lane, r represents right lane)
$\boldsymbol{a_n^p(t)}$	Acceleration of car $\#n$ on lane p in one time interval. (l represents left lane, r represents right lane)
$\boldsymbol{b_n^p(t)}$	Deceleration of car $\#n$ on lane p in one time interval. (l represents left lane, r represents right lane)
$\boldsymbol{S_n^p(t)}$	Safe distance between car $\#n$ and $\#(n-1)$ on lane p at time t. (l represents left lane, r represents right lane)
$\boldsymbol{Z_n^p(t)}$	Whether the car $\#n$ on lane p at time t wants to emerge. (l represents left lane, r represents right lane)
$\boldsymbol{v_n^a(t)}$	The maximum speed car $\#n$ can accelerate to at time t.
$\boldsymbol{v_n^b(t)}$	The minimum speed car $\#n$ can decelerate to at time t.
$\boldsymbol{v_n^s(t)}$	The maximum safe speed of car $\#n$.
$\boldsymbol{q}$	Traffic flux.
$\boldsymbol{\rho}$	Traffic density (number of cars per unit of length of road).

3 Assumptions

a. All cars are considered the same size, same maximum acceleration and maximum deceleration (braking rate), and all drivers have the same reaction time.
Justification: to study the efficiency and fairness generally, the individual diversity before entering the system is not taken into account.

b. Before noticing the sign of lane closure, all cars enter the system according to equal Poisson distributions on each lane.
Justification: there should be no difference in each lane before the sign shows up.

c. After and before entering the studied traffic system, we consider the traffic flow to be unlimited.
Justification: to study the given system of mergence, we eliminate all the unnecessary factors.

d. If a car is currently in an unblocked lane, it will not shift to any other lane.
Justification: We simplify the situation by reducing the number of unnecessary shifts in lanes.

4 Model Design & Part I

4.1 Fluid Dynamics Traffic Model

4.1.1 Derivation

(This model is not used for testing and problem solving due to its complexity.) We consider the traffic flow as a continuous compressible fluid model. Hence, before we start, it is important to consider three main physical quantities: cars speed $v = v\ (x, t)$, traffic density $\rho = \rho\ (x, t)$ and traffic flux $q = q\ (x, t)$. And by viewing the traffic as a fluid, we can obtain a key equation as $q\ (x, t) = v\ (x, t) \cdot \rho\ (x, t)$.

Now, we are aiming to deduce that if the density of the cars at time $t = 0$ in a part of the road form 0 to *l* is given ($\rho\ (x, 0)$ is known), we can find the density of cars in that part of the road at any time ($\rho\ (x, t)$).

Because of the absence of any sources or sinks of cars between $x = 0$ and $x = l$, the number of cars(N) is conserved.

N has relationship with q as $\frac{dN}{dt} = -q(0, t) + q(l, t)$

Also can N be calculated by formula $N = \int_0^l \rho\ (x,t)dx$

Thus $\frac{d}{dt}\int_0^l \rho\ (x,t)dx = -q(0,t) + q(l,t)$

Because 0 and *l* are constant, we can rewrite the above formula as

$$\int_0^l \frac{\partial\rho\ (x,t)}{\partial t}dx = \int_0^l -\frac{\partial q\ (x,t)}{\partial x}dx$$

Because we define *l* as any positive number, thus we find $\frac{\partial\rho}{\partial t} + \frac{\partial q}{\partial x} = 0$

We may consider $v\ (x,t)$ as a function of ρ, which we will talk about later. Thus, we can rewrite the above formula as $\frac{\partial\rho}{\partial t} + \frac{\partial q}{\partial\rho}\frac{\partial\rho}{\partial x} = 0$

which we may discover $\frac{\partial q}{\partial\rho}$ as a dimension of speed.

As a result, we have $\frac{\partial\rho}{\partial t} + v(\rho)\frac{\partial\rho}{\partial x} = 0$

Before solving the partial differential equation, we need to determine the function $v\ (\rho)$ as a function of ρ, and find the boundary conditions. By viewing the reality, we have discovered that the speed of the cars decreases as the density of cars increases. Hence it is reasonable to use a widely known relationship $v = u_k(1 + k - \frac{\rho}{\rho_{max}})$ where $u_0 = u_k(1+k)$ is the maximum speed of the cars which may be determined by road limit or cars' own functions. As we can see, when $\rho = 0$, the cars reach their maximum speed, and when $\rho = \rho_{max}$, the cars have their minimum speed $u_k k$. The partial differential equation then changes to $\frac{\partial\rho}{\partial t} + (u_0 - \varepsilon\rho)\frac{\partial\rho}{\partial x} = 0$

in which $\frac{u_k}{\rho_{max}} = \varepsilon$. And in the problem, we assume the boundary condition at time 0 as

$$\rho = \begin{cases} \rho_0 & x < 0 \\ \rho_0 + \frac{x_0}{2}\omega & 0 \le x < l \\ \rho_1 & l \le x \end{cases}$$

where $\omega = \frac{2}{l}(\rho_1 - \rho_0)$, in order to make function ρ continuous at $x = l$.

In order to solve the partial differential equation $\frac{\partial\rho}{\partial t} + v(\rho)\frac{\partial\rho}{\partial x} = 0$, we have to introduce the characteristic method. We view $\rho = \rho\ (x,t)$ as a surface in three-dimensional coordinate with axis x, t and ρ. And the characteristic curves $x = x(t)$ represent the curves on which ρ is identical. Thus we can represent ρ as $\rho = \rho(x(t),t)$. By taking derivative of ρ, we can obtain $\frac{d\rho(x(t),t)}{dt} = \frac{\partial\rho}{\partial x}\cdot\frac{\partial x}{\partial t} + \frac{\partial\rho}{\partial t}$

And because on the curves $x = x(t)$, ρ is constant, so that $\frac{d\rho(x(t),t)}{dt} = 0$

As a result, we acquire that $\frac{d\rho(x(t),t)}{dt} = \frac{\partial \rho}{\partial t} + (u_0 - \varepsilon\rho)\frac{\partial \rho}{\partial x}$

By comparing both sides, it is reasonable to get $\frac{dx}{dt} = u_0 - \varepsilon\rho$

And $x = (u_0 - \varepsilon\rho)t + x_0 = vt + x_0$

When $x < 0$ and $x \geq l$, v and ρ are constant. So we only need to consider the interval $0 \leq x < l$. By substituting $\rho = \rho_0 + \frac{x_0}{2}\omega$, we can have the function of v.

$$v = u_0 - \frac{u_k}{\rho_{max}}(\rho_0 + \frac{\omega}{2}x_0)$$

at $t = 0$. And function x is also achievable as

$$x = t\left(u_0 - \frac{\rho_0 u_k}{\rho_{max}}\right) - (\frac{u_k \omega t}{2\rho_{max}} - 1)x_0$$

Thus x_0 can be solved as a function of x and t.

$$x_0 = \frac{t(u_0\rho_{max} - u_k\rho_0) - x\rho_{max}}{\frac{1}{2}u_0\omega t - \rho_{max}}$$

and ρ is calculated as

$$\begin{cases} \rho = \rho_0 + \frac{\omega t(u_0\rho_{max} - u_k\rho_0) - \omega x\rho_{max}}{u_0\omega t - 2\rho_{max}} \\ \omega = \frac{2}{l}(\rho_1 - \rho_0) \end{cases}$$

4.1.2 Conclusions

This result has given us an important conclusion: Once the rate of cars enters the system is given with a linear relationship with distance at the beginning, at any time t, the change in traffic density can be written as a sum of a rational expression of only t and a rational function of both x and t.

This property shows that, at any time, the traffic density will be at the similar frequency (shape), but the enhanced amplitude with the original density, which may be understood as the cause of the flux from the right lane car moving into the left.

By using this model, we can calculate the shortest time for cars to exit the system and the density of car flow at any point of time. We didn’t use this model to solve later questions, because we can only know the behavior of cars but are not able to control it.

4.2 Car-following Model

4.2.1 Car-following Process

On a timeline, all cars enter our system according to Poisson distribution, which is represented as the following: $P(x=k)=\frac{\lambda^k}{k!}\times e^{-\lambda}$, in which, λ represents the expected number of cars entering the system per minute. All cars are moving at its maximum speed v^{max} when $x=-s$ (sm before seeing the sign). Then, for every time interval Δt, we calculate the maximum speed $v_n^a(t)$ a car can accelerate to in one time interval, the minimum speed $v_n^b(t)$ a car can decelerate to in one time interval and the maximum safe speed $v_n^s(t)$ of that car by the following equations, based on the displacement, speed and acceleration (or deceleration) of the cars before this time interval (simulating the reaction time T of human being).(Olstam and Tapani, 2004)

$$v_n^a(t)=v_n(t-T)+2.5\times a^{max}\times T\times(1-\frac{v_n(t-T)}{v^{max}})\times\sqrt{0.025+\frac{v_n(t-T)}{v^{max}}}$$

$$v_n^b(t)=v_n(t-T)-b^{max}\times T$$

$$v_n^s(t)=-b^{max}\times T+\sqrt{T^2\times {b^{max}}^2-2b^{max}(S_n(t-T)-D)+(v_{n-1}(t-T))^2}$$

(derived from:

$$\begin{cases} x_{n-1}=\frac{{v_{n-1}}^2}{2b^{max}}+D \\ x_n=\frac{{v_n}^2}{2b^{max}}+T\times v_n \end{cases} \quad (v^2=u^2+2as)$$

so, $x_{n-1}-x_n=\frac{{v_{n-1}}^2-{v_n}^2}{2b^{max}}+D-T\times v_n=S_n$)

where $D=x_{n-1}(t-T)-x_n(t-T)-l$

and safe distance is calculated by $S_n^p(t)=2v_n(t)+l+k$, based on the 2-second rule of traffic system[3]

The speed of the car in this time interval is obtained by comparing these three values:

a. if $T^2\times {b^{max}}^2-2b^{max}(S_n(t-T)-D)+(v_{n-1}(t-T))^2<0$,

then $v_n(t)=v_n^b(t)$;

b. if $T^2\times {b^{max}}^2-2b^{max}(S_n(t-T)-D)+(v_{n-1}(t-T))^2\geq 0$,

and $v_n^b(t)\leq v_n^s(t)\leq v_n^a(t)$,
then $v_n(t)=v_n^s(t)$;

c. if $T^2\times {b^{max}}^2-2b^{max}(S_n(t-T)-D)+(v_{n-1}(t-T))^2\geq 0$,

and $v_n^s(t)>v_n^a(t)$,
then $v_n(t)=v_n^a(t)$;

d. if $T^2 \times b^{max\,2} - 2b^{max}(S_n(t-T) - D) + (v_{n-1}(t-T))^2 \geq 0$,

and $v_n^s(t) < v_n^b(t)$,
then $v_n(t) = v_n^b(t)$;
We can calculate the displacement, speed and acceleration (or deceleration) of each car after this time interval by the following equations:

$$x_n^p(t) = T \times v_n^p(t-T) + x_n^p(t-T)$$
$$a_n^p(t) = \frac{v_n^p(t) - v_n^p(t-T)}{T}$$

4.2.2 Mergence Process

In our model, we assume the right lane is closed one.

When the mergence of car C_n^r happens, we take the following steps:
1. Take the left back car C_m^l which satisfies $x_m^l(t) = max(x^l(t) | x^l(t) < x_n^r(t))$;
2. Insert a car C_{new}^l in the left lane in front of car C_m^l, set $x_{new}^l(t) = x_n^r(t)$;
3. Delete C_n^r from the right lane after c seconds, where c is a constant. (During the mergence process, the car is considered as the front car of car C_m^l and C_{n-1}^r)

If a car C_n^r in the right lane and its surrounding cars satisfy the conditions written below (to make sure when the car will not crash with the car at its left back position and the car at its left front position when it is at a particular acceleration, which is possible for the car's acceleration limitations):
for any $t \epsilon [0, c]$,
the following inequalities should be satisfied:

$$\begin{cases} x_{m-1}^l(t) - x_{new}^l(t) + \left(v_{m-1}^l(t) - v_{new}^l(t)\right)t - \frac{1}{2}at^2 \geq \frac{v_{new}^l(t)^2}{2b^{max}} \\ x_{new}^l(t) - x_m^l(t) + \left(v_{new}^l(t) - v_m^l(t)\right)t + \frac{1}{2}at^2 \geq \frac{v_m^l(t)^2}{2b^{max}} \\ -b^{max} \leq a \leq a^{max} \end{cases}$$

it is obtainable that, car C_n^r merges into the left lane when

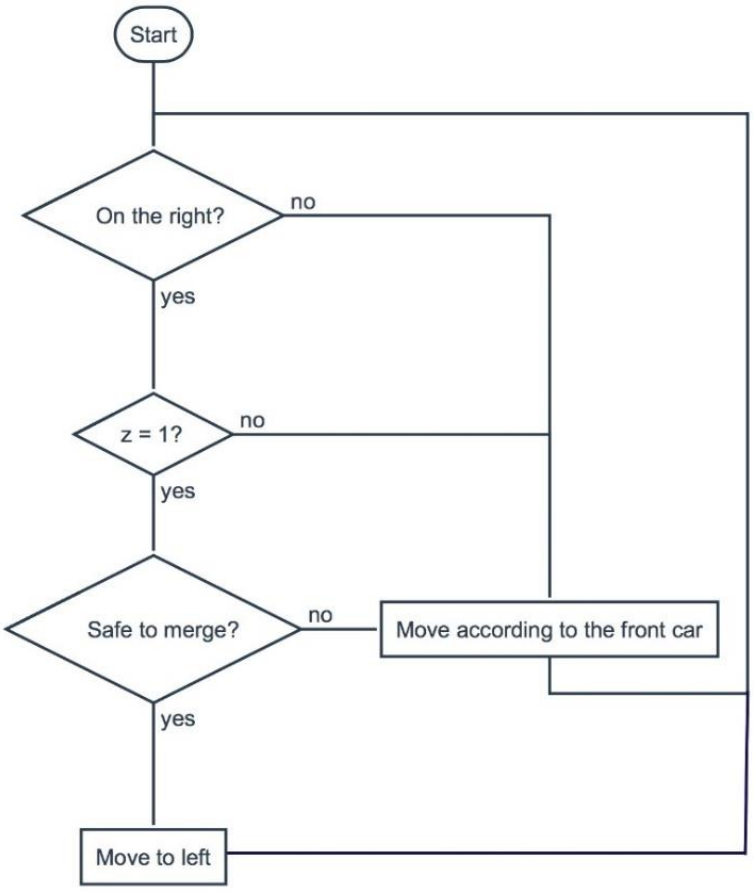
$$\begin{cases} min(a_1, a^{max}) \geq \max(a_2, -b^{max}) \\ Z_n^r(t) = 1 \end{cases}$$

where

$$a_1 = \frac{2\left(x_{m-1}^l(t) - x_{new}^l(t) + \left(v_{m-1}^l(t) - v_{new}^l(t)\right)t - \frac{v_{new}^l(t)^2}{2b^{max}}\right)}{t^2}$$

$$a_2 = \frac{2\left(x_m^l(t) - x_{new}^l(t) + \left(v_m^l(t) - v_{new}^l(t)\right)t - \frac{v_{new}^l(t)^2}{2b^{max}}\right)}{t^2}$$

4.2.3 Flowchart



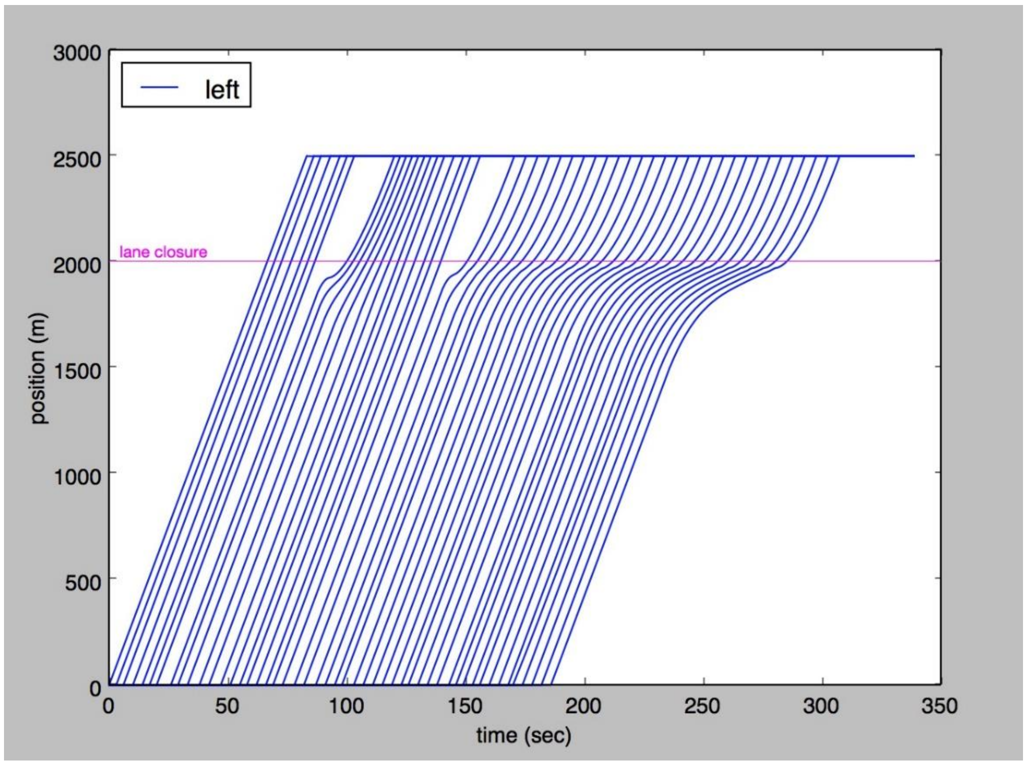
5 Model Testing and Part II (all data is in Appendix A)

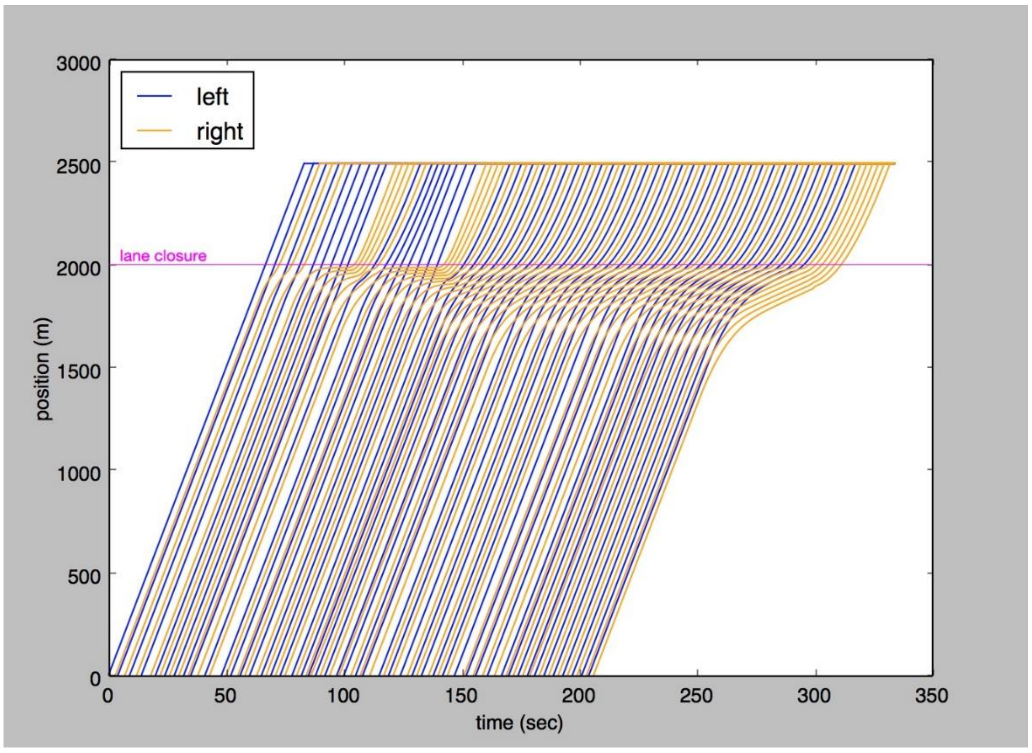
5.1 Model Testing

We test our model with various sets of data, including different input rates, sign distances, speed limits and strategies.

To simulate real-world situation, we use $l = 5.5m$, $a^{max} = b^{max} = 5ms^{-2}$, $L = 2000m$, $T = 0.1s$, $c = 1s$ and $k = 2m$ as default value.[3]

Here is a typical set of data: sign distance = 1600m, speed limit is 65 miles per hour (30m/s), Strategy 2, $\lambda = 26$. When we run the model, we keep tracks of all positions on a timeline. Then, we plot graphs of the positions of cars against time.





In the first graph, the positions of the cars started in the left lanes are recorded, while all the positions of those started in left and right are plotted in the second graph.

According to the data shown, no crash happened even if all cars moved fast and lots of cars chose to merge.

5.2 Traffic Control Strategies

We suggested three strategies to control the traffic:

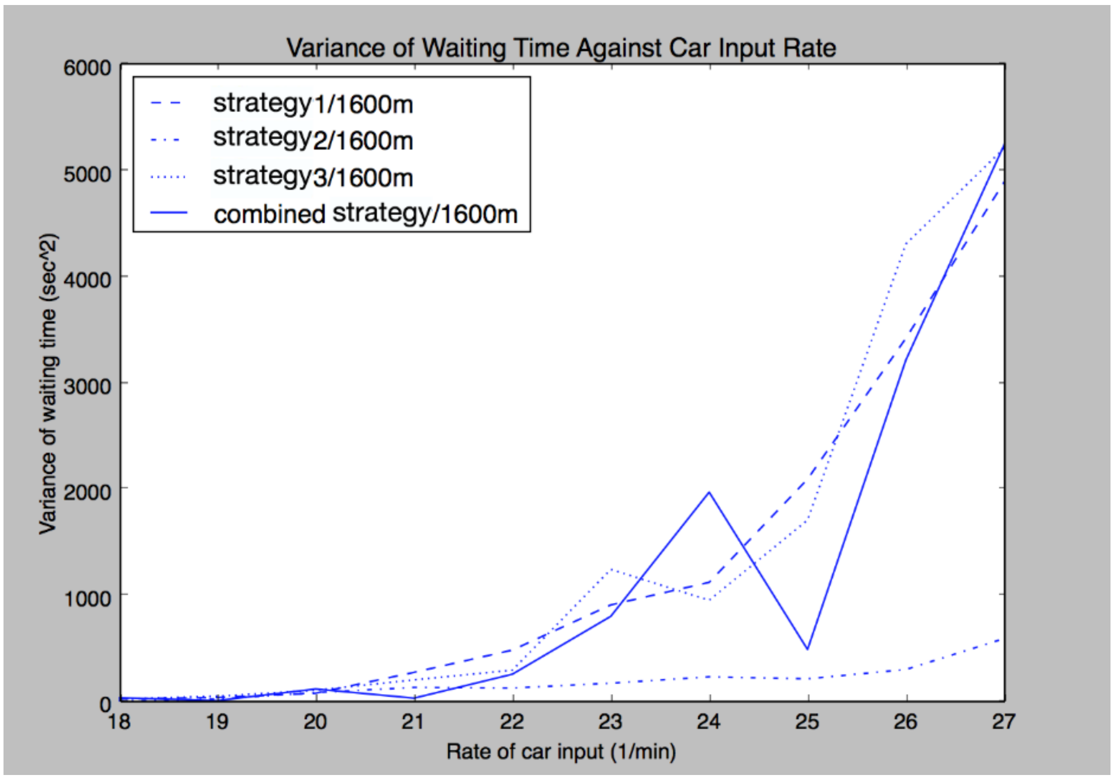
Strategy 1: all cars merge to the left lane in between the sign and the closing lane point as soon as possible;
Strategy 2: all cars merge to the left lane when they are the first on the lane;
Strategy 3: all cars only merge to the left lane when they are moving slower than the car on their left.

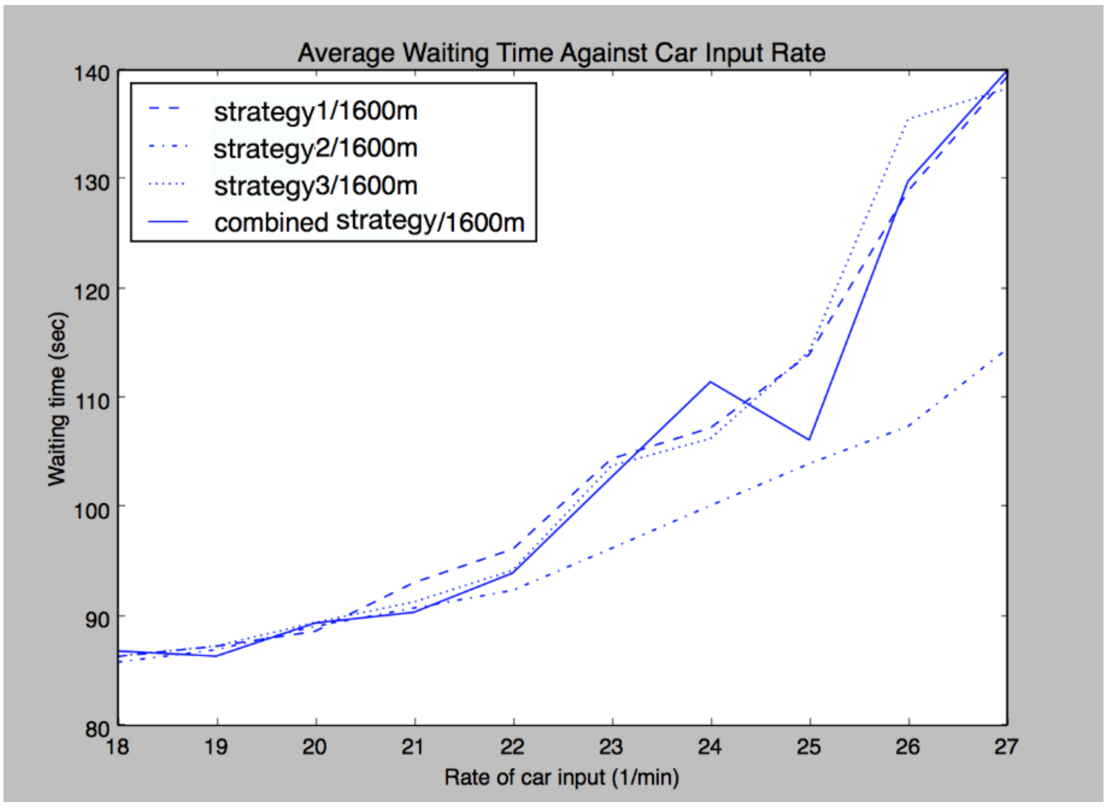
The first strategy is $Z_n^p(t) \equiv 1$;

the second strategy is $Z_n^p(t) = \begin{cases} 0 & (\text{there is at least one car in front of } C_n^p) \\ 1 & (\text{there is no car in front of } C_n^p) \end{cases}$;

the third strategy is $Z_n^p(t) = \begin{cases} 0 & (v_n^r(t) \geq v_{m-1}^l(t)) \\ 1 & (v_n^r(t) < v_{m-1}^l(t)) \end{cases}$.

The graphs below demonstrates that, the use of uniform strategy by all drivers is necessary in road mergence. According to our experiment, if all drivers arbitrarily choose any strategy, the system would be far less efficient or fair than a given uniform strategy.





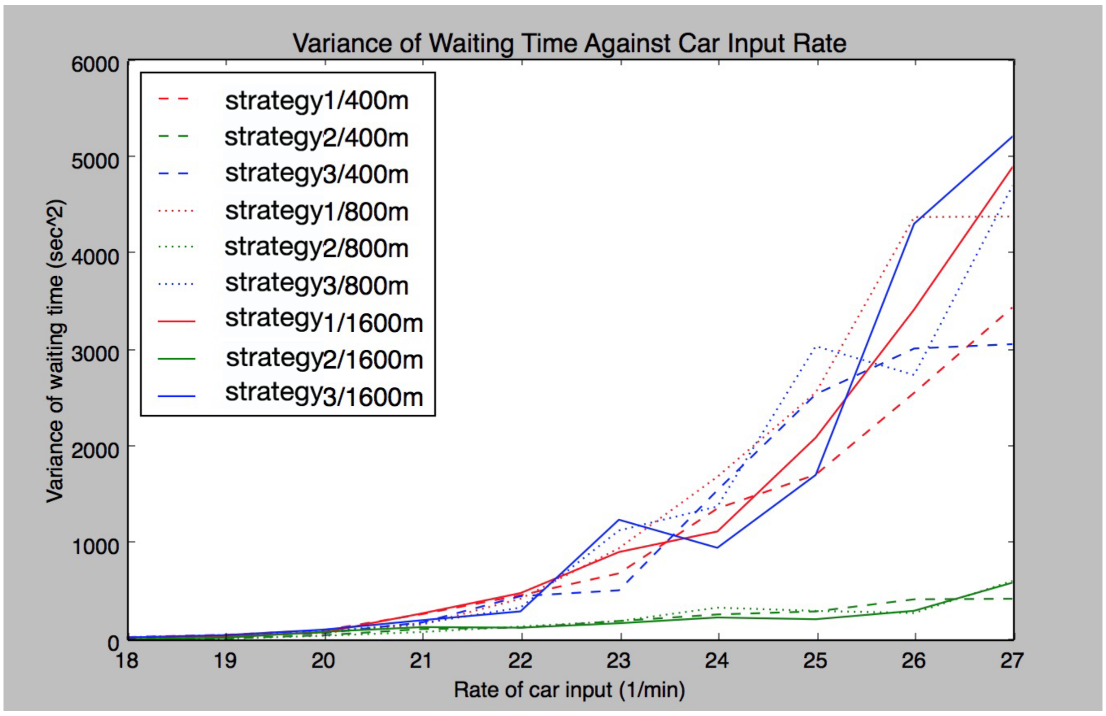
5.3 Fairness and Efficiency

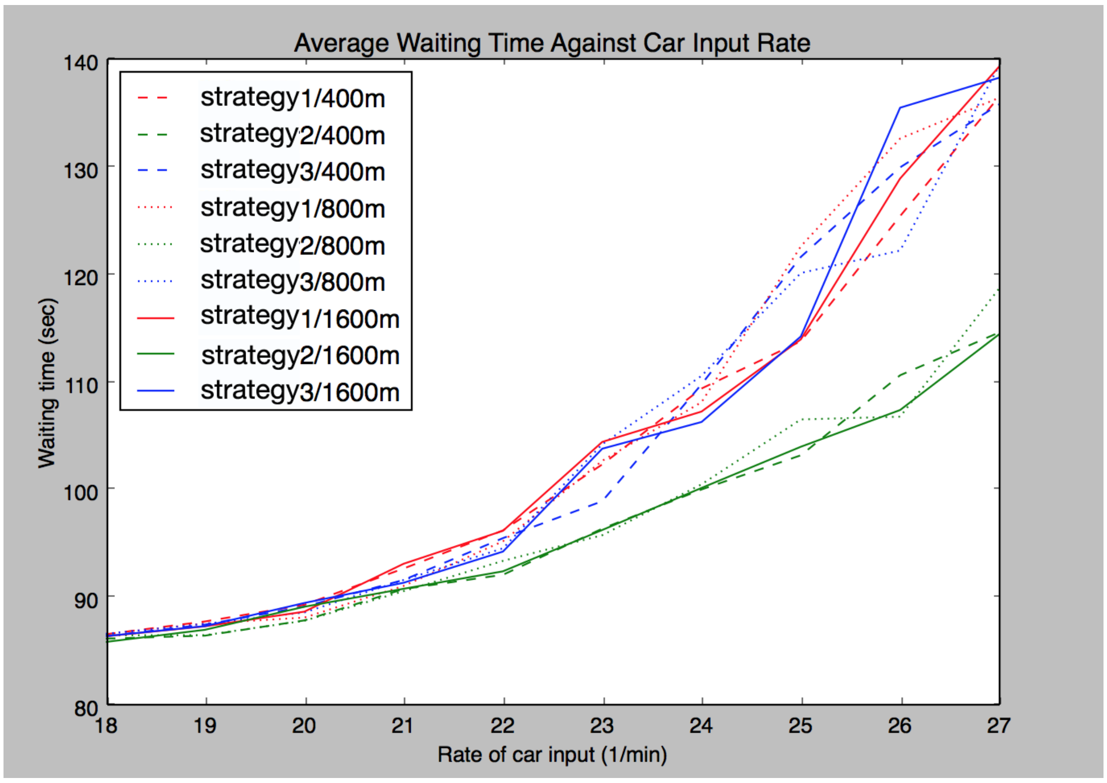
5.3.1 Determination

Fairness can be judged by the variance of waiting time, because small variance of waiting time means that all the drivers spent similar amount of time on this road.
As for efficiency, it is reasonable to be judged by the average of waiting time, because small average of waiting times means that more cars exit the system during a certain period of time.

5.3.2 Two Lanes to One Lane

The speed limitation on the high way is 65 miles per hour (about 30m/s). When λ varies between 18 and 27 (cars per minute), we plot graphs of variance and average of waiting time against λ respectively. From the graphs below we can see that, Strategy 2 always has the best fairness and efficiency.

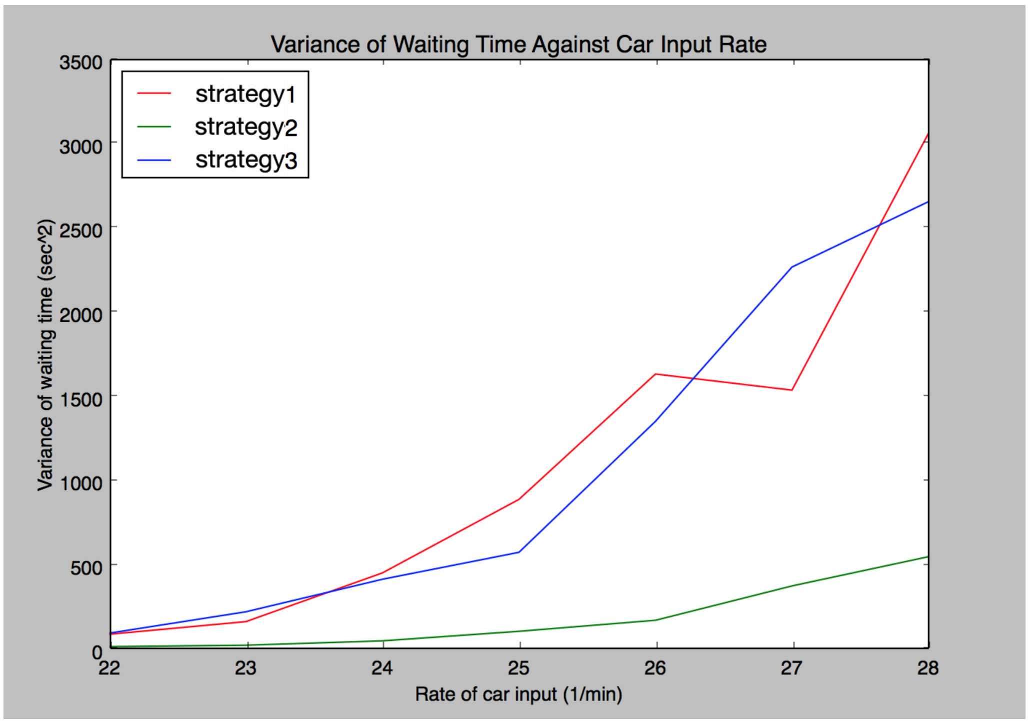


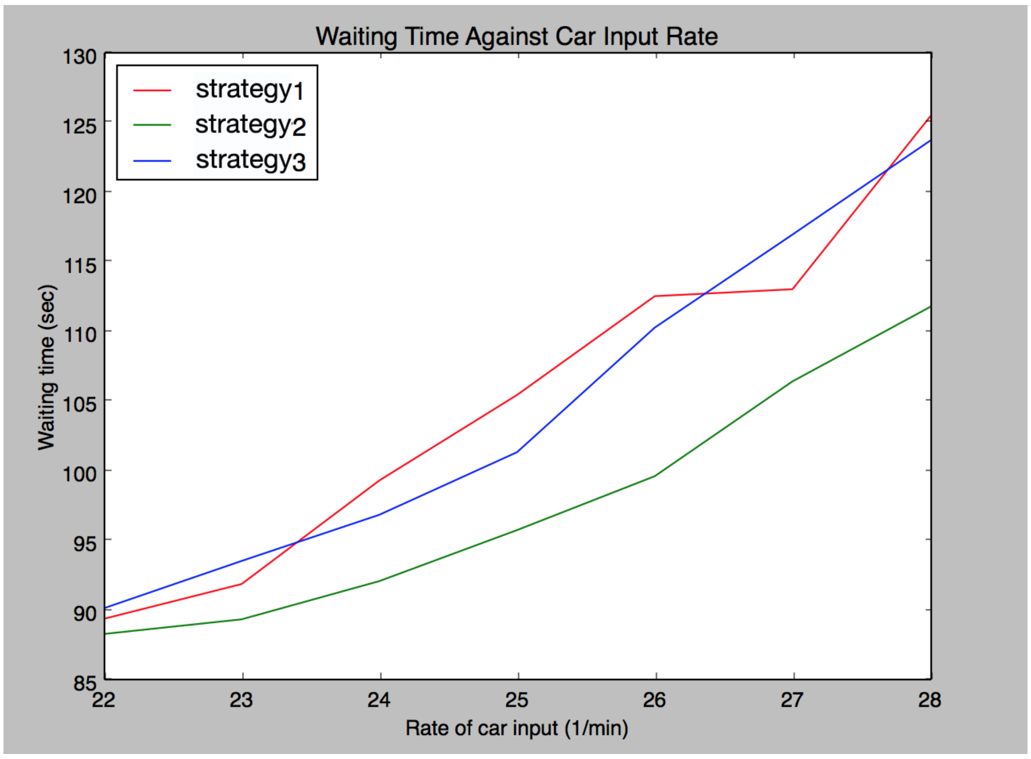


5.3.3 Three Lanes

5.3.3.1 Two Lanes Closed

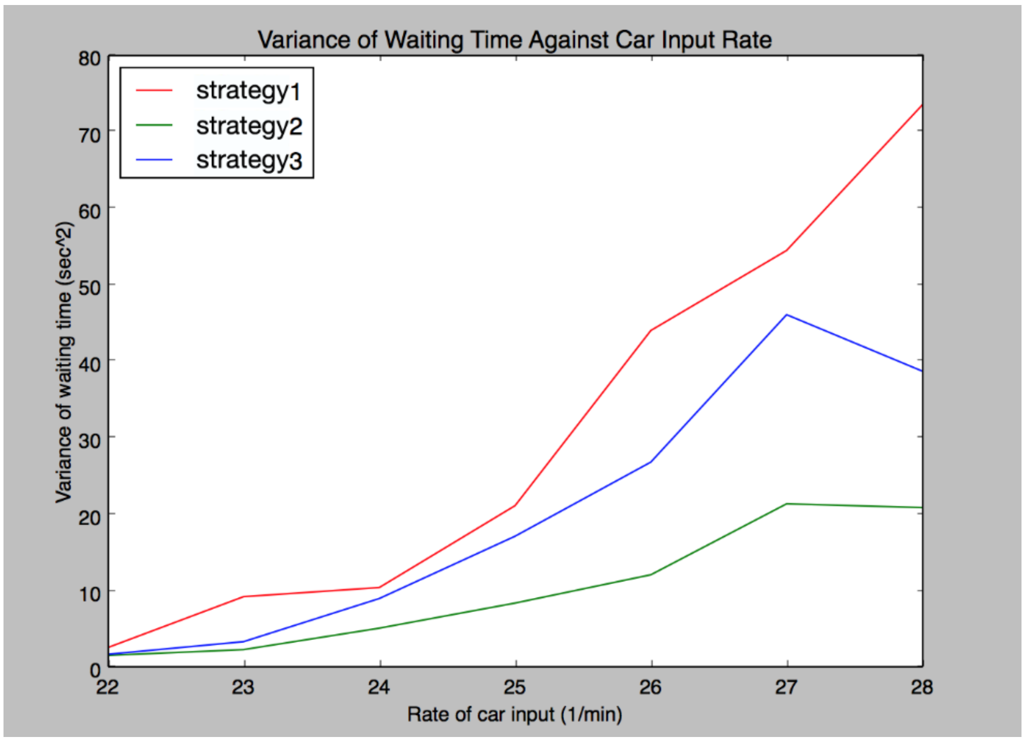
When λ varies between 22 and 28 (cars per minute), we plot graphs of variance and average of waiting time against λ respectively. From the graphs below we can see that, Strategy 2 always has the best fairness and efficiency.

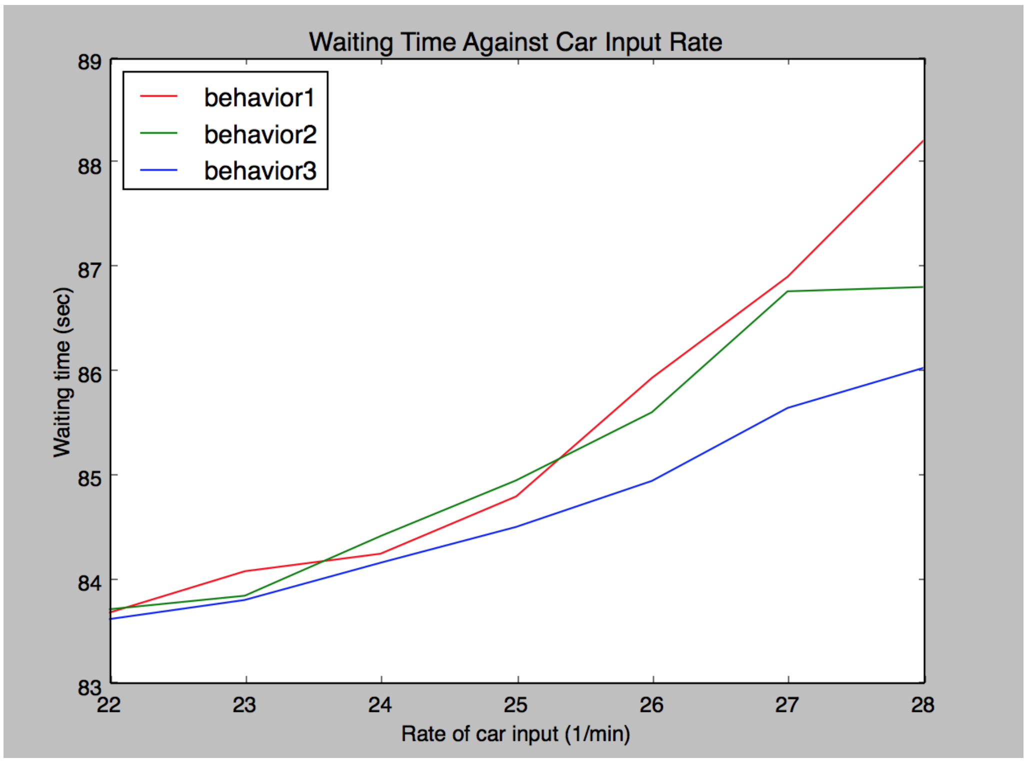




5.3.3.2 One Lane Closed

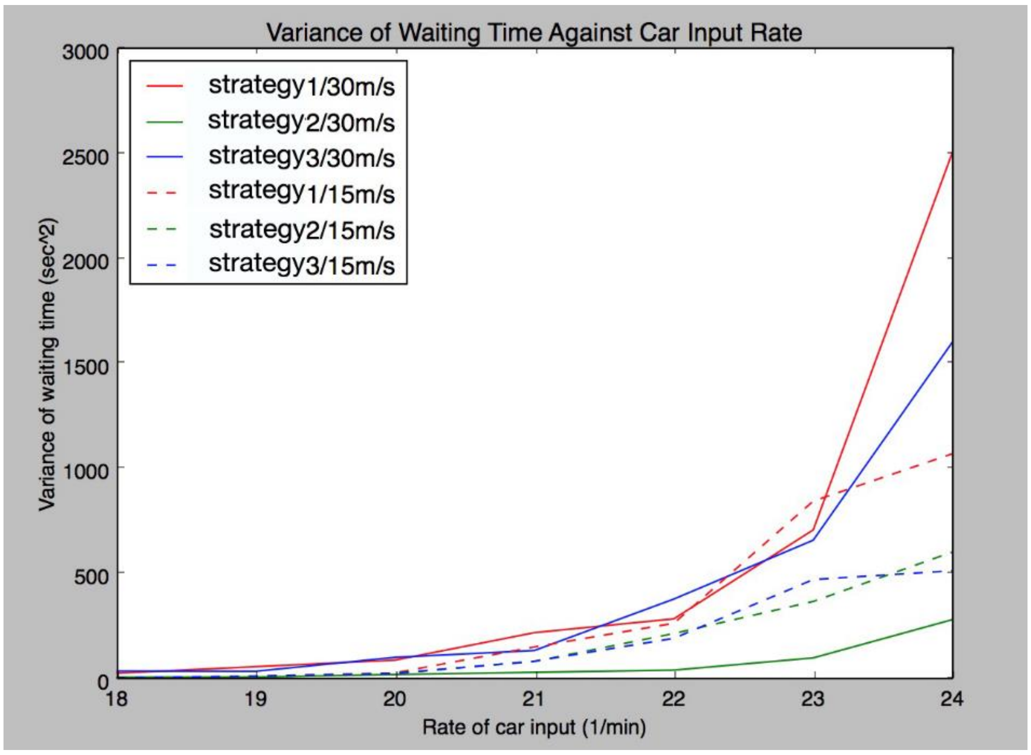
When λ varies between 22 and 28 (cars per minute), we plot graphs of variance and average of waiting time against λ respectively. From the graphs below we can see that, Strategy 2 always has the best fairness, while Strategy 3 has the best efficiency.

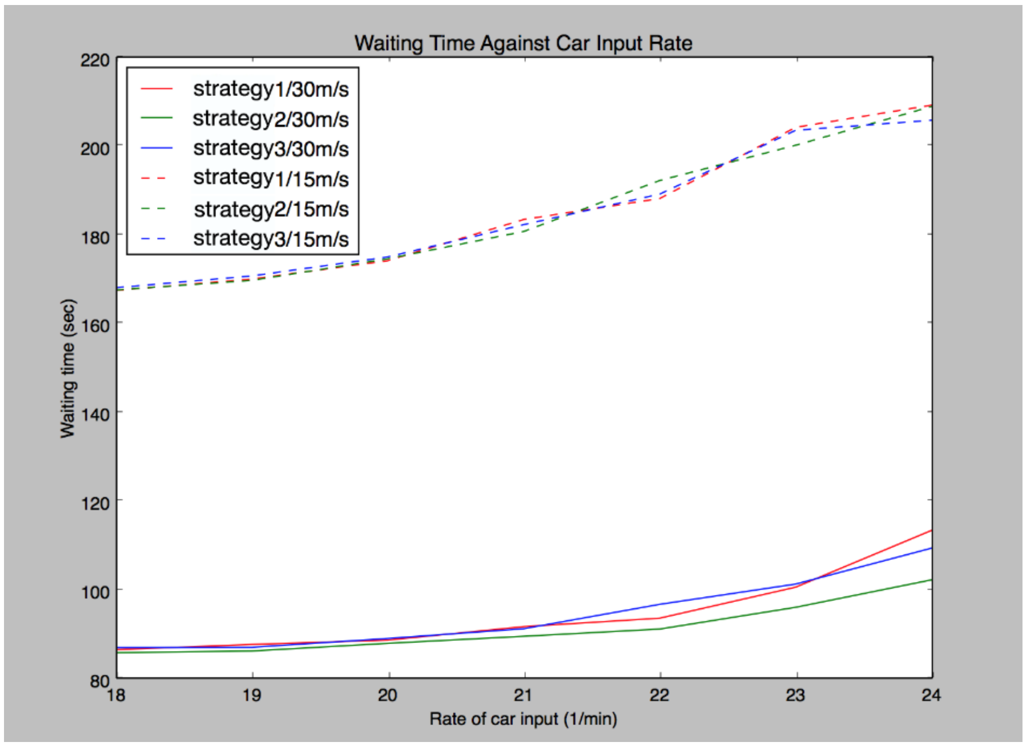




5.3.4 Secondary Road

When the speed limitation of the road is reduced to 35 miles per hour (about 15 m/s), we plot graphs of variance and average of waiting time against λ respectively. We can see from the graph below that the efficiency of the three strategies are similar when λ varies from 18 to 24 (cars per minute), while Strategy 2 & 3 have slightly better fairness than Strategy 1.





6 Robustness Test

For our model, we'd like to analyze the robustness of our model. According to *Statistical Inference* by *George Casella* and *Roger L. Berger*, the robustness of a model is defined by the following:

a. It should have a reasonably good efficiency at the assumed model.

b. It should be robust in the sense that small deviations from the model assumptions should impair the performance only slightly.

c. Somewhat larger deviations from the model should not cause a catastrophe.

As a result, we examine our Car-following model with the three aspects of the definition.

a. In the assumed model, we run the simulation 10 times with same condition settings. Then, we compare the mean of waiting time t_w we get every time.

λ	1	2	3	4	5	6	7	8	9	10
16	25.41	25.24	25.39	25.00	25.41	25.23	25.64	24.89	25.45	24.42
20	26.53	26.23	26.09	27.12	26.53	26.33	26.13	26.29	26.37	26.32
24	30.11	29.89	29.31	29.35	29.54	29.97	29.44	29.76	29.89	29.13

b. With some small deviations of maximum acceleration a^{max} and maximum deceleration b^{max}, we compare the new mean to the original one.

λ	$a^{max} + 10\%$	$a^{max} - 10\%$	$b^{max} + 10\%$	$b^{max} - 10\%$	**Original**
16	25.59	25.62	25.11	25.41	**25.41**
20	26.23	26.49	26.35	26.52	**26.53**
24	28.57	29.48	29.04	30.11	**30.11**

c. With some large deviations of maximum acceleration a^{max} and maximum deceleration b^{max}, we compare the new mean of waiting time t_w to the original one.

λ	$a^{max} + 50\%$	$a^{max} - 50\%$	$b^{max} + 50\%$	$b^{max} - 50\%$	**Original**
16	25.48	26.00	25.07	26.00	**25.41**
20	26.05	28.47	25.56	28.47	**26.53**
24	31.42	32.32	27.47	32.32	**30.11**

We can see from the table that the model is robust.

7 Strengths and Weaknesses

7.1 Fluid Dynamics Traffic Model

7.1.2 Strengths

a. The model changes a discrete problem in reality into a continuous model, which simplifies the problem and makes the result much more concise.

b. The model is adaptable in most of the traffic problems. Because the only difference between each traffic problem is the boundary condition, while the equation is unchanged, thus when the PDE is solvable, the problem is settled.

7.1.3 Weaknesses

a. Because of the continuity of the model, the approach is unable to perfectly mimic what happens when the cars try to emerge.

b. The model does not include the research on each driver's behavior, which means it views the whole traffic as a unity and neglects the effect of a single driver's action.

7.2 Car-following Model

7.2.1 Strengths

a. The model includes various limitations about safety, so it's impossible for any cars to crash in any situation.

b. The model determines accurately the behavior of each individual, so the process is close enough to the reality.

c. The model takes full statistics on each individual and the overall system, so after running the model, all information needed about the system can be obtained.

d. The model is robust enough when the given conditions change.

e. The model is capable of adapting to all kinds of data inputs.

7.2.2 Weaknesses

a. It takes a long time to calculate the results of every single vehicle.

b. We assume all drivers have absolutely accurate estimation of all the other vehicles. This is not true in the reality.

Reference List

[1] Casella G. and Berger R. L. Statistical Inference.[M] Cengage Learning. 2001.

[2] Olstam J. and Tapani A. Comparison of Car-following Models [J] VTI meddelande. 2004

[3] Physics – Car Breaking Distances [OL] [2000-3-22]
http://www.batesville.k12.in.us/physics/phynet/mechanics/kinematics/BrakingDistData.html

[4] The two-second rules of the road [OL] Road Safety Authority. 2014.
http://www.rulesoftheroad.ie/rules-for-driving/speed-limits/speed-limits_2-second-rule.html

Guidelines of mergence

You might meet the following situations of lane closure. Here is the guide to choose mergence strategy to make sure the fairness and efficiency of your driving experience.

1. On a two-lane highway, the lanes merge into one on the left side. If you are on the right lane, always remember to stay on the right lane and keep moving. Merge only if you are the first car closing into the lane closure.

2. On a three-lane highway, the lanes merge into two on the left side. If you are either on the right lane or the middle lane, you are supposed to merge to the left only when the cars on your left is faster than you.

3. On a three-lane highway, the lanes merge into one on the left most lane. If you are on the closing lanes, always remember to stay straight and keep moving. Merge only if you are the first in your lane.

4. On a secondary road, the two lanes merge into one lane on the left. If you are on the right lane, you are supposed to merge only when the car on the left lane is faster than you.

Dear the Director of the Department of Transportation,

Road mergence has always been a problem disturbing the drivers as well as the Department of Transportation. As a result, we have studied on the issue of road mergence and determine the best strategies that should be carried out on the road.

Our model is a Car-following model based on computer simulation, in which all cars determine their speed according to the cars in front of them. We judged the results by efficiency and fairness. Such simulation provides realistic and comprehensive data on each car in the system and describes the actions of drivers well.

We did experiments on three suggested strategies to control the traffic: all cars merge to the open lane as soon as possible; all cars merge to the open lane when they are the first on the lane; and all cars only merge to the open lane when they are moving faster than the car on their left.

According to our results, we highly suggest the following strategies to control the traffic: On a highway with two lanes merging into one or three lanes merging into one, all cars are supposed to merge only if they are the first in the lane. Such strategy can make sure that both lanes are fully used, resulting the improvement in efficiency. Also, it ensures fairness because the time spent by each car on both lanes is approximately the same. To convince more people to behave in such pattern, there are two measures the Department of Transportation could take. Firstly, a sign saying “keep in lane until the junction” can be set on lanes. Secondly, lanes can be separated before cars reach the lane closure, so no cars merge ahead of time.

When speed limit is 35 mph and when highways have one lane closed among three, the recommended strategy is to merge when cars on the open lane is faster than the merging car. Such strategy makes sure that all cars on lanes can be balanced automatically.

The use of uniform strategy by all drivers is necessary in road mergence. According to our experiment, if all drivers choose arbitrarily any strategy, the system would be far less efficient or fair than a given uniform strategy. This clearly demonstrates the necessity of proper laws for the correct merging strategy.

Yours sincerely,
Team 5813

Appendix A (Data)

Data with 2-lane road

Data with speed limit 35 miles per hour, with strategy 1, 2, and 3 respectively, sign distance 1600m
[[[28.029441831572637, 86.56755135330916], [9.17904493370923, 85.8959220315748], [38.4967616679815, 87.04286721072722]], [[59.424881894353575, 87.75329014500616], [9.656185306145437, 86.27188578354944], [37.35274378655112, 87.10306922026744]], [[90.4973620187777, 88.72992796072938], [22.14053109454482, 88.00230732542573], [103.7361912263676, 89.09525771384969]], [[221.34953163940827, 91.76449783681505], [32.397133694017924, 89.58920565823284], [136.42018506604754, 91.29955670014006]], [[287.13076848715025, 93.67068750415336], [42.85177050110432, 91.22047056568725], [381.35856165940834, 96.79237182674427]], [[710.7245404007466, 100.66262981632288], [101.10021173630052, 96.12360311502013], [661.7681605168257, 101.36133464143859]], [[2510.6562360388916, 113.47276426264621], [283.7099413409021, 102.31264283037976], [1606.3199764999194, 109.46489355586141]]]

Data with speed limit 65 miles per hour, with strategy 1, 2, and 3 respectively, sign distance 1600m
[[[29.574522102817546, 86.42008045494138], [9.199564185647452, 85.89578270845553], [28.435746032911993, 86.43550552728888]], [[41.26795535736941, 87.34972412666187], [25.778213566806592, 87.02833116478386], [52.987058220009864, 87.36952600306472]], [[81.50100609941606, 88.7127969015611], [87.6449939282846, 89.16832182284999], [109.60557704155576, 89.5197631484587]], [[278.09285093822257, 93.16564398047942], [136.08250128954813, 90.84047427929943], [207.26759099140796, 91.39944904575307]], [[488.11208695604347, 96.22864123852382], [130.29439649491843, 92.46237055351035], [299.15679635790264, 94.29722161487031]], [[913.7760789656144, 104.49132495848862], [176.1127551746533, 96.30291751479902], [1248.5621696652984, 103.85980304191666]], [[1126.388274953732, 107.3145891348172], [235.81037927686498, 100.21389474694864], [957.6250104079261, 106.3432007165439]], [[2099.045323774918, 114.02338769186699], [217.52535443125674, 104.0415502357163], [1713.0114818593224, 114.29395640576419]], [[3427.224521855147, 128.979268656256], [305.14614692844674, 107.45698241747502], [4315.811960570807, 135.55788490402384]], [[4901.870172933827, 139.41299563513928], [597.02771507104, 114.5188880677584], [5219.59350734593, 138.35039680399697]]]

Data with speed limit 65 miles per hour, with strategy 1, 2, and 3 respectively, sign distance 800
[[[31.983784728025388, 86.64057392468453], [20.12643379804579, 86.51651626823293], [30.08717034449192, 86.6518541669819]], [[53.674463700297984, 87.60789015183386], [17.305769080514317, 86.5262395836481], [50.02392891406415, 87.58068554480884]], [[76.68911567141173, 88.16237579654353], [45.81836420309428, 87.86416574896579], [93.75073473836568, 88.6357712219722]], [[161.30759276186404, 91.10634277517877], [85.2911600181693, 90.62055487517736], [174.71038436306193, 91.6572016565408]], [[429.8709597109805, 95.25380323982502], [142.40232984898662, 93.43323806266636], [338.0301979036075, 94.60091115061812]], [[952.9785395547949, 102.73800341219075], [197.981065998182, 95.81978761538667], [1136.864052767499, 104.27018562261306]], [[1693.317400922228, 108.18648209237054], [337.5200591179455, 100.54945884850706],

[1385.5527146660538, 110.63518648947941]], [[2566.708938745858, 122.67294979561291], [303.80807006281356, 106.58112818650352], [3046.492304006057, 120.19045933981435]], [[4384.106076448309, 132.69533658585868], [282.1010545018589, 106.83609387426058], [2748.333622087597, 122.28155627248218]], [[4389.962451732233, 136.48952290029564], [614.9756204828111, 118.79377652723481], [4711.324043845021, 139.52299331296007]]]

Data with speed limit 65 miles per hour, with strategy 1, 2, and 3 respectively, sign distance 400m
[[[32.46782874442806, 86.61051038185904], [13.680098356671314, 86.2036130407702], [24.72149673043871, 86.44405314964706]], [[56.266330867268096, 87.80088373687856], [18.435026366071153, 86.48351231092067], [44.5221697084754, 87.50621370956648]], [[103.27642020923199, 89.35087723944213], [52.01151536411267, 87.89743179713678], [74.05010829008998, 89.18190111673741]], [[270.5182670506091, 92.72355358757194], [116.21379666891653, 90.81511209920049], [184.3218538396477, 91.66087877328223]], [[465.67345522351235, 96.26171174494796], [130.02040325570215, 92.13853702059329], [456.67636228955564, 95.54667169387342]], [[692.5504212725116, 102.39768453602737], [199.18142222782745, 96.42702026886708], [517.0679194112157, 98.97998664309893]], [[1365.155174693499, 109.4546117230232], [267.01477267377334, 100.07042686366626], [1553.3194835182771, 109.84778069015256]], [[1715.7907987078083, 113.8499828189575], [298.73555194932044, 103.24651578879403], [2548.8228213231714, 121.68826789379406]], [[2563.384806010893, 125.48625479124425], [423.8396361532003, 110.68771490997912], [3022.304537655921, 130.0134201304918]], [[3448.793659623406, 136.6168024138739], [429.6214048602891, 114.67705086855474], [3067.7855409995054, 135.8898117726157]]]

Data with speed limit 65 miles per hour, with strategy 1, 2, and 3 all combined, sign distance 1600m
[[39.33375595026364, 86.91277476953931], [11.884178469648804, 86.44071034961722], [121.41561764339534, 89.47210650656436], [35.0410602260479, 90.45323735284583], [260.9358830309301, 94.04596429867408], [806.2772408410964, 102.8217122584975], [1976.3829354140169, 111.53258061744123], [493.94885645215174, 106.21087580014478], [3222.3626972861916, 129.88166909311516], [5252.662017101646, 139.9662419148887]]

Data with speed limit 65 miles per hour, with strategy 1, 2, and 3 respectively, sign distance 1600m, merge into one lane
[[[91.4178175656691, 89.41859914525318], [17.98414922253638, 88.33043974372977], [97.92488512338765, 90.17734786357593]], [[166.25200377646635, 91.91438089400435], [26.425050937979904, 89.38098076955593], [224.7332155243705, 93.5740909781674]], [[455.40218989637185, 99.35183855973763], [52.15655045758054, 92.12582756615363], [417.69660016258206, 96.88820458412802]], [[891.5091542116231, 105.49289161844044], [108.82507679481088, 95.78922991407164], [577.4899202105573, 101.38317121757471]], [[1635.571793822329, 112.58419686623395], [174.29559135821802, 99.66010098377146], [1354.6901311502559, 110.32986836592724]], [[1539.8480248965816, 113.08422338290659], [378.06669643915336, 106.4683498168516], [2270.1980983066383, 117.0098407264996]], [[3063.6305464902075, 125.48767462838744], [551.5395776972989, 111.81667125913035], [2658.4370712639475, 123.75218116738401]]]

Data with speed limit 65 miles per hour, with strategy 1, 2, and 3 respectively, sign distance 1600m, merge into two lanes.

[[[2.668078174069568, 83.69013333329163], [1.6482890648602428, 83.72274830929325], [1.7916363206920487, 83.62816111141535]], [[9.326557228220029, 84.08786282081871], [2.4065461467722034, 83.85290128235633], [3.4445144118265083, 83.8123589746639]], [[10.520089724806603, 84.25570384498853], [5.210477668905876, 84.42455365499178], [9.083984561998289, 84.16908252489102]], [[21.220948375497805, 84.80527230917157], [8.489076355870052, 84.95931618381319], [17.228525339665357, 84.51303867737552]], [[44.111759236674274, 85.94175512952754], [12.188173995459795, 85.61249847835036], [26.91419786071114, 84.95412197725355]], [[54.5750201182268, 86.91220458427318], [21.463857411741024, 86.77253645802088], [46.179514983270465, 85.65401684716696]], [[73.60095790553135, 88.21669573648488], [20.965633485704156, 86.81356805201165], [38.79231518459767, 86.03772824176932]]]

Appendix B (Code)

Phython code (two lanes merge to one):

```
def probabilityCalculation(x,desiredx):      #x is defined as how many cars per minute
    if x==0:
        return math.e**(-desiredx)
    prob = float(probabilityCalculation(x-1,desiredx))*float(desiredx)/float(x)
    return prob

def getinstantinputrate():
    p = []
    for i in range(0,int(2*carspermin+1)):
        p.append(probabilityCalculation(i,carspermin))
        if (i>=1):
            p[i] = p[i]+p[i-1]
    a = random.random()
    while a>=p[-1]:
        a = random.random()
    for i in range(2,int(2*carspermin+1)):
        if a<p[i]:
            return i-1;

def instantinputrate_insec():
    a = getinstantinputrate()
    return int(60/a)

class CAR:
    carlength = 5.5
    deceleration = 5
    acceleration = 5

    def findleftbackcarindex(self):
        i = 0
        found = False
        while i < len(self.road.carsleft):
            if self.road.carsleft[i].position[-1] < self.position[-1]:
                found = True
                break
            i += 1
        if found:
            return i
        else:
            return -1
```

```
def __init__(self,v,x,road,plannumber,start):
        self.velocity = [v]
        self.safespeed = []
        self.position = [x]
        self.road = road
        self.plannumber = plannumber
        self.starttime = start
        self.endtime = start
        if plannumber==0:
                self.z = -1
        else:
                self.z = 0
        self.readytodeletion = False
        self.deletionCount = 0
        #self.deceleration = [0]

def findfrontcar(self):    #return the car object in front of self
        if self in self.road.carsleft:
                for i in range(1,len(self.road.carsleft)):
                        if self == self.road.carsleft[i]:
                                return self.road.carsleft[i-1]
        else:
                if self == self.road.carsright[0]:
                        virp = self.road.rightlength+self.carlength
                        virtualCar = CAR(0,virp,self.road,0,timeLine)
                        virtualCar.velocity=[0,0,0,0,0,0,0,0,0,0,0]
                        virtualCar.position=[virp,virp,virp,virp,virp,virp]
                        return virtualCar
                for i in range(1,len(self.road.carsright)):
                        if self == self.road.carsright[i]:
                                return self.road.carsright[i-1]



def update(self):
        if self.readytodeletion:
                if self.deletionCount>=(mergeTime/refreshRate):
                        if self in self.road.carsright:
                                self.road.carsright.remove(self)
                                return 1
                self.deletionCount+=1
        if self in self.road.carsright:
            self.updateZ()
        frontv = self.findfrontcar().velocity[-1-T]
        frontp = self.findfrontcar().position[-1-T]
        xnew = self.position[-1]+self.velocity[-1]*refreshRate
        vpre = self.velocity[-1]
        ppre = self.position[-1]
        va          =          vpre          +          2.5*self.acceleration*refreshRate*(1-
vpre/vdesire)*math.sqrt(0.025+vpre/vdesire)
        d = self.deceleration
        #d = 5
        t = refreshRate
        Sn = 2*vpre+self.carlength+w
        #vb = d*t+math.sqrt(d*t*d*t-d*(2*(frontp-Sn-ppre)-vpre*refreshRate-frontv*frontv/d))
#FREAKING AWESOME MODEL FAILED
        #vb = vpre - d*refreshRate
        #print Sn,frontp,ppre,frontv
        if (t*t-2*(Sn)/d+2*(frontp-ppre-self.carlength)/d+frontv*frontv/d/d)>=0:
            vb          =          d*(math.sqrt(t*t-2*(Sn)/d+2*(frontp-ppre-
self.carlength)/d+frontv*frontv/d/d)-t)
        else:
            vb = vpre - d*refreshRate
        if min(va,vb)==vb:
            if vb<vpre-d*refreshRate:
                vb = vpre-d*refreshRate
        self.position.append(xnew)
        self.safespeed.append(vb)
        if min(va,vb)<0:
            self.velocity.append(0)
        else:
            self.velocity.append(min(va,vb))


    def updateFirstguy(self):
        xnew = self.position[-1]+self.velocity[-1]*refreshRate
        self.position.append(xnew)
        vmax = vdesire
        v = self.velocity[-1]+self.acceleration*refreshRate
        self.velocity.append(min(v,vmax))
        #self.deceleration.append(-(self.velocity[-1]-self.velocity[-2])/refreshRate)

    def updateZ(self):
        if self.readytodeletion:
            self.z = 0
            return
        if self.position[-1]>=self.road.signPos:
            if self.plannumber == 1:
                self.z = 1
            elif self.plannumber == 2:
                if self == self.road.carsright[0]:
                    self.z = 1
                else:
                    self.z = 0
            elif self.plannumber == 3:
                if self.findleftbackcarindex()==-1:
                    self.z = 1
                    return
                leftbackv = self.road.carsleft[self.findleftbackcarindex()].velocity[-1]
                selfv = self.velocity[-1]
                if leftbackv<=selfv:
                    self.z = 0
                else:
                    self.z = 1
        else:
            self.z = 0

    def whetherMerge(self):
        previousIndex = self.findleftbackcarindex()
        if previousIndex == -1:
            if len(self.road.carsleft) == 0:
                return True
            else:
                frontLeftCar = self.road.carsleft[-1]
                t = 0.1
                allowed = True
                while t < mergeTime:
                    if                    min(2*(frontLeftCar.position[-1]-self.position[-
1]+frontLeftCar.velocity[-1]*t-self.velocity[-1]*t-(self.velocity[-
1]**2/2/self.deceleration+self.carlength+w))/t**2,self.acceleration) < -self.deceleration:
                        allowed = False
                        break
                    t += 0.2
                return allowed
        else:
            if len(self.road.carsleft) == 1:
                backLeftCar = self.road.carsleft[previousIndex]
                t = 0.1
                allowed = True
                while t < mergeTime:
                    if          max(2*(backLeftCar.position[-1]-self.position[-1]-self.velocity[-
1]*t+backLeftCar.velocity[-1]*t+backLeftCar.velocity[-1]**2/2/self.deceleration
+self.carlength+w)/t**2,-self.deceleration) > self.acceleration:
                        allowed = False
                    t += 0.2
                return allowed
            else:
                backLeftCar = self.road.carsleft[previousIndex]
                frontLeftCar = self.road.carsleft[previousIndex-1]
                t = 0.1
                allowed = True
                while t < mergeTime:
                    k     =     max(2*(backLeftCar.position[-1]-self.position[-1]-self.velocity[-
1]*t+backLeftCar.velocity[-1]*t+backLeftCar.velocity[-1]**2/2/self.deceleration
+self.carlength+w)/t**2,-self.deceleration)
                    j            =            min(2*(frontLeftCar.position[-1]-self.position[-
1]+frontLeftCar.velocity[-1]*t-self.velocity[-1]*t-(self.velocity[-
1]**2/2/self.deceleration+self.carlength+w))/t**2,self.acceleration)
                    if k>j:
                        allowed = False
                    t += 0.2
                return allowed

class ROAD:
    carsleft = []
    carsright = []
    def __init__(self,l,r,lmd):
        self.leftlength = l
        self.rightlength = r
        self.carspermin = lmd
        self.signPos = self.rightlength-signLength

    def updateleftcars(self):
        if len(self.carsleft)==0:
            return
        self.carsleft[0].updateFirstguy()
        if self.carsleft[0].position[-1]>self.leftlength:
            self.carsleft[0].endtime = timeLine
            stats.append(copy.deepcopy(self.carsleft[0]))
            self.carsleft.remove(self.carsleft[0])
            if len(self.carsleft)==0:
                return
            self.carsleft[0].updateFirstguy()
        for i in range(1,len(self.carsleft)):
                self.carsleft[i].update()
    def updaterightcars(self):
        i = 0
        while i<len(self.carsright):
            if self.carsright[i].whetherMerge() and self.carsright[i].z==1:
                leftbackindex = self.carsright[i].findleftbackcarindex()
                copycar = copy.deepcopy(self.carsright[i])
                copycar.plannumber = 0
                copycar.z = -1
                if leftbackindex != -1:
                    self.carsleft.insert(leftbackindex,copycar)
                else:
                    self.carsleft.append(copycar)
                self.carsright[i].readytodeletion = True
            if self.carsright[i].update()==1:
                continue
            i+=1


    def rollover(self):    #in second
        loopCount = int(instantinputrate_insec()/refreshRate)
        global timeLine
        for i in range(0,loopCount):
            self.updaterightcars()
            self.updateleftcars()
            timeLine += refreshRate
            for j in range(1,len(self.carsleft)):
                if                  self.carsleft[j-1].position[-1]-self.carsleft[j].position[-
1]<=self.carsleft[j].carlength:
                    print "crash"
                    #err = 1/0
            for j in range(1,len(self.carsright)):
                if                  self.carsright[j-1].position[-1]-self.carsright[j].position[-
1]<=self.carsright[j].carlength:
                    print "crash"
                    #err = 1/0

    def insertAcarleft(self):
        self.carsleft.append(CAR(vdesire,0,self,0,timeLine))

    def insertAcarright(self):
        self.carsright.append(CAR(vdesire,0,self,universalplannum,timeLine))
```