GENETIC FUZZY LOGIC APPROACH TO LOCAL RAMP METERING CONTROL USING MICROSCOPIC TRAFFIC SIMULATION

A thesis presented in partial fulfillment of the
requirements for the degree of

Master of Engineering
in
Mechatronics

at
Massey University,
Auckland, New Zealand

Yu Xue Feng

June 2009

Abstract

Ramp metering, one of the most effective solutions for improving motorway traffic flows, is playing increasingly important role in traffic management systems. Because of its capability to handle nonlinear and non-stationary problems, fuzzy logic based ramp metering algorithms have been always considered as an extremely suitable control measures to handle a complex nonlinear traffic system. This thesis proposes a genetic fuzzy approach to design a traffic-responsive ramp control algorithm for an isolated on-ramp. For a local ramp meter algorithm, the problem could be described as the inflow optimization of on-ramp, based on the evaluation of motorway traffic condition. If the inflow of on-ramp is considered as the decision variable, the ramp control problem could be treated as a nonlinear optimization problem of maximizing the evaluation function. The adaptive genetic fuzzy approach is actually a control approach to maximize the inflow of on-ramp under the restriction of evaluation function.

In this thesis, a well-known fuzzy logic based ramp metering algorithms developed by Bogenberger is introduced and implemented with an on-ramp congestion model of Constellation Drive Interchange in a stochastic microscopic traffic simulator, Aimsun. To improve the performance of fuzzy control system, genetic algorithm is applied to tune the parameterized membership function of each fuzzy input to maintain the flow density of motorway blow the estimated congestion density. The performances of the genetic fuzzy logic control ramp metering are compared with FLC (fuzzy logic control) ramp metering by means of the percentage change of TTT (Total Travel Time) based on no control condition in Aimsun. The simulation results show the genetic fuzzy ramp metering has a more significant improvement on TTT and more strong stability to maintain system flow density than FLC ramp metering.

Acknowledgements

I would like to thank all my supervisors, especially Prof. Peter Xu and Dr. Frakhul Alam, for their support throughout the duration of the project. Without their guidance and persistent help， this thesis would not have been possible. Thanks also to Dr. Johan Potgieter and Dr. Clara Fang for their suggestions, patience and encouragement, which makes me finally brave enough to start this academic journey from a very beginning level.

I would also like to thank my friends Ben Lin and Van Cao for their advices and help. The spectacle of we collecting field data in the rain would be the most precious picture in my memory.

Finally, I want to thank my parents:

谢谢你们在我辞掉工作后的支持，谢谢你们永远尊重我选择生活的权利。

TABLE OF CONTENTS

LIST OF FIGURE

List of tables

Chapter 1 Introduction

1.1 Motorway Congestion

Originally, motorways are designed for long distance travel between cities, and high speed transportation is possible without the disturbance of local traffic. However, during the past four decades, when the increasingly people move to suburban and are more depend on automobiles, motorways are constructed for both intercity travel and commuter traffic especially when surface streets can not satisfy the growth of traffic demand. Therefore, congestion problem that only exists on surface streets appears on motorways now.

In order to understand how motorway congestion forms and how it affects traffic situation, it is important to understand the theory of traffic flow. In this thesis, Greenshield's Model is employed to explain the above questions [1].

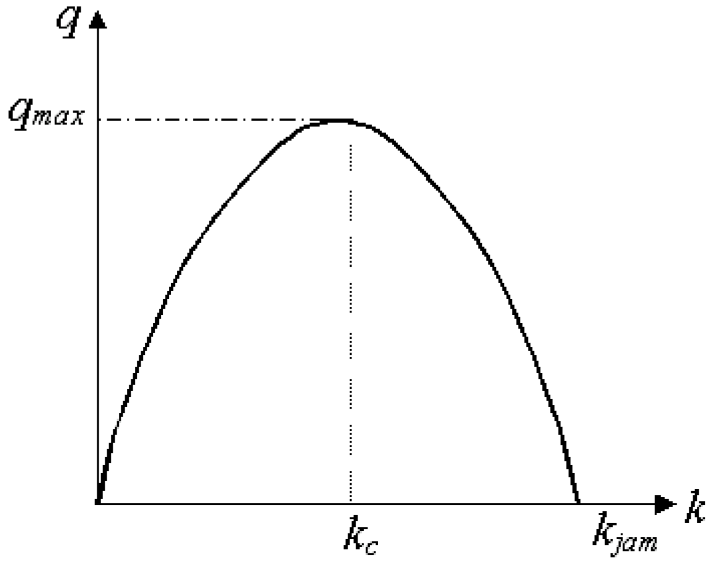


Figure 1.1 the Density-Flow relationship in Greenshield's Model

The traffic parameters above are defined as:

q_{max} is the maximum number of vehicles passing a certain point, which is given as vehicles per hour (vehs/h).

K_{jam} is the number of vehicles occupying a certain space when traffic jam formed on a freeway, which is given as vehicles per kilometer (vehs/km)

K_c is the traffic density when traffic flow is at the maximum value, which is given as vehicles per kilometer (vehs/km).

Figure 1.1 displays the relationship between traffic density (vehs/km) and traffic flow (vehs/h). When the traffic density reaches a certain point, the critical density (K_c), the traffic flow on a motorway reaches its maximum flow (q_{max}). Then the traffic flow would actually decrease with the increase of traffic density until flow density reaches the jam density where no car move any more and traffic flow turns to be zero. In other words, only when the flow density is below the critical density is traffic flow possible to reach its maximum value, which means the motorway is not congested.

On the contrary, once traffic density exceeds the critical value, traffic flow drops due to motorway capacity drop, which means motorway is congested. The fact that many publication notice traffic congestion degrades the available infrastructure capacity can be seen as the best evidence for that. Therefore, motorway congestion may form when traffic density exceeds the critical density.

For an isolated on-ramp, when downstream density exceeds the critical point, congestion may form and degrade the downstream capacity. Such a capacity drop would make ramp inflow much less than normal level. For a motorway network, congestion may happen in some network links where the traffic demand exceeds the motorway capacity; this limited congestion reduces the motorway capacity and sometimes leads to the increased congestion, which might lead to further capacity degradation and further expanded congestion. Finally the motorway network throughput would much less than the normal motorway capacity.

1.2 Ramp Metering

To alleviate or eliminate motorway congestion, ramp metering, or on-ramp control, which is used to control the traffic amount allowed to flow into each controlled on-ramp, has been consider as the most direct and effective way for motorway traffic control. The potential improvement achieved by ramp metering could be generalized as follows [2]:

- Reduce motorway congestion in space and time
- Increase motorway throughput.
- Reduced congestion spillback to the adjacent urban traffic network or to other merging motorways.
- Significant improvement of traffic safety due to reduced congestion duration.

To explain how ramp meter works, a typical fundamental diagram is displayed blow.

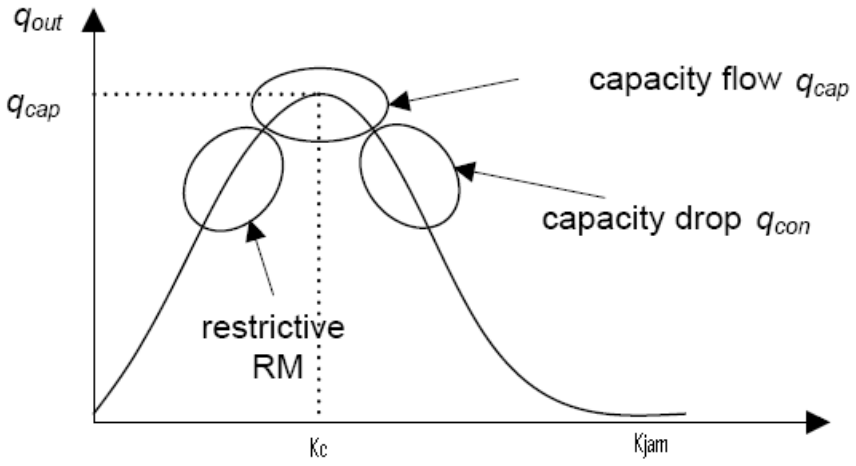


Figure 1.2 the Fundamental Diagram [2]

As we discussed before, motorway congestion may form when traffic density exceeds the critical density, so a successful ramp control strategy should be capable of maintaining traffic density blow the critical point and traffic flow around the maximum flow rate, motorway capacity or capacity flow (q_{cap}). If ramp metering is too restrictive, the mainstream throughput could not reach the motorway capacity and ramp metering actually causes extra delay for traffic flow. If ramp metering is too permissive, congestion may still happen and causes the capacity drop.

From the drivers' point of view, they want to shorten the duration of reaching their respective destinations at the motorway network exits. Ramp metering should be able to decrease the total travel time (TTT) when drivers driving on motorway s and waiting on ramps due to the avoidance of capacity drop caused by motorway congestion.

1.3 Objectives

There exist a large number of ramp metering strategies published in literature, such as demand-capacity, ALINEA and Zone algorithm. Fuzzy logic control based ramp metering algorithm is considered as one of the most suitable solution for the nonlinear traffic system by means of its characteristic of handling inaccurate information and inexact system model. Since fuzzy logic control (FLC) based ramp metering has been studied for years, some evolutionary algorithms such as neural networks and genetic algorithm have been applied with FLC based ramp metering to improve the performance of FLC based ramp meter in many publications. However, most of them focus on

optimizing the coordination of ramp meters for a traffic network; very few evolutionary algorithms have been applied to optimize an isolated on-ramp as local ramp metering algorithm.

The aims of this thesis research is to develop a new local traffic responsive ramp metering algorithm by applying genetic algorithm to a typical FLC ramp metering as an evolutionary algorithm and to study the difference of the performance between the genetic fuzzy ramp metering and the conventional FLC ramp metering.

1.4 The thesis contribution

The major contribution of this thesis is to present a genetic fuzzy ramp meter control approach as a local traffic responsive ramp metering algorithm. The objective function based on a local on-ramp model is developed to tune the fuzzy parameters to maintain the system flow density below the critical density. The research results show genetic fuzzy ramp metering is more effective to control the flow density to prevent the formation of congestion by means of the objective function. The research results also show the genetic fuzzy ramp metering has more significant improvement on the change of TTT than the conventional FLC ramp metering especially when the traffic demand is very high.

Chapter 2 Literature review

This chapter reviews the literature about some existing ramp control strategies and generalizes the developing trend of FLC based ramp approaches.

Basically, there are three types of ramp-metering schemes based on the level of complexity of the control approach: fixed time, local traffic responsive and coordinated traffic responsive ramp metering [4].

2.1 Fixed time ramp metering algorithm

Fixed-timed ramp control normally generates a ramp signal that operates at constant time cycle for a specific time of day, usually rush hour. For example, from 6:30 A.M. to 8: 00 A.M. during peak hour, a given ramp signal might be set to green cycle on for 5 seconds, then cycle off for 30 seconds. After 8:00 AM, the ramp meter would be shut down since the intense decrease of ramp demand would be impossible to cause any congestion for downstream. The ramp metering rates are preset based upon historical data that could be years, months, or days old [3].

The disadvantage of fixed time ramp metering is the lack of reaction to the changing traffic condition such as some irregular change of ramp inflow.

2.2 Local traffic responsive ramp metering algorithm

Local traffic-responsive ramp metering can automatically adjust the ramp metering rate based on current traffic conditions in the vicinity of the ramp. Local traffic condition will be detected by loop detectors. Controller electronics and software algorithms can select an appropriate metering rate based on the occupancy or flow data from the ramp and mainline detectors; therefore, traffic-responsive ramp-metering systems can generally deliver better results than fixed time metering [4] [5].

The physical components for local responsive ramp metering normally include:

- Ramp Metering Signal and Controller

  The signal is typically located to the drivers left, or on both sides of the ramp. Each ramp meter typically has one nearby weatherproof control cabinet which

houses the controller, modems, and inputs of each loop. The controller is set to a specific algorithm, which generates the ramp metering rate.

- Demand detector

  The check-in, or demand detector is located at the ramp cordon line. The check-in detector notifies the controller that a vehicle is waiting on ramp and to activate the green cycle.

- Merge Detector

  The merge detector is an optional component which senses the presence of vehicles in the primary merging area of the ramp. This prevents waiting vehicle from passing the ramp signal while the front vehicle still stopping in the merging area for some unexpected reasons.

- Queue detector

  The queue detector is located at the end of a ramp. The queue detector prevents vehicles from spilling over onto the surface street network.

- Mainline detector

  Mainline detectors are located on the motorway upstream and sometimes downstream. The detector collects the information of upstream or downstream traffic condition to feedback to ramp controller.

Figure2.1 shows a typical local traffic-responsive ramp meter system. The distribution of detector loops might be different according to deferent application.

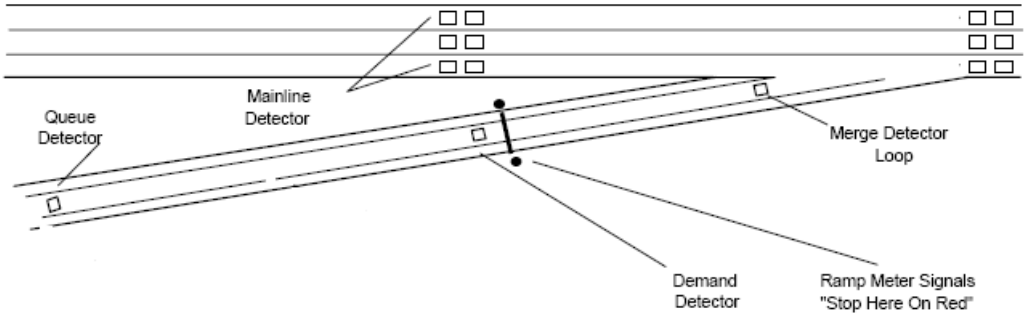


Figure 2.1 A typical local traffic-responsive ramp meter system

Also, to better understand how an adaptive ramp meter work with changing traffic condition, it is necessary to review several existing popular local traffic responsive ramp control algorithms.

2.2.1 Demands-Capacity Control Strategy

Demand-capacity control was introduced with the earliest field implementations of responsive ramp control. Under demand-capacity control, metering rates are based on the comparison between the upstream flow measured in the previous period, typically 1 minute earlier, and the downstream capacity [29].

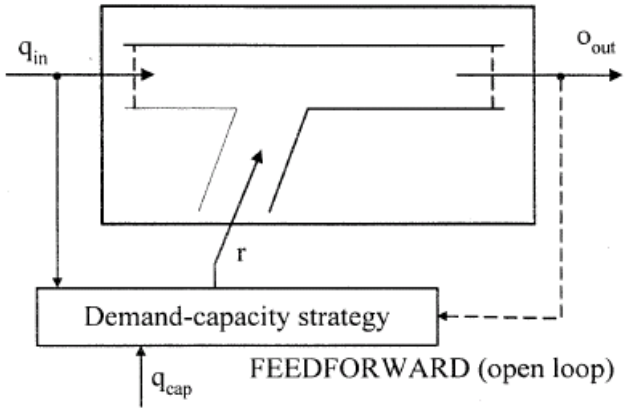


Figure 2.2 the Demands-Capacity Control Strategy [29]

The equation is shown blow:

$$R(t) = C - q_{in}(t-1) \tag{2.1}$$

Where,

R is the number of vehicles allowed entering motorway in period t.

C is the capacity of downstream section, the maximum numbers of vehicles entering downstream.

q_{in}(t-1) is the upstream flow rate in period t-1.

The upstream flow, $q_{in}(t-1)$, is measured by the loop detectors, and the downstream capacity, C, is a predetermined value.

The main disadvantage of this strategy is that the generated metering rate will be very unstable and sensitively oscillate with the change of the upstream flow rate since the feed-forward strategy is very sensitive to the system disturbances, such as a slow vehicle, a short shock wave or merging difficulties [2].

2.2.2 Percent-Occupancy Control Strategy

This strategy detects upstream occupancy to identify and measure the potential congestion. It is assumed that there is a decreasing linear function to describe the relationship between the commanded metering rates and mainline occupancy, as plotted in Figure 2.3.

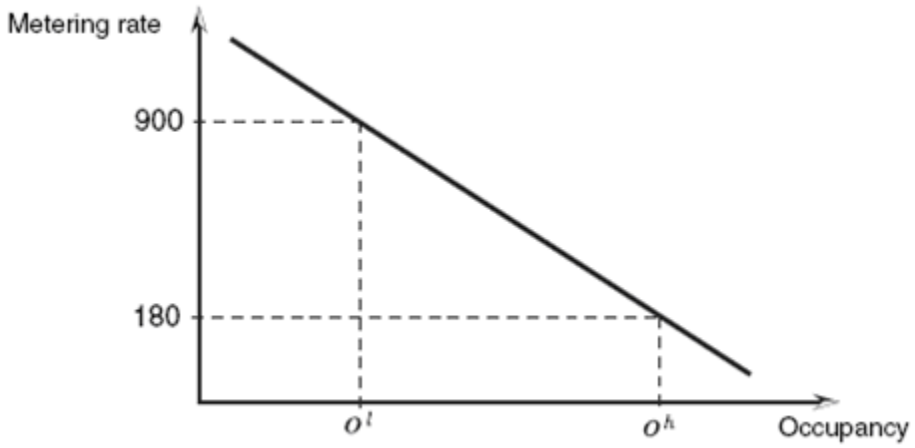


Figure 2.3 Percent-Occupancy Control Strategy

The equation is given as:

$$r(k) = 900 - \frac{900 - 180}{o^h - o^l}(o^{up} - o^l) \tag{2.2}$$

Where:

r(k) is the meter rate in time interval k.

$\mathbf{o^h}$ is the high occupancy threshold.

$\mathbf{o^l}$ is the low occupancy threshold.

$\mathbf{o^{up}}$ is the detected upstream occupancy

o^h and o^l are measured using historical data, by which the ramp metering rates is generated as a proportional rates. Percent-occupancy control is one of the most widespread on-ramp metering approaches in the U.S. due to its simplicity of implementation [6]. The main disadvantage of this strategy is the linearity assumption for the fundamental diagram which sometimes even causes more inaccurate metering rates than Demand-Capacity control [2].

2.2.3 AlINEA

Asservissement Linéaire d'Entrée Auotroutière (ALINEA), as a successful local responsive feedback ramp metering strategy, has been implemented in many cities, such as Paris, Amsterdam and Glasgow [4]. The algorithm adjusts the metering rate to keep the occupancy downstream blow a preset value, the critical occupancy. Different with both demand-capacity and percent-occupancy, ALINEA is a local- feedback control algorithm. The previous time metering rate will be used as the input for the next iteration. This smoothes the generated metering rates and avoids the wide swings between short time intervals [7]. Figure 2.4 shows a typical ALINEA algorithm.

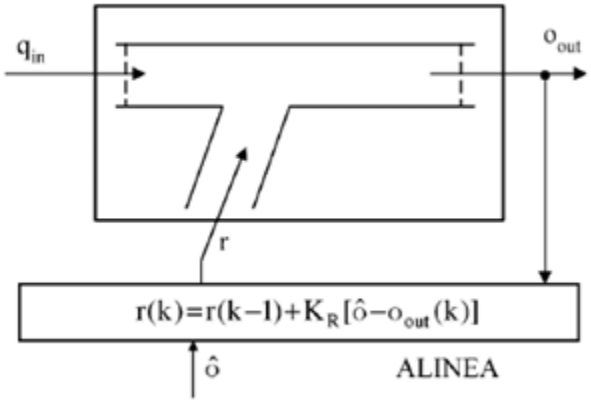


Figure 2.4 ALINEA

The equation is given as:

$$r(t) = r(t-1) + K[O_c - O_{out}(t)] \tag{2.3}$$

Where:

r(t) is the meter rate in time interval k.

K is a tunable parameter (weighting factor) greater than zero.

$\mathbf{O_{out}}$ is downstream occupancy

$\mathbf{O_c}$ is the preset occupancy valve (Paris: 29%; Amsterdam: 18%; Glasgow: 26%)

The main disadvantage of ALINEA is the control strategy does not consider ramp queue spill-back situation, which generally generates over-restrictive metering rates, so it is very hard to balance motorway congestion and ramp queue length. In addition, ALINEA must take occupancy measurements collected from the downstream at a specific location where potential merge congestion is possible to be detected. Such a position will not be easily found [7].

2.3 Coordinated traffic responsive ramp metering algorithm

In order to improve the efficiency of ramp control approaches, coordinated traffic responsive ramp metering algorithms are designed to optimize traffic flow over a section of the motorway rather than just a single ramp, which could be further divided into three classes: cooperative algorithms, competitive algorithms and integral ramp metering algorithms [8].

2.3.1 Competitive algorithms

Under competitive algorithms, two metering rates would be generated for a single ramp. One is based on local traffic condition and other one is based on traffic network condition. The restrictive one would be finally applied to the ramp.

The famous implemented competitive algorithms include Bottleneck (Seattle), Compass (Toronto) and SWARM (Lo Angeles) [4]. Figure 2.5 shows Bottlenect Algorithm flow charts.

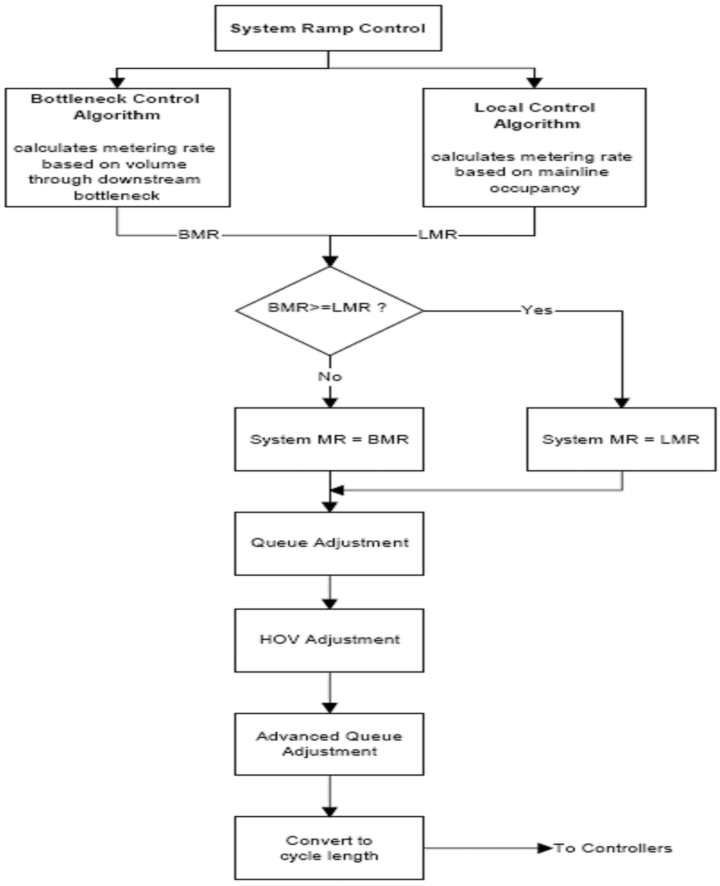


Figure 2.5 Bottleneck Flow Charts [4]

2.3.2 Cooperative algorithms

Under cooperative algorithms, ramp metering rates are first computed with the local traffic condition, and then adjusted with the traffic conditions of the whole network. The famous implemented algorithms include Helper (Denver) and Linked-ramp algorithm (San Diego) [4].

Take Denver algorithm as an example, the ramp controlled area has been divided into six zones, and one to seven ramp meters are assigned to each zone. The meter rate is first generated by local traffic condition of each ramp and then adjusted with the motorway traffic condition or zone traffic condition monitored by the central computer.

2.3.3 Integral ramp metering algorithms

Under integral ramp-metering algorithms, metering rates is generated by considering local traffic conditions and system-wide traffic conditions at the same time [8]. The famous implemented algorithms include Linear Programming or LP (Kobe, Japan) and METALINE (Paris) algorithm [4].

Take LP algorithm as an example, the algorithm considers a motorway network as the Linear Programming formula consisting of a number of tunable parameters and weighting factors for a series of ramps. To find the optimal ramp flow rate for each ramp, the objective function will be maximized under certain constraint equations.

The objective function is:

$$Z = (A1 \times U1) + (A2 \times U2) + ... + (Ai \times Ui) \tag{2.3}$$

Where:

Ai is the weighting factor for the ith ramp.

Ui is the ramp flow for the ith ramp.

The function is subject to the following constraints:

a) Ramp queue plus ramp demand minus ramp flow must be less than or equal to the maximum queue length.

b) Ramp demand plus ramp queue must be less than or equal to the ramp flow rate.

c) The metering rate must be between the maximum and minimum values.

Finally, the ramp meter rates will be solved simultaneously for all ramp locations by optimizing the Linear Programming function model.

2.4 The development of FLC based ramp control approaches

2.4.1 Advantages of FLC based ramp control approaches

The traditional ramp metering algorithms we mentioned before are all based on the assumption of the existence of some certain traffic mathematical models, such as ALENEA and Linear Programming equation. In other words, the more accurate the mathematical models would be, the better the performance of the ramp metering would be. Unfortunately, because of the complexity and nonlinear and non-stationary behaviour of the traffic system, obtaining an accurate control model is extremely difficult [10]. Under this background, fuzzy logic control has been involved in ramp metering control.

The reasons why FLC is better suited for ramp metering than traditional approach have been generalized to four main reasons by Taylor and Meldrum [9][16].

a) It can utilize incomplete or inaccurate data.
b) It can generate more smooth outputs rather than oscillatory metering rate.
c) It does not require extensive system modelling.
d) It is easy to tune by changing weighting factor and the parameters of membership functions.

2.4.2 The developing trend of FLC based ramp control approaches

As we have reviewed so far, there have been three control approaches in the development of ramp metering strategies: pre-timed, local traffic responsive and coordinated traffic responsive.

The disadvantage of pre-timed approach is its lack of response to current traffic conditions, either changes in demands or capacities. A local traffic responsive ramp metering plan is developed based on current traffic information obtained from upstream or downstream and ramp detectors, so it apparently overcomes the drawbacks the pre-timed one has, but the disadvantage of this approach is its lack of coordination between ramps in order to optimize the motorway network. Therefore, based on local traffic information, a system-wide approach is to be developed to utilize all local information to work toward systemic optimization of the motorway network, which is what we call coordinated traffic responsive.

Following the trend of development of ramp metering algorithm, FLC (fuzzy logic control) ramp metering has also been developed from a local design to a coordinated approach.

FLC has been used for local ramp metering strategies in two previous applications, Seattle, Washington/ Zoetermeer, Netherlands [11] [12].

In America, FLC ramp metering has been under development at the University of Washington for a number of years. This algorithm was installed in early 1999 by WsDOT, controlling 15 metered ramps along I-405. Early evaluation results have shown a promising improvement when comparing more traditional Seattle Bottleneck algorithm [13].

In Netherlands, FLC metering was first installed in 1989 and nine ramp meters were in place by 1995. The evaluation focused on the A12 motorway between Utrecht and Hague, and for the 11 km study area, it showed 3% increase in bottleneck capacity. Other positive results included higher speeds during congested hours, and 13% shorter travel times. Although ramp travel time had about 20 seconds increase, total system effects were acceptable. Along the A12 near Zoetermeer, a comparative study between three different local metering algorithms RWS strategy, ALINEA and Fuzzy-Logic had been implemented and the results showed FLC appears more beneficial than other two [12].

Based on local FLC ramp metering algorithm, adaptive fuzzy control concept had been used in a proposed structure of a coordinated FLC ramp metering by K. Bogenberger in 1999 [4]. There was two kinds of coordinated FLC ramp metering algorithms that had been proposed, Genetic algorithm based and neuro-fuzzy based [14][15][17].

In 2001, these two kinds of algorithms had been generalized into a new model family, ACCEZZ (Adaptive and Coordinated Control of Entrance Ramps with Fuzzy Logic), and expanded to five versions: neuro-fuzzy online, neuro-fuzzy offline, GA fuzzy online, GA fuzzy offline and GA fuzzy reality. The performance of the new model was assessed in a simulation context with a microscopic traffic flow model and compared with the results of five different standard ramp metering algorithms: demand-capacity, occupancy strategy, ALINEA, Denver's HELPER algorithm and Minnesota's Zone approach [10]. The total time spent (TTS) in the system was used to evaluate the overall system

performance of a strategy, and the final results showed ACCEZZ model better than others. Table2.1 shows the table of compared results.

Table 2.1 Total time spent (TTS) in system [10]

	3-On-Ramp Scenario		*5-On-Ramp Scenario*		*Incident Scenario*	
	7.00 – 20.00	11.00 – 18.00	7.00 – 20.00	11.00 – 18.00	7.00 – 20.00	11.00 – 18.00
No Control	100%	100%	100%	100%	100%	100%
HELPER	87%	82%	88%	78%	85%	79%
Demand-Capacity	87%	82%	89%	79%	89%	83%
Occupancy	86%	80%	87%	77%	87%	81%
ALINEA	85%	79%	86%	77%	86%	80%
Zone	85%	78%	87%	77%	86%	79%
Neuro-Fuzzy Online	85%	78%	86%	76%	86%	80%
Genetic Fuzzy Online	85%	78%	86%	76%	86%	80%
Neuro-Fuzzy Offline	84%	77%	86%	76%	86%	80%
Genetic Fuzzy Offline	83%	76%	86%	76%	86%	79%
Genetic Fuzzy Reality	83%	76%	-	-	-	-

Figure 2.6 and Figure 2.7 show the structures of Neuro-Fuzzy and Genetic Fuzzy Model.

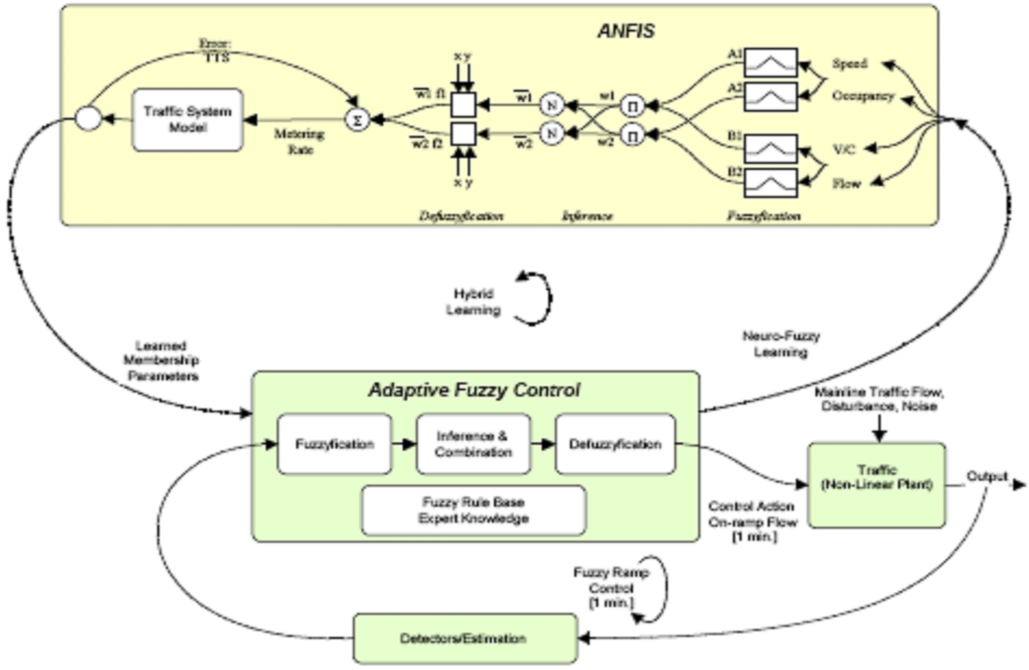


Figure 2.6 the Neuro Fuzzy Model [10]

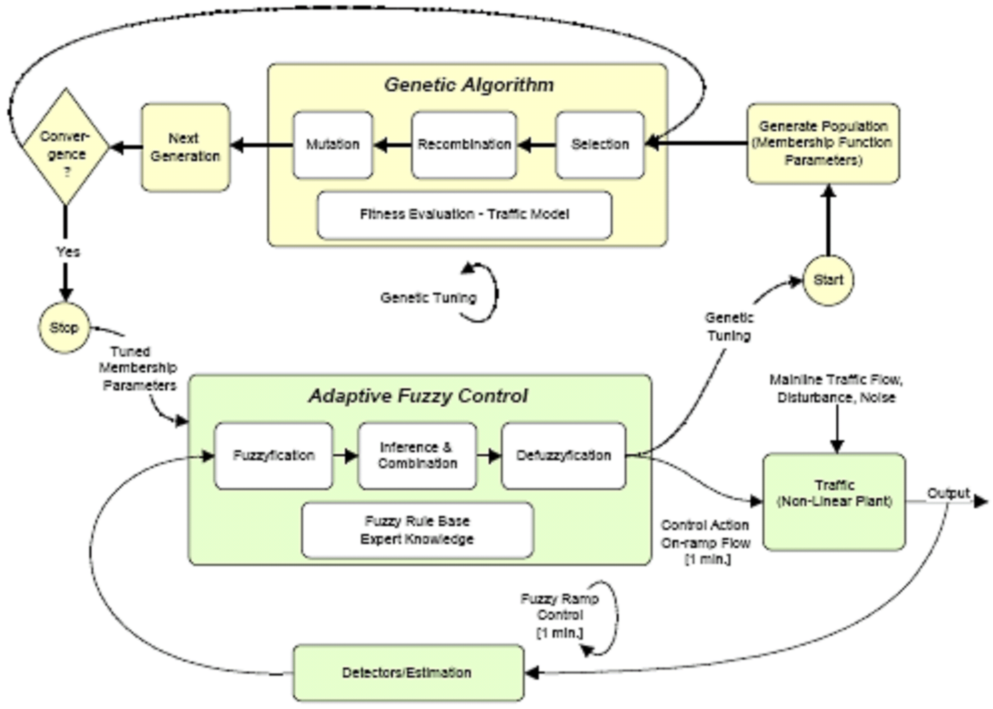


Figure 2.7 the Genetic Fuzzy Model [10]

2.5 Conclusion

From the research on the existing ramp metering approaches and the development of FLC based ramp metering, the information to design a FLC based local traffic responsive ramp meter could be generalized as flows:

a) Local traffic responsive ramp metering is actually a specific algorithm that reacts to local traffic condition periodically, so the installation of loop detectors is essential to collect real time traffic information.

b) The positions to place detectors are not fixed, which normally should include the upstream and downstream of on-ramp, the entrance of on-ramp and the position of the ramp meter.

c) Different with the traditional ramp control approaches, a typical FLC ramp metering algorithm does not need an exact traffic model or mathematical formula, but the rules of FLC must be based on the knowledge of traffic system.

d) Evolutionary algorithms such as Neuro-Fuzzy and Genetic Fuzzy could be used to improve the performance of FLC ramp metering, but it will not be unnecessary

for FLC ramp metering to select a proper traffic model since the objective function of evolutionary algorithms needs a exact mathematical expression.

e) For an isolated metered on-ramp, generalized ramp metering rates should be able to maintain the downstream traffic volume (occupancy or flow density) close to but no more than the critical value.

f) To evaluate the performance of FLC ramp metering, a traffic simulation environment will be involved to implement and analyse the ramp control algorithm.

Chapter 3 Fuzzy ramp metering algorithm

This chapter will present the outline of a simple fuzzy logic ramp meter proposed by Bogenberger [4]. The control logic has been coded in both Matlab and Microsoft C++ for test.

3.1 Overview of fuzzy logic control

Fuzzy logic is more like human language than traditional logical systems. Rather than forcing a yes or no, on or off response, fuzzy logic utilizes the linguistic variables like small, very small, big, very big and normal. To process the approximate and inexact information for the real world, the fuzzy logic controller is actually a set of linguistic control rules related by the dual concepts of fuzzy implication and the compositional rule of inference [18] [19].

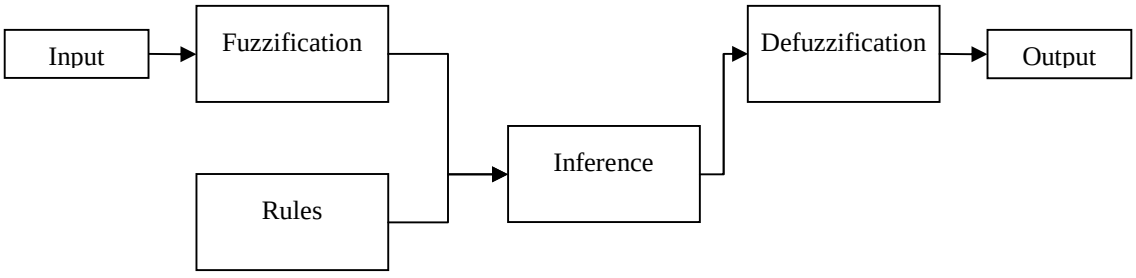


Figure 3.1 a typical fuzzy rule-based system

A typical fuzzy rule-based system is presented as Figure 3-1. Following this procedure, a simple FLC could be defined by four steps as follows:

- Fuzzification

  The fuzzification translates each input into a set of fuzzy variables via membership functions, which could be discrete or continuous and defined as triangles or bell-shaped curves.

- Rules

  Rules, or the knowledge base, are the essential part of a fuzzy logic controller, which are based on expert opinions, operator experience and system knowledge.

- Inference

By applying fuzzy operators and implication methods to rule base, the fuzzy inputs would be converted into one fuzzy output.

- Defuzzification

Defuzzification produces a crisp output based one the fuzzy output.

3.2 Fuzzy logic control based ramp metering algorithm

Figure 3.2 shows the layout of the local traffic responsive FLC ramp metering.

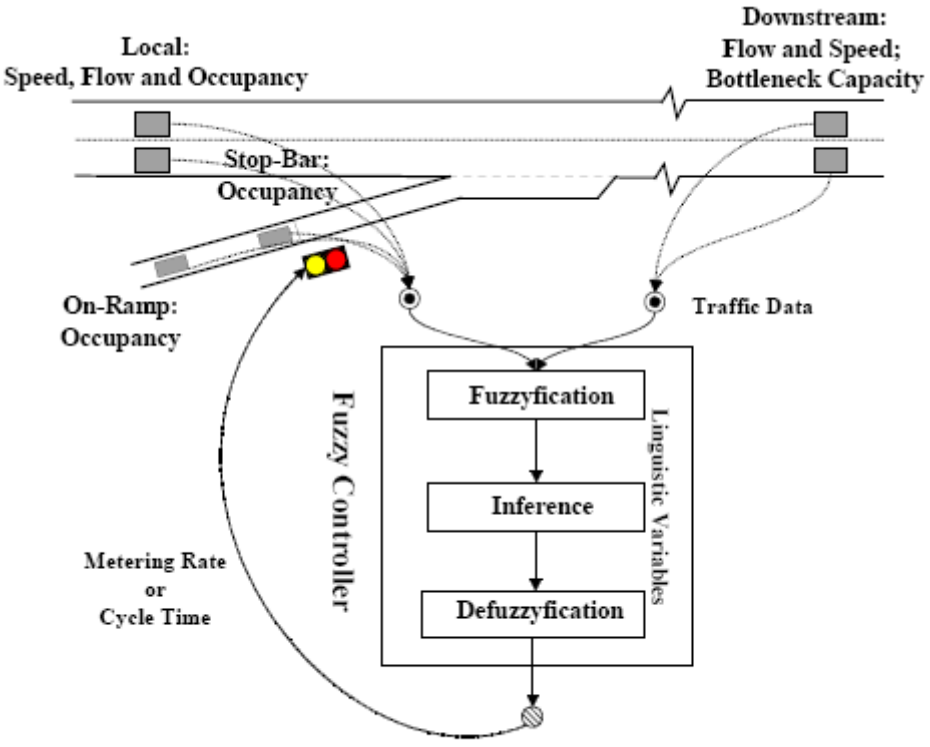


Figure 3.2 the layout of FLC ramp metering [10]

Local traffic condition could be monitored by four detectors installed at different locations, which include upstream detector, downstream detector, queue detector and check-in detector.

The upstream detector is used to collect traffic information including local speed, local traffic flow and local occupancy of the upstream of the ramp. The downstream detector is to detect the downstream speed and flow rate (volume). According this value, the downstream volume/capacity-ratio (v/c-ratio) is calculated with being divided by the capacity of the major downstream bottleneck, which is the historical measured maximum

flow rate of mainline. The reason why v/c radio is involved in the control logic is because it is a quite popular measurement for bottleneck behaviour. The detector at the end of the ramp storage is to detect the queue occupancy, which is also called queue detector. The check-in detector is the detector located at the ramp metering stop bar and used to detect the occupancy of the vehicle waiting by the stop bar.

Each detection interval is one minute in order to smooth the input signal while still keeping a quick response with the change of traffic condition. Since occupancy and speed are main indicators for the change of traffic condition, they would be detected by most sensors. Here the occupancy is defined as the time percentage that vehicles occupy at a detector location during a detection circle.

Finally, the detected information, which reflects the real time local traffic condition, will be processed to generate a ramp metering rate by the fuzzy logic controller, following three steps: fuzzification, inference and defuzzification.

3.2.1 Fuzzification

As generally fuzzification does, each crisp input and output has to be translated into a set of fuzzy class, so each detected input signal would translate into a fuzzy set with a understandable term like “low”, “medium” or “high”. For example, the local occupancy, local flow and local speed are described by the terms "low", "medium" and "high" and the degree of activation indicates how true that class is on a scale of 0 to 1.

Totally six inputs and one output need to be fuzzified. Input fuzzy sets are defined as sigmoid function or Gauss function, and output fuzzy sets are defined as triangular function that is easy to be defuzzified.

Table 3.1 lists the fuzzy sets of inputs and of output.

Table 3.1 Input and output fuzzy sets

	Fuzzy Sets			
	Terms of the Fuzzy Sets			Membership function
Local Speed	Low	Medium	High	Gauss
Local Flow	Low	Medium	High	Gauss
Local Occupancy	Low	Medium	High	Gauss
Downstream V/C	Very High			Sigmoid
Downstream Speed	Very Low			Sigmoid
Check-In Occupancy	Very High			Sigmoid
Queue Occupancy	Very High			Sigmoid
Metering Rate	Low	Medium	High	Triangular

Local speed is from 0 to 100 km/h and described as three Gaussian fuzzy set, low, medium and high, with an overlap of 50%. The parameters (centre point and the sigma value) are found by Matlab plot function as depicted in Figure 3.3.

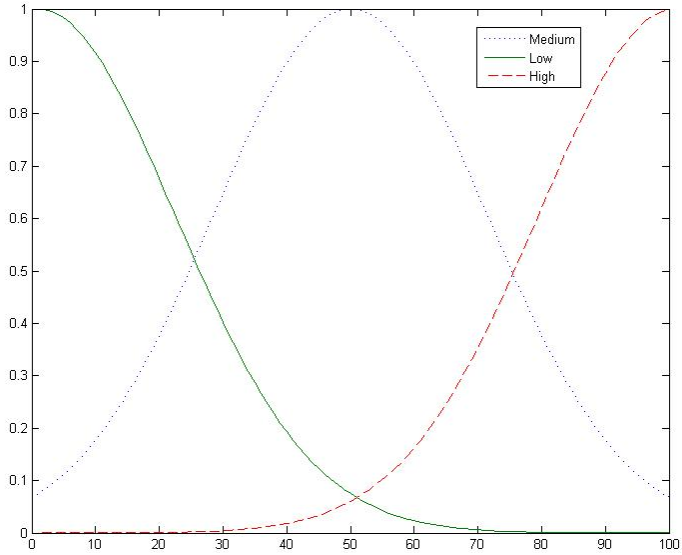


Figure 3.3 Fuzzy sets for the upstream speed

Local occupancy is from 0 to 30% and described as three Gaussian fuzzy set, low, medium and high, with an overlap of 50%. The parameters (centre point and the sigma value) are found by Matlab plot function as shown in Figure 3.4

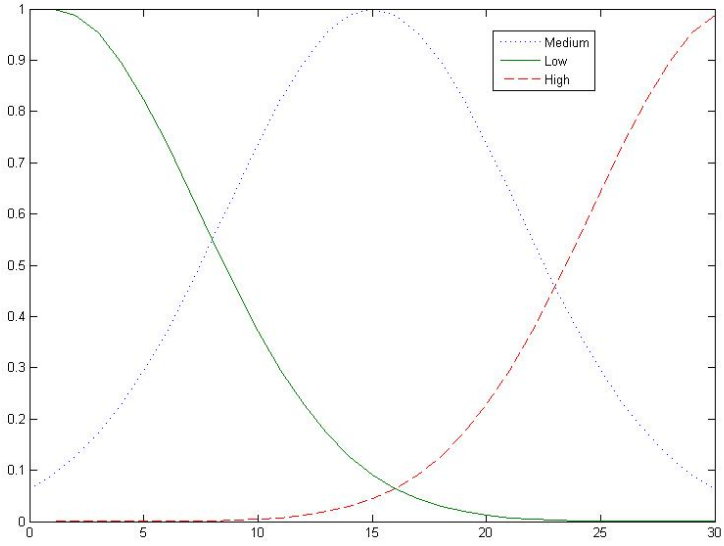


Figure 3.4 Fuzzy sets for the upstream occupancy

Local flow rate is from 0 to 4000vehs/h and described as three Gaussian fuzzy set, low, medium and high, with an overlap of 50%. The parameters (centre point and the sigma value) are found by Matlab plot function as shown in Figure 3.5.

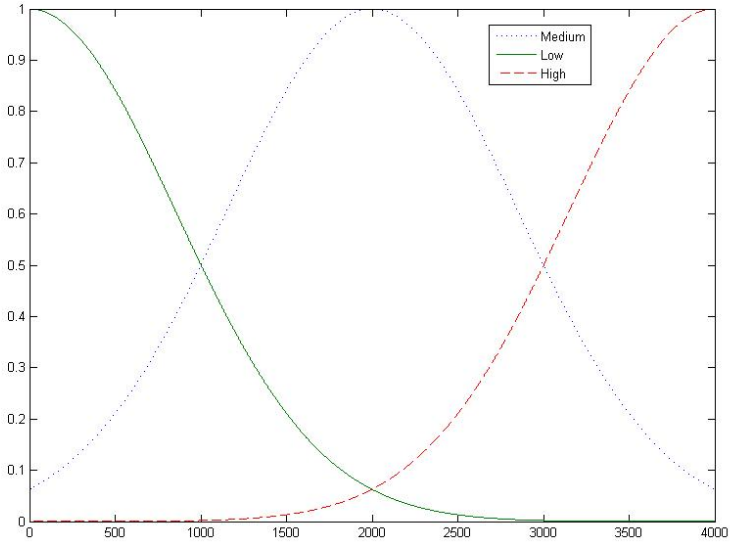


Figure 3.5 Fuzzy sets for the upstream flow rate

The downstream volume-capacity ratio is from 0 to 1 and fully activated at 0.9. A sigmoid function is used as the membership function. The parameters (centre point and the sigma value) are found by Matlab plot function.

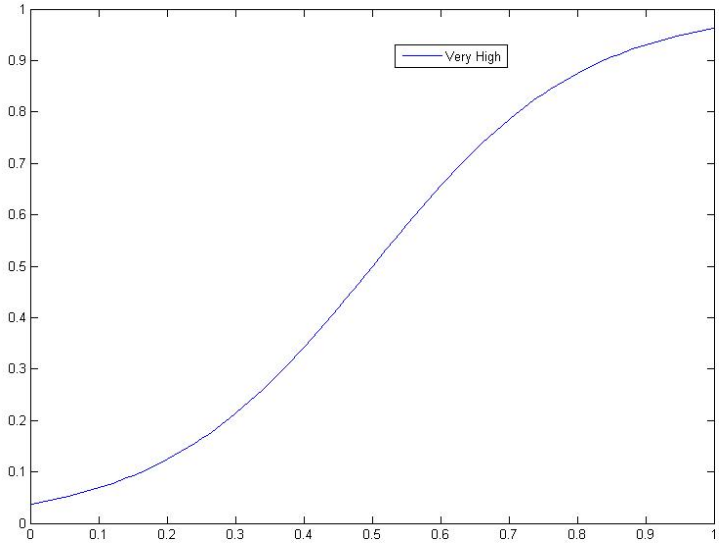


Figure 3.5 the fuzzy set for the downstream volume-capacity ratio

Downstream speed is from 0 to 100 km/h and activated at 50km/h and ended at 80km/h. A sigmoid function is used as the membership function. The parameters (centre point and the sigma value) are found by Matlab plot.

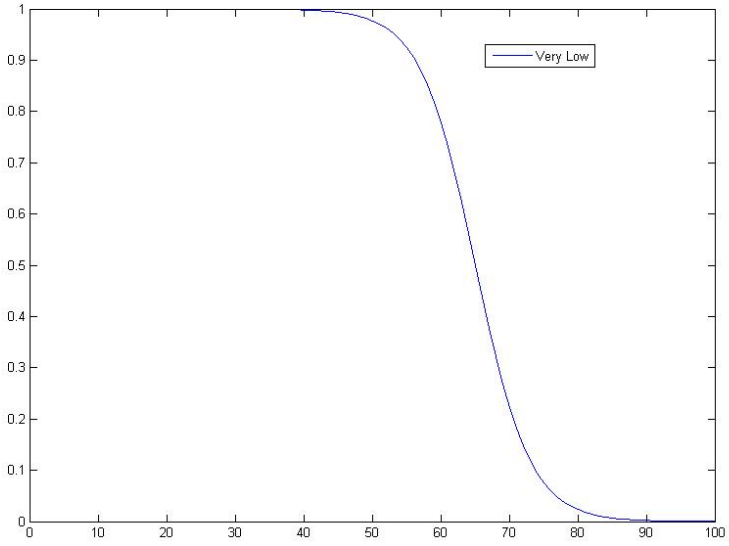


Figure 3.6 the fuzzy set for the downstream speed

Check-in occupancy is from 0 to 50% and activated from 10% to 30%. A sigmoid function is used as the membership function. The parameters (centre point and the sigma value) are found by Matlab plot function as shown in Figure 3.7.

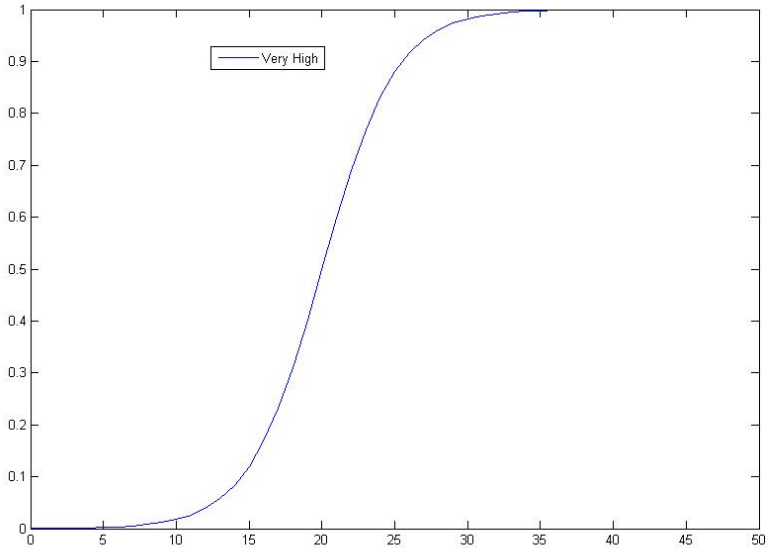


Figure 3.7 the fuzzy set for the check-in occupancy

Queue occupancy is from 0 to 50% and activated from 10% to 30%. A sigmoid function is used as the membership function. The parameters (centre point and the sigma value) are found by Matlab plot function as shown in Figure 3-8.

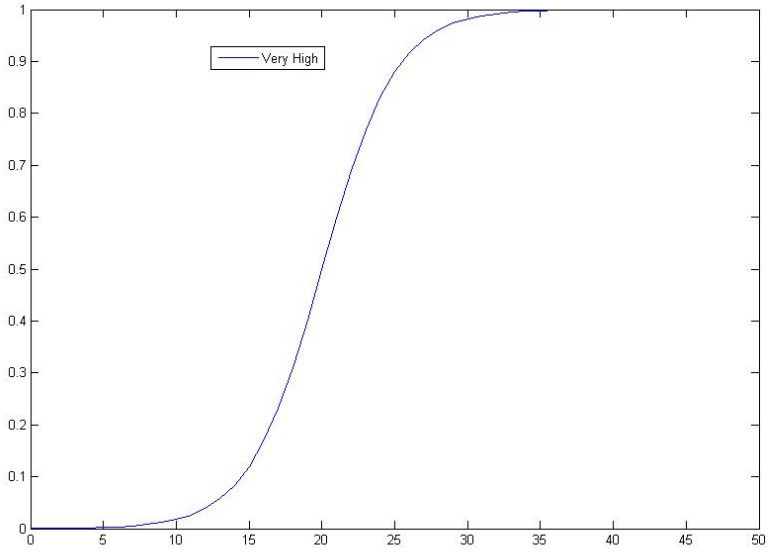


Figure 3.8 the fuzzy set for the queue occupancy

Metering rate as the output of fuzzy algorithm has also been converted to fuzzy sets, which is from 240vehs/h to 900vehs/h and described as three triangular fuzzy set, low, medium and high, with an overlap of 50%.

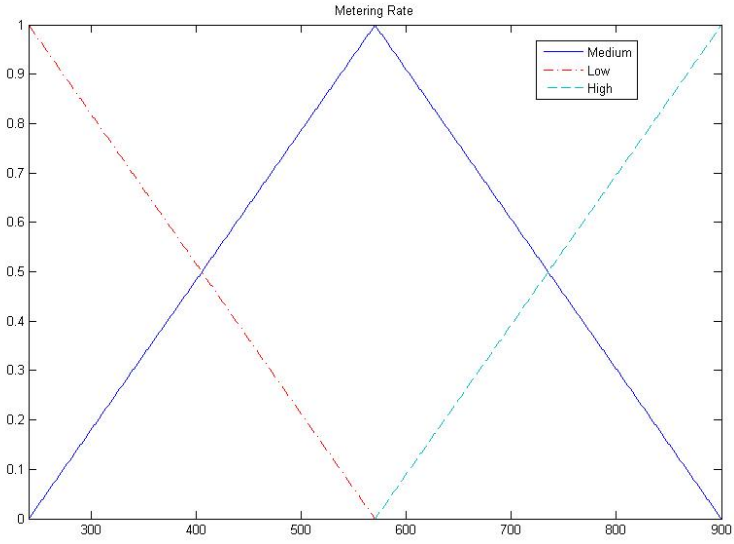


Figure 3.9 Fuzzy sets for the metering rates

To make the programming easy, the following scaling equation normalizes the output variables from the (240, 900) range to the (0, 1) range.

$$\text{Scaled metering rate} = (\text{metering rate} - LL)/(HL - LL)$$

HL (high limit) and LL (low limit) are two scaling parameters. HL is equal to 900 vehs/h and LL is equal to 240 vehs/h. Therefore, the output fuzzy sets are actually shown as Figure 3.10.

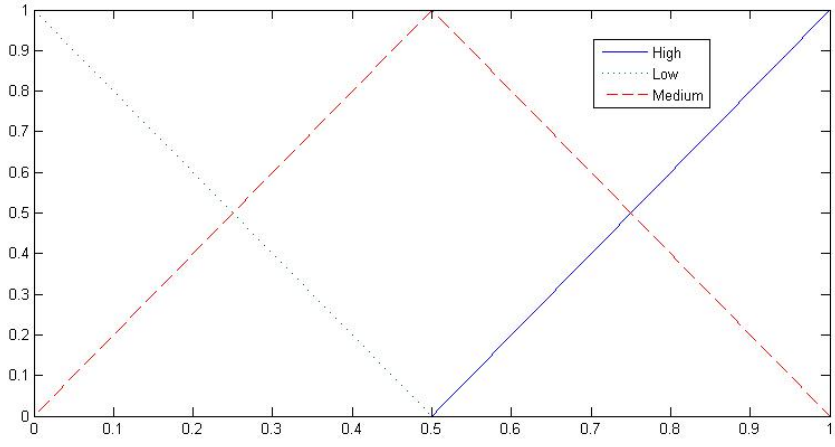


Figure 3.10 Fuzzy sets for the scaled metering rates

3.2.2 Inference

Fuzzy relation between inputs condition and outputs responses will be defined as a list of if-then pairs, where input condition is a “premise” and output response is a “consequent”. For example,

IF <premise> THEN<consequent>

IF <premise 1> AND/OR <premise 2> AND/OR <premise 3>..... THEN <consequent>

Where, an AND operation is analogous to the intersection of fuzzy sets, which takes the minimum value of given membership degrees. A OR operation is analogous to the union of fuzzy sets, which takes the maximum value of given membership degrees.

Rule base for a simple fuzzy ramp metering controller is given in Table 3.2.

Table 3.2 Rule base for fuzzy ramp metering

Rule	Rule Weight	Rule Condition	Rule Outcome
1	1.5	If local occupancy is Low	Then metering rate is High
2	1.5	If local occupancy is Medium	Then metering rate is Medium
3	2.0	If local occupancy is High	Then metering rate is Low
4	2.0	If local speed is Low AND local flow is High	Then metering rate is Low
5	1.0	If local speed is Medium AND local occupancy is High	Then metering rate is Medium
6	1.0	If local speed is Medium AND local occupancy is Low	Then metering rate is High
7	1.0	If local speed is High AND local flow is Low	Then metering rate is High
8	3.0	If downstream speed is Very Low AND downstream v/c is Very High	Then metering rate is Low
9	3.0	If Check-in occupancy is Very High AND Queue occupancy is Very High	Then metering rate is High

Rule1 to Rule 3

The purpose of rule 1 through 3 is to form a complete rule base, which means at least one of the rules would fire since the whole occupancy range is covered.

Rule 4 to Rule 7

The rules couple speed with either occupancy or flow to generate metering rates according to the fundamental diagram of traffic flow shown blown:

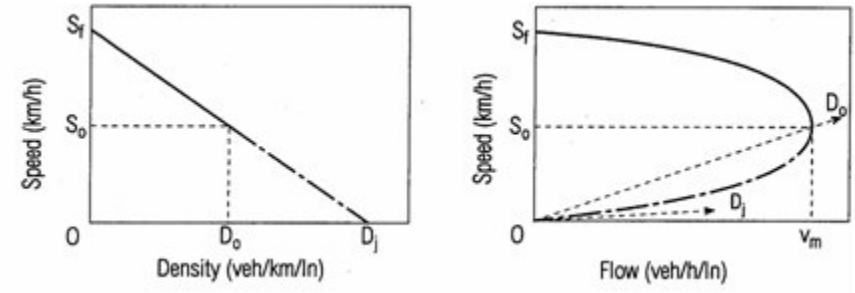


Figure 3-11 Fundamental diagrams

Rule 8

The purpose of this rule is to prevent the formation of downstream congestion rather than just simply react to it. Volume/capacity-ratio (v/c-ratio) calculated with the historical measured maximum flow rate of downstream can be seen as a prediction of the downstream bottleneck behavior.

Rule 9

Rule 9 is to prevent the excessive queue formation and to avoid a spillback onto the arterial street road on the basis of information collected from queue detector.

Rule weights

The rule weight is to stress the priority of each rule. Rule weighting scheme is flexible for different applications.

Aggregation of fuzzy rules

With different input patterns, at least one rule would generate at least one outcome at a time. When more than one rules firing at same time, a method is going to be needed to calculate an overall output degree. The additive method, which is to add different output degrees together, is used in this case. The advantage of additive method is less sensitive to faulty loop detector data [1].

3.2.3 Defuzzification

The defuzzification process is to convert a fuzzy output variable into a crisp value (metering rate). The centroid method is commonly used for the defuzzification process. The equation of central gravity method shown blow:

$$\frac{\int xf(x)dx}{\int f(x)dx} \tag{3.1}$$

In practice, a discrete fuzzy centroid equation is used to replace the continuous centroid equation since it is easier to calculate.

$$\frac{\sum_{i=1}^{N} w_i c_i I_i}{\sum_{i=1}^{N} w_i c_i} \tag{3.2}$$

Where:

N is the numbers of the output classes.

$\mathbf{c_i}$ is the centroid of the ith output class.

$\mathbf{w_i}$ is the results of the aggregation of rules at the ith output class.

$\mathbf{I_i}$ is the area of the ith output class.

3.3 Conclusion

In this chapter, a typical fuzzy logic ramp meter is presented and coded in Microsoft C++. Rules base is designed by common traffic knowledge and membership functions are defined as Gaussian function or sigmoid function. The parameters of membership function are manually found by means of Matlab. Theoretically, this fuzzy ramp control approach is already able to response to the change of traffic condition, which will be proved by later simulation. However, will such a response maintain the downstream flow density blow critical point or reasonable range? If not, the parameters of membership functions could apparently be further tuned for the better performance of the ramp metering. From this perspective, it seems inevasible to involve a tuning algorithm in this ramp control approach.

Chapter 4 Genetic Fuzzy Ramp Metering Algorithm

This chapter presents a genetic tuning process for the optimization of fuzzy control ramp metering. Genetic algorithm is applied to optimize the fuzzy ramp metering algorithm that was presented in Chapter 3. The objective of the genetic algorithm is to maximize the ramp inflow under the restriction of the critical density by tuning the fuzzy parameters. The evolutionary algorithm has been coded in Microsoft c++ and the part of essential c++ codes will be given with explanation.

4.1 The framework of genetic fuzzy ramp metering algorithm

Generally, genetic fuzzy system could be concluded into two major approaches: genetic tuning processes and genetic learning processes [20] [21]. Genetic tuning processes are targeted at optimizing the performance of a predefined fuzzy system by adjusting the parameters of membership functions. Genetic learning processes are concerned with automatically generating a set of fuzzy if-then rules that establishes the appropriate relationship between input and output states.

Figure 4.1 shows the layout of a general genetic fuzzy system

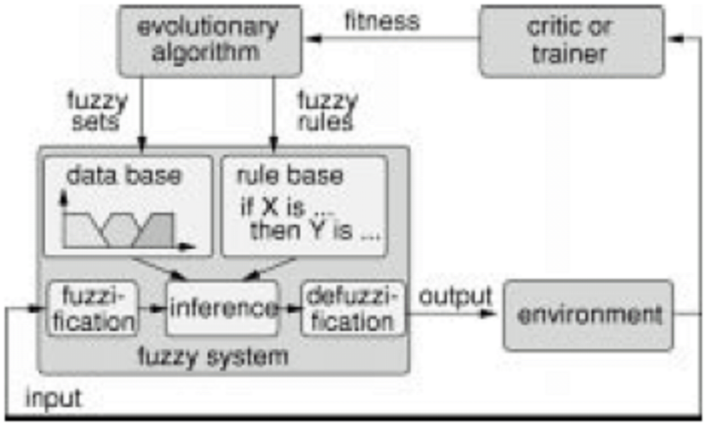


Figure 4.1 the Genetic fuzzy system [20]

Genetic fuzzy ramp metering algorithm is actually a genetic tuning process, so the core of optimization is to tune the parameters of fuzzy sets and the rule base will not change during the tuning process. Figure 4.2 shows the framework of a genetic fuzzy ramp metering approach:

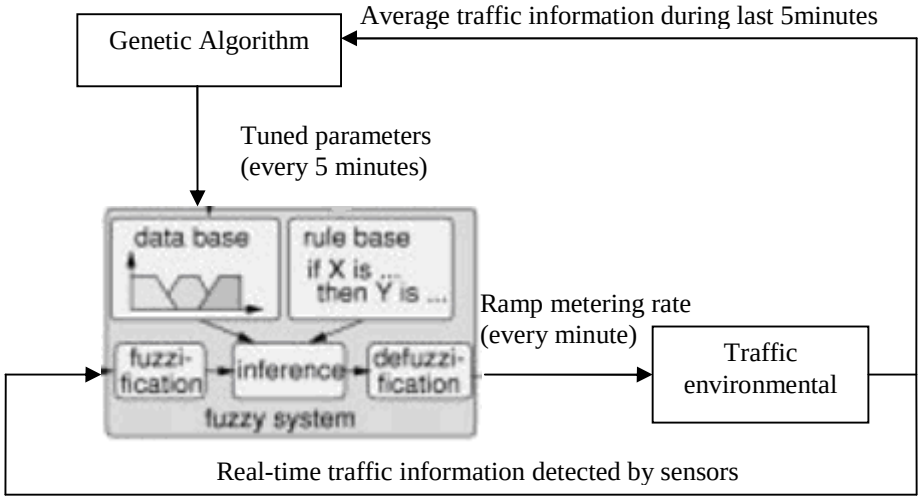


Figure 4.2 the framework of genetic fuzzy ramp metering algorithm

Basically, ramp metering rate will be generated by fuzzy logic algorithm every minute according to local traffic condition. The parameters of fuzzy sets will be updated every 5 minutes based on the average values of the last 5 minutes that is collected by all detectors. Genetic algorithm is used as the evolutionary algorithm to tune the parameterized membership functions.

4.2 Genetic algorithm

4.2.1 Overview of genetic algorithm

Genetic algorithms (GAs) are self-adapting strategies for searching, based on the random exploration of the solution space coupled with a memory component which enables the algorithms to learn the optimal search path from experience [22].

As a probabilistic search algorithm, GAs follows the principle of Charles Darwin of survival of the fittest. First, the standard GA evolves a multiset of elements called a population of individuals as a group of possible solutions for a given optimization problem. Each individual Ai (i =1…n) of the population A represents a possible solution of the optimization problem to be solved. And each individual is composed of genes which may take on a number of values restricted to {0, 1}. In other word, these individuals are represented as binary strings with fixed length, such as, “100010001”. Then, the fitness of these individual will be computed by the fitness function, the

objective function of the optimization problem. The individuals with higher fitness could have more opportunity to transfer their genes to the next generation by means of selection and crossover. After a numbers of generations, the individual with the highest fitness in the last generation will be finally considered as the final solution of the optimization problem. The standard genetic algorithm could be generalized to the following sequence:

Step 1: Randomly generate an initial population $A(0) := (A_1(0)...A_n(0))$.

Step 2: Compute the fitness f (Ai (t)) of each individual Ai (t) of the current population A(t).

Step 3: After the fitness of each individual has been calculated, a procedure known as selection is performed to keep the individuals with higher fitness.

Step 4: Once selection has occurred, crossover takes place between pairs of selected individuals. The strings of two individuals are mixed.

Step 5: The next operation that occurs is mutation, the random changing of bits in the individual. It is generally performed with a relatively low probability.

Step 6: Generate A (t+1)

Step 7: Repeat step 2 until satisfying solution is obtained.

Where step 3, step 4 and step 5 are commonly used in GA, they are generalized into three operations: selection, crossover and mutation. The procedures of GA could also be shown as Figure 4.3.

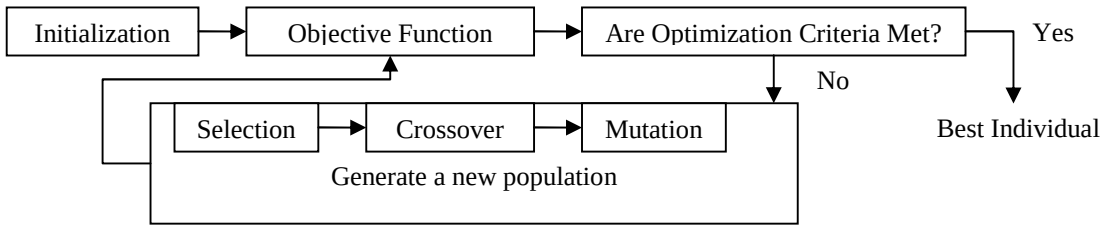


Figure 4.3 the layout of standard genetic algorithm

4.2.2 The application of genetic algorithm

Genetic algorithm has been applied to optimize the parameters of the membership functions of the fuzzy ramp metering based on the critical traffic density of the on-ramp model. The genetic tuning process will be explained in this section, following the procedures shown in Figure 4.3.

4.2.2.1 Initialization

To generate an initial population, all parameters to be optimized need to be represented as binary strings. Since there are seven fuzzy inputs that include 13 fuzzy sets in the fuzzy ramp metering controller and they are defined as Gauss or sigmoid functions, totally 26 parameters (each function includes center point and sigma value) could be optimized. To shorten the time of computation, only center points will be tuned, so only 13 parameters will be tuned finally, which also means the shape of membership functions will keep same during the genetic tuning process. Table 4.1 shows all parameters to be tuned and the corresponding tuning ranges.

Table 4.1 the fuzzy parameters to be tuned

Parameter	Fuzzy Sets		
	Initial value	Tuning range	Membership function
Local Speed Low	0	0~100	Gauss
Local Speed Medium	50	0~100	Gauss
Local Speed High	100	0~100	Gauss
Local Flow Low	0	0~4000	Gauss
Local Flow Medium	2000	0~4000	Gauss
Local Flow High	4000	0~4000	Gauss
Local Occupancy Low	0	0~-30	Gauss
Local Occupancy Medium	15	0~30	Gauss
Local Occupancy High	30	0~30	Gauss
Downstream V/C	0.5	0~1	Sigmoid
Downstream Speed	65	0~100	Sigmoid
Check-In Occupancy	20	0~50	Sigmoid
Queue Occupancy	20	0~50	Sigmoid

Figure 4.4 shows the layout of generating an initial population.

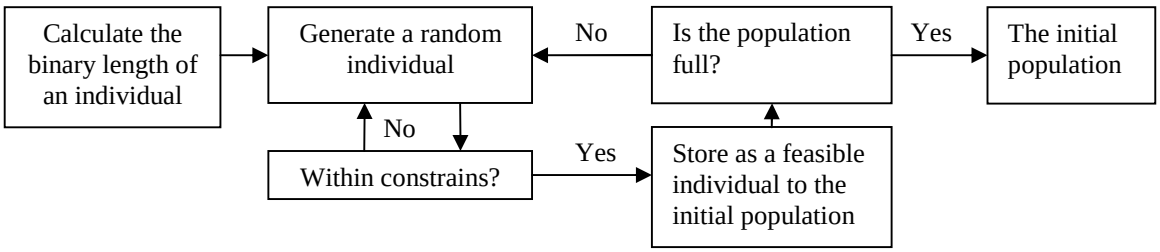


Figure4.4 the layout of generating an initial population

a) Calculate the binary length for each individual

Suppose that N is the length of the binary string of a specific parameter within a range (a, b). The integer number, N, could be found such that:

$$2^{N-1} < (b-a) \times 10^{precision} \leq 2^{N} \tag{4.1}$$

The relative C++ codes could be shown blow:

……..

```
int cLength(int precision, double rangeStart, double rangeEnd)
{
int length=0;
double total=(rangeEnd-rangeStart)*pow(10.0,precision);
while(total>pow(2.0,length))
{length++;}
return(length);
}
```

Therefore, the total length of an individual will be the sum of the binary lengths of all parameters, which could be given by c++ code shown blow:

…….

```
int speed_Length=cLength(precision, domain[0], domain[1]);
int flow_Length=cLength(precision, domain[2], domain[3]);
int occupancy_Length=cLength(precision, domain[4], domain[5]);
int vc_Length=cLength(precision, domain[6], domain[7]);
int downstream_speedLength=cLength(precision, domain[8],
domain[9]);
int checkin_Length=cLength(precision, domain[10], domain[11]);
int qocc_Length=cLength(precision, domain[12], domain[13]);

int Total_length= 3*speedLength +3*flowLength+ 3*occLength +
vcLength + dspeedLength + checkinLength + qoccLength;
/* where domain [] is the array defined as the ranges of all
parameters */
```

b) Generating a feasible individual within constrains

As we mentioned before, the general genetic tuning process will keep the rule base same, which means the center points of fuzzy sets representing the linguistic terms such as “low”, “medium” and “high” will be monotonically increasing within the relative range during the tuning process. Take the fuzzy sets of local speed as an example, the center points of fuzzy sets should maintain the relationship like:

$$0 \leq \text{Local Speed Low} < \text{Local Speed Medium} < \text{Local Speed High} \leq 100 \qquad (4.2)$$

If each center point is coded as the positive distance from the previous center like Figure 4.5, we could maintain the relationship above by means of an inequality constrain like:

$$0 \leq A+B+C \leq 100 \text{ or } 100 - A-B-C \geq 0 \qquad (4.3)$$

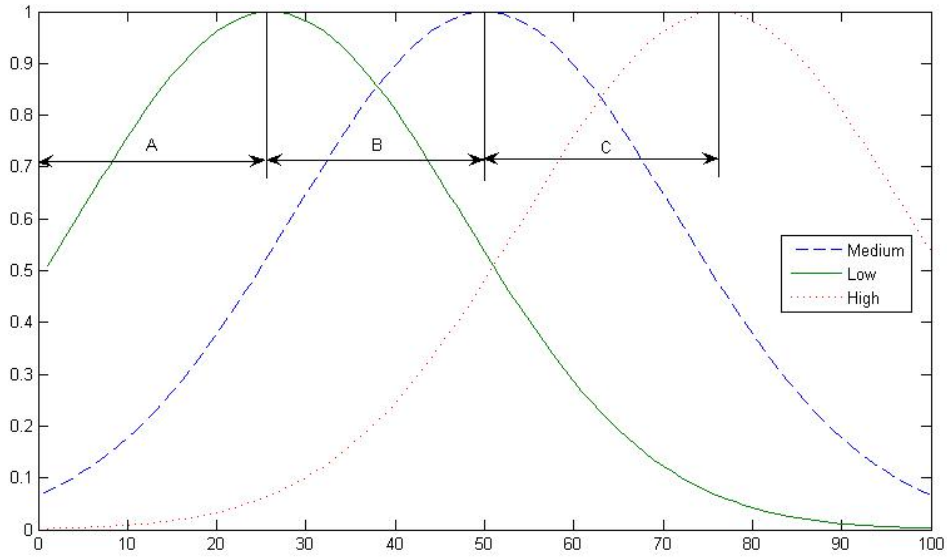


Figure 4.5 the sample fuzzy sets of local speed

Certainly, by coding the center points of Local Occupancy and Local Flow in the same way, we could have other two inequality constrains:

$$4000 - A-B-C \geq 0 \text{ and } 30 - A-B-C \geq 0 \qquad (4.4)$$

Then each generated individual could be tested by these constrains to find a feasible individual and finally accumulated to be a feasible population.

Figure 4.6 shows the layout of generating a feasible individual in C++

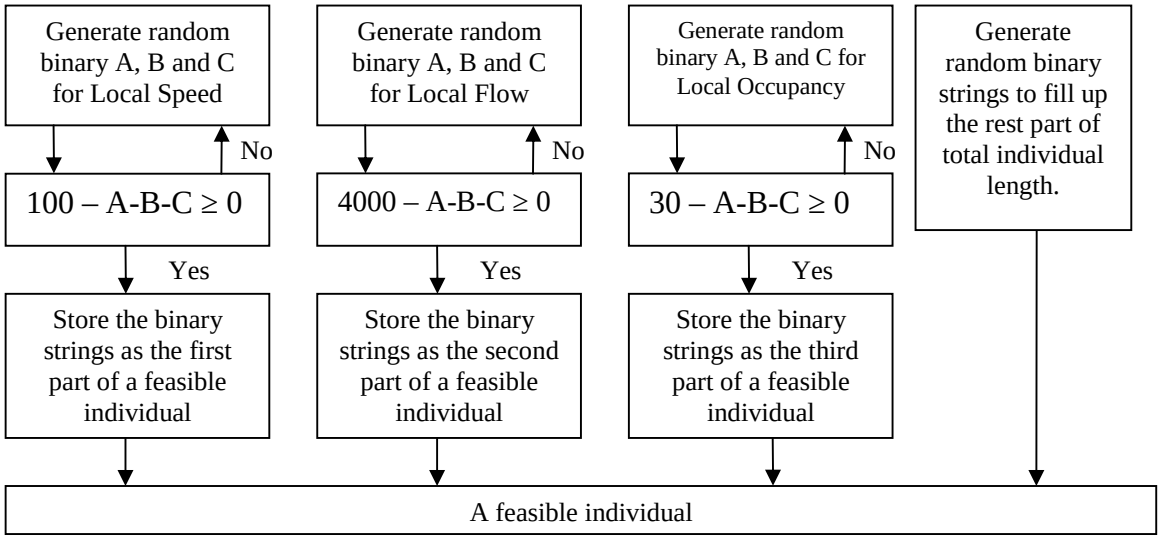


Figure 4.6 the programming layout of generating a feasible individual

The c++ codes are given as blow:

```
// Generate random binary A, B and C for Local Speed
int ispop=0;
int i=1
while(!ispop)
{
   for(int j=0; j<3*speedLength; j++)
   { population[i][j]=rand()%2;}
   //caculate A
   double Ls=Lspeed(population[i], speedLength, domain);
   //caculate B
   double Ms=Mspeed(population[i], speedLength, domain);
   //caculate C
   double Hs=Hspeed(population[i], speedLength, domain);
   if((100-Ls-Ms-Hs)>=0)
   {ispop=1;}
}
ispop=0;

// Generate random binary A, B and C for Local Flow
while(!ispop)
{
   for(int j=(3*speedLength-1); j<(3*speedLength+3*flowLength); j++)
   { population[i][j]=rand()%2;}
   //caculate A
   double Lf=Lflow(population[i], speedLength, flowLength, domain);
   //caculate B
   double Mf=Mflow(population[i], speedLength, flowLength, domain);
   //caculate C
   double Hf=Hflow(Population[i], speedLength, flowLength, domain);
   if((4000-Lf-Mf-Hf)>=0)
   {ispop=1;}
}
```

```
 ispop=0;

 // Generate random binary A, B and C for Local Occupancy
 while(!ispop)
{
    for(int j=(3*speedLength+3*flowLength-1);
    j<(3*speedLength+3*flowLength+3*occLength); j++)
    {population[i][j]=rand()%2;}
    //caculate A
    double Lo=Locc(population[i], speedLength, flowLength, occLength,
    domain);
    //caculate B
    double Mo=Mocc(population[i], speedLength, flowLength, occLength,
    domain);
    //caculate C
    double Ho=Hocc(population[i], speedLength, flowLength, occLength,
    domain);
    if((30-Lo-Mo-Ho)>=0)
    {ispop=1;}
 }
 ispop=0;

 //generate the random binary strings to fill up the total length
 for(int j=(3*speedLength+3*flowLength+3*occLength-1);
 j<Total_length;j++)
 {population[i][j]=rand()%2;}
```

4.2.2.2 Objective function

Figure 4.7 shows an uncontrolled motorway on-ramp model.

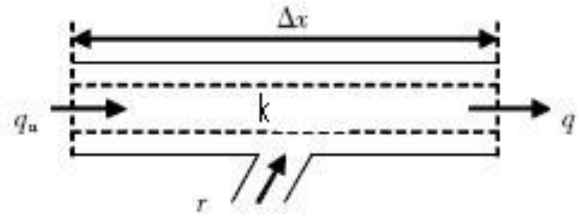


Figure 4.7 Fundamental diagrams

Where:

Δx (m) is the length of section.

q_u (vehs/h) is the upstream traffic flow.

q (vehs/h) is the downstream traffic flow.

r (vehs/h) is the ramp inflow.

K (vehs/km) is the average flow density of the motorway section.

If N is the total number of the vehicles in the motorway section during a time interval, Δt, an equation could be given as:

$$N(t+\Delta t) = N(t) + \Delta t \times [q_u(t) - q(t) + r(t)] \tag{4.5}$$

If the equation is divided by Δx on both sides, it could become:

$$K(t+\Delta t) = K(t) + \Delta t / \Delta x \times [q_u(t) - q(t) + r(t)] \tag{4.6}$$

This equation shows a basic relationship between traffic density and traffic flow for a motorway on-ramp model. It is supposed that $K_{congestion}$ density, the critical density, is predefined, which could be expressed as $K_{congestion} = N_{congestion}/\Delta x$ (where $N_{congestion}$ is the number of vehicles staying in motorway section when congestion happens.) Then, we could obtain an equation to describe the maximum allowed inflow density of the whole ramp section for the next time interval:

$$\begin{aligned} K_{\max}(t+\Delta t) &= K_{congestion} - K(t+\Delta t) \\ &= K_{congestion} - \Delta t / \Delta x \times [q_u(t) - q(t) + r(t)] - K(t) \end{aligned} \tag{4.7}$$

And if the equation above times Δx on both sides, it becomes

$$N_{\max}(t+\Delta t) = N_{congestion} - N_u + N_d - N_r - N(t) \tag{4.8}$$

Where:

N_{max} (t+Δt) is the maximum number of vehicles allowed to get in the motorway section without causing congestion for the next time interval.

$N_{congestion}$ is the number of vehicles staying in motorway section when congestion just happens.

N_r is the number of vehicles getting in the motorway section from the ramp.

N_d is the number of vehicles leaving the motorway section.

N_u is the number of vehicles getting in the motorway section from the upstream section.

N (t) is the number of vehicles staying motorway section.

Then, an ideal ramp metering rate for next time interval (minutes) could be given as:

$$R_{ideal}(t+\Delta t) = 60 \times N_{\max}(t+\Delta t)/\Delta t$$

$$= 60 \times (N_{congestion} - N_u + N_d - N_r - N(t))/\Delta t \quad (4.9)$$

When Δx is appropriately chosen and Δt is reasonable short, the ideal ramp metering rate can prevent the motorway section from congestion and fully utilize the road capacity for next time interval. Based on this assumption, an objective function or fitness function could be given as:

$$1/(R_{ideal}(t+\Delta t) - R_{evolution})^2 \quad (4.10)$$

$R_{evolution}$ is a metering rate generated by FLC controller with the updated parameters (the feasible individuals), based on the traffic information of last five minutes. The optimization of fuzzy ramp control approach is actually to find a certain set of fuzzy parameters, which is able to generate a ramp metering rate close to the R_{ideal} of the next time interval. Figure 4.8 shows the programming layout of calculating a fitness value for a feasible individual by the objective functions above.

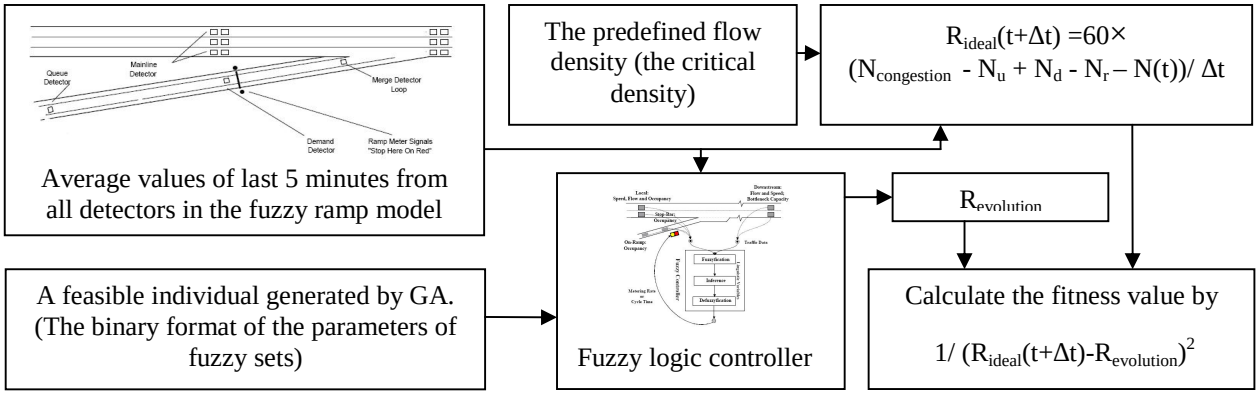


Figure 4.8 the layout of calculating a fitness value

4.2.2.3 Selection

When a feasible initial population is generated and each individual's fitness could be calculated by a fitness function, we apparently need a selection algorithm to pick up the individuals with higher fitness as much as possible. A popular selection algorithm, "Roulette Wheel", is used in programming. This method works in a way that is analogous to a roulette wheel. Each individual in a population is allocated a part of a wheel, and the size of the part is in proportion to the individual's fitness. A random generated number is used as a spinning pointer to select the individual that is involved to next generation. The selection continues until the new population is full.

The procedures of "Roulette Wheel" algorithm are shown below:

Step1. Calculate the total fitness (F) of the last population or initial population

$$F := \sum_{i=0}^{M-1} f(v_i) \tag{4.11}$$

(Where M is the size of population, f is the fitness function and v_i is a individual)

Step2. Calculate the probability of a selection p_i and the cumulative probability q_i for each individual v_i.

$$P_i := \frac{f(v_i)}{F}, q_i := \sum_{k=0}^{i} p_k \quad I = 0,1...M-1. \tag{4.12}$$

(Where $q_{M-1} = 1$)

Step3. Generate a random binary number with M bits for the range [0, 1] and given as r_i.

Step4. Select the individual of k+1, when $q_k < r_i < q_{k+1}$.

Step5. Back to step3 until r_M is reached.

Figure 4.9 shows the flow chart of the programming for Selection:

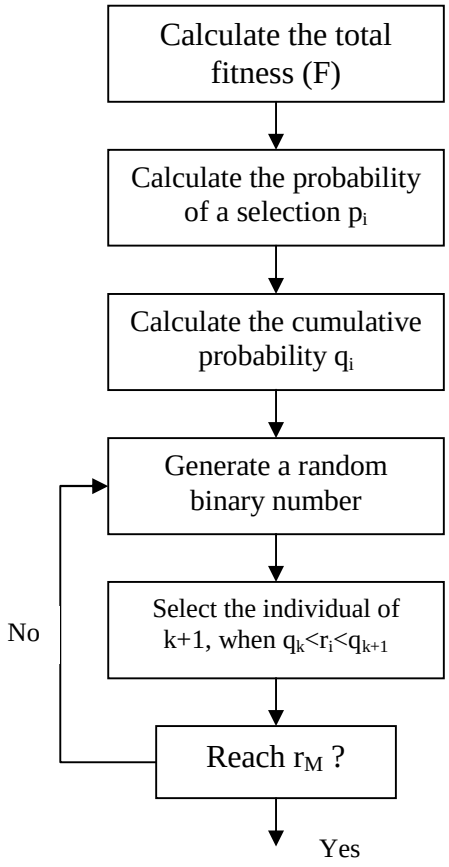


Figure 4.9 the flow chart of Selection

4.2.2.4 Crossover

Once selection is finished, the recombination operator called crossover will be applied to the selected population. Single Point Crossover known as the most basic crossover algorithm has been used in programming. Since Single Point Crossover will swap two strings at a random point to create new individuals (Figure 4-8), it is possible for new individuals not to satisfy constrains such as $4000 - A\text{-}B\text{-}C \geq 0$ and $30 - A\text{-}B\text{-}C \geq 0$. Therefore, different with the standard Single Point Crossover Algorithm, a constraint checking loop will be added to the standard procedures of Single Point Crossover to ensure the feasibility of swapped individuals.

Parent 1	101|11100
Parent 2	110|10010
Child 1	101 10010
Child 2	110 11100

Figure 4.10 the Single Point Crossover

The crossover method used in programming could be show as the following procedure:

Step1. Set a default value (0.4) for the probability of crossover.

Step2. Generate a group of random numbers in the range [0, 1] with the same size as that of the population. Each random number (R_i) in the group represents the individual at the identical position in the population.

Step3. Select all individuals with the corresponding R_i less than 0.4 for crossover.

Step4. Check the numbers of the selected individual. If it is odd, we add one extra individual with R_i <0.4. By doing so, we could pair the select individuals for crossover.

Step5. Choose a pair of selected individuals.

Step6. Backup the pair of individuals and generate a random integer number POS in the range [0, N-2] (where N is the total length of an individual). The number POS indicates the position of the crossing point, at which, the pair of the backup individuals are swapped over.

Step7. Check the swapped individuals whether they are within constraints, such as 4000-A-B-C<0 and 30-A-B-C<0, or not. If yes, we swap the original individuals and go back to Step8. If not, we go back to Step 6.

Step8. Move to next pair of selected individuals, and repeat Step5 until all selected individuals are finished.

Figure 4.11 shows the flow chart of the programming for Crossover.

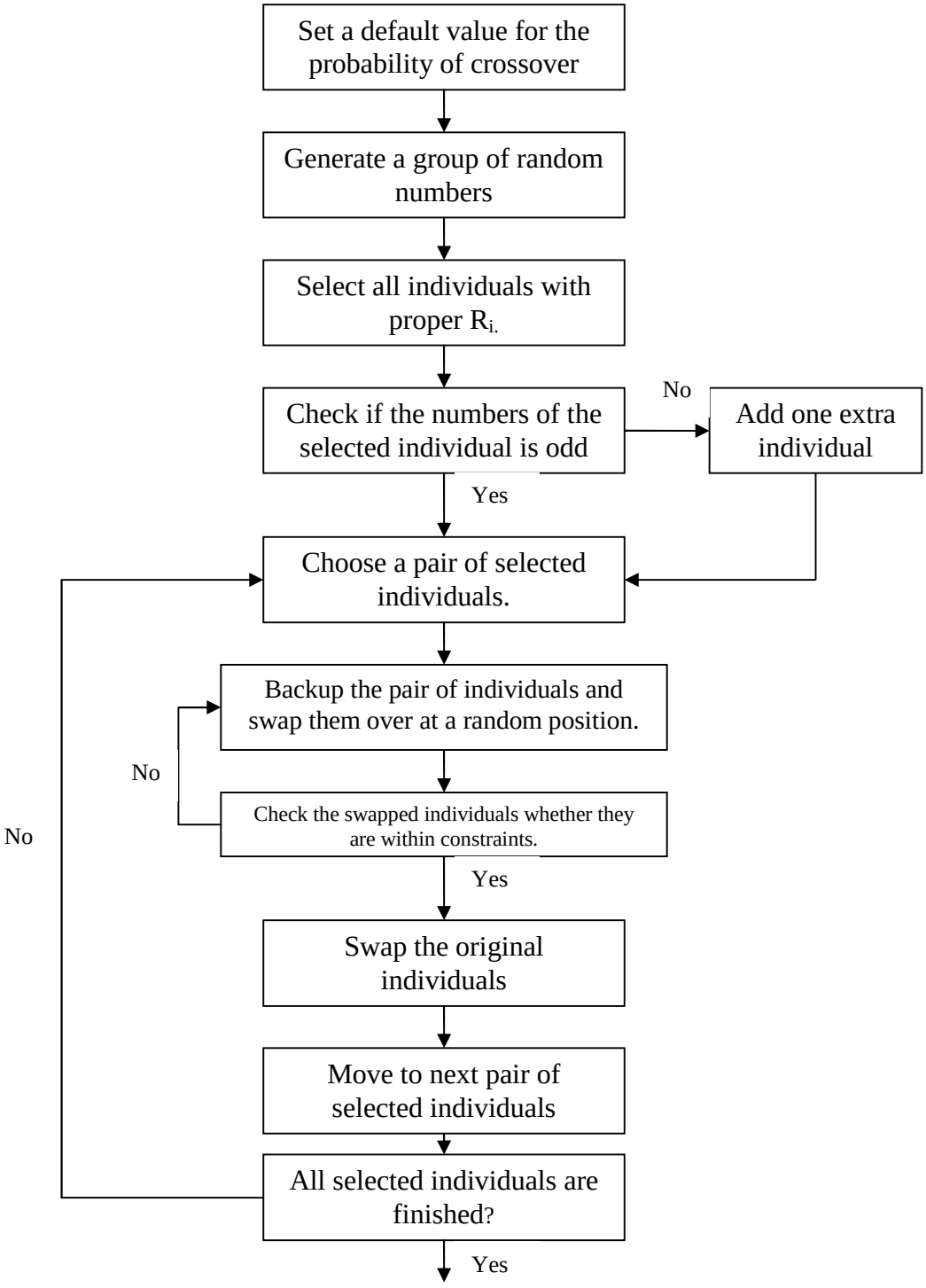


Figure 4.11 the flow chart of Crossover

4.2.2.5 Mutation

The next operator, mutation ensures that the probability of searching a given part of the solution space is never zero, which is performed on a bit-by-bit basis. It generates a random number for each bit of the population and if the number is less than the specified mutation probability, the corresponding bit is flipped, i.e., if the bit is a 1, it becomes 0 and vice versa. Also, a constraint checking loop is added in programming to filter the infeasible individuals.

The method of mutation could be generalized in the following steps:

Step1. Set a default value (0.01) for mutation.

Step2. Calculate the total bits of the population by N×M, where N is the total length of one individual and M is the size of population.

Step3. Generate N×M random numbers in the range [0, 1]. Each random number (R_i) in the group represents the bit at the identical position in the population.

Step4. Select a bit in the population where the corresponding R_i is less than 0.01.

Step5. Calculate which individual the selected bit belongs to in the population

Step6. Backup the corresponding individual.

Step7. Flip the selected bit in the backup individual.

Step8. Check the flipped individuals whether they are within constraints, such as 4000-A-B-C<0 and 30-A-B-C<0, or not. If yes, we flip the bit in the original individuals and go back to Step4. If not, recovery the individuals and go back to step 4.

Figure 4.12 shows the flow chart of the programming for Mutation.

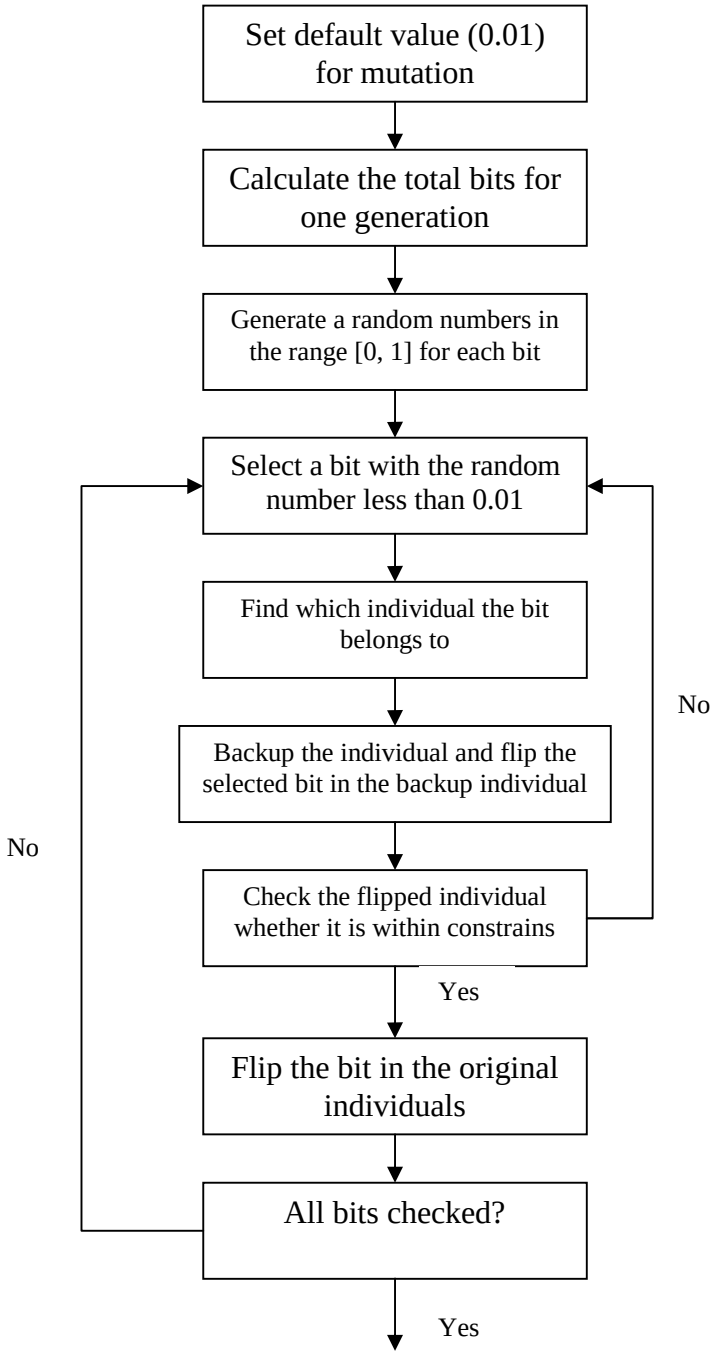


Figure 4.12 the flow chart of Mutation

4.2.2.6 Encoding and Decoding

Although Genetic Algorithms can be performed using either binary or real-valued encodings, binary encoding is applied in this genetic tuning process since it allows for greater solution space and for more combinations of alleles.

For binary encoding, each tuned parameter is converted into a binary string with a fixed length, and the content of the binary string is generated randomly within constraints, which has been discussed in 4.3.2.1. For binary decoding, which is normally performed when the best individual is found or when a random individual needs to be ensured within constraints, the binary strings of the tuned parameters will be converted into real values. The procedure could be explained by the following example:

Suppose that we have to map the binary string into a real number x with a given range [a, b] and the binary string is denoted by

$$s_{n-1,}s_{n-2}\ldots s_{1,}s_0$$

(Where s_0 is the least significant bit (LSB) and s_{n-1} is the most significant bit (MSB))

We could convert the binary string from base 2 to base 10 by

$$m = \sum_{i=0}^{N-1} s_i 2^i \tag{4.13}$$

Then corresponding real value could be given by

$$x = a + m\frac{b-a}{2^n - 1} \tag{4.14}$$

The C++ codes blow shows that the fuzzy parameter, Local Speed Low, has been converted from a binary string to a real number.

```
double Lspeed(int* chromosome, int speedLength, double* domain)
{
 double m=0.0;
 for(int i=0; i<speedLength; i++)
 {m+=chromosome[speedLength-i-1]*pow(2.0,i);}
 double x=domain[0]+m*(domain[1]-domain[0])/(pow(2.0,speedLength)-1.0);
 return x;
}
```

4.3 Investigating GA Parameters

The performance of the genetic tuning process has greatly affected by the parameters of GA, such as population size, mutation probability and crossover probability. It is necessary to perform an experiment for the investigation of how these factors affect GA. By doing so, the GA parameters could be found in the suitable range.

For the objective function, $1/(R_{ideal}(t+\Delta t)-R_{evolution})^2$, if R_{ideal} is defined as 300veh/h and all inputs of fuzzy logic controller is set to zero, we run the GA program in Microsoft C++ environment and generate the optimized metering rate based on different GA parameters. The results are given as the following table (each row in the table has been run 10 times):

Table 4.2 the test of GA parameters

Generation	Population size	Crossover Probability	Mutation Probability	The range of optimized metering rates
400	50	0.25	0.01	293.56~310.87
300	50	0.25	0.01	283.23~307.71
200	50	0.25	0.01	274.01~306.65
150	50	0.25	0.01	220.20~310.65
100	50	0.25	0.01	224.30~337.99
400	60	0.25	0.01	292.03~311.25
400	40	0.25	0.01	290.25~326.26
400	30	0.25	0.01	275.01~307.24
300	60	0.25	0.01	285.23~310.71
300	40	0.25	0.01	269.56~326.75
300	30	0.25	0.01	275.87~310.25
400	50	0.10	0.01	273.98~302.85
400	50	0.20	0.01	284.21~306.21
400	50	0.30	0.01	290.21~310.23
400	50	0.40	0.01	295.12~307.26 (chose)
400	50	0.50	0.01	293.21~305.54

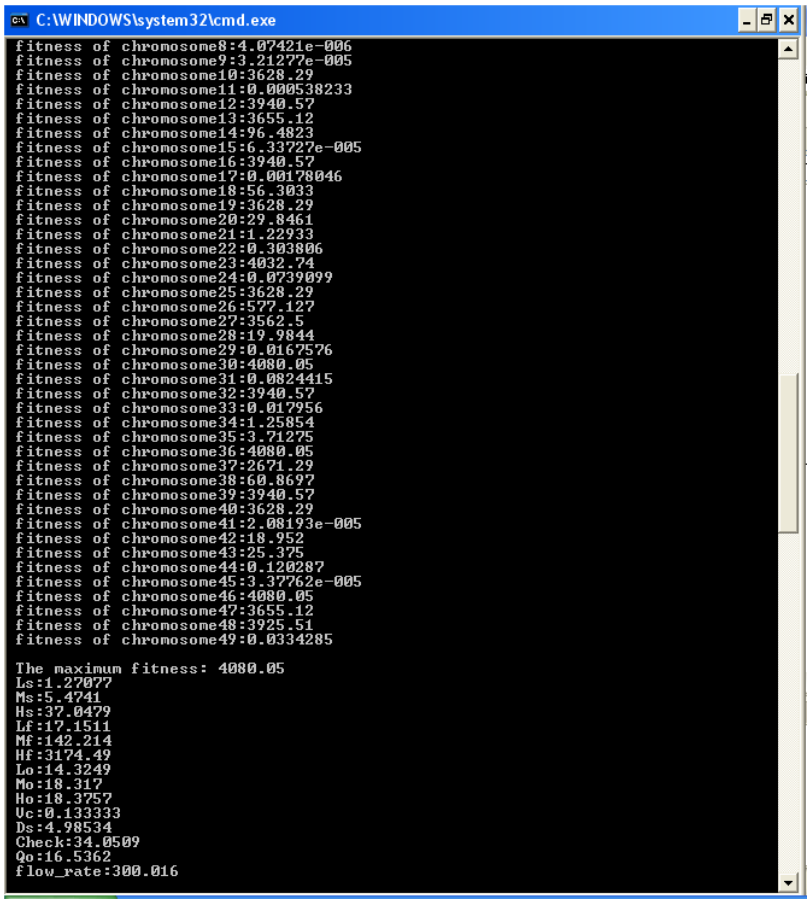
During the test when GA is performed more than 400 generations and the population size is more than 60, GA performance could be better, but it is more time-consuming. Therefore we finally select the GA parameters blow:

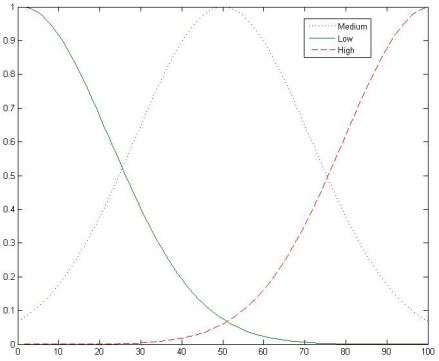
Population size: 50

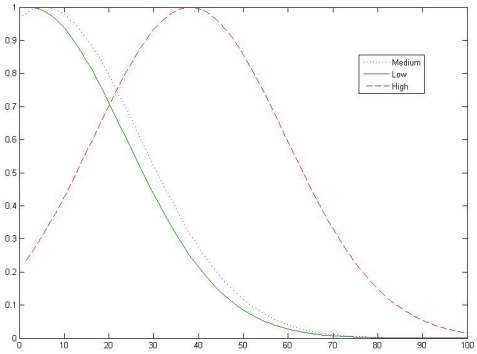
Generation: 400

Crossover rate: 0.4

Figure 4.13 shows one result generated by GA in Microsoft C++ and we can see the shift of membership functions (local speed) after tuning the fuzzy parameters.







(Original fuzzy sets) **(The tuned fuzzy sets)**

Figure 4.13 Results from GA test in Microsoft C++

4.4 Conclusion

The genetic tuning algorithm presented in this chapter is to optimize a typical fuzzy ramp metering algorithm as a local traffic responsive ramp metering algorithm. The objective function of genetic tuning process focuses on the adjustment of ramp metering rates based on the critical flow density of the on-ramp section.

During the tuning process, the fuzzy parameters will be converted into binary code to perform the standard GA operators: selection, crossover and mutation. The tuned parameters will be applied to the fuzzy logic controller to generate optimized metering rates for next time interval (5 minutes). Meanwhile, the fuzzy logic ramp meter still generates a ramp metering rate based on local traffic condition every minute as a real time control approach.

To test and evaluate the proposed control approach, computer simulation has no difficulties and limitations as field implementation, so the genetic fuzzy ramp metering algorithm would be implemented and evaluated in a stochastic microscopic traffic simulator, Aimsun.

Chapter 5 Simulation study

This chapter will present the traffic simulations for the proposed ramp metering algorithms and the comparison of the performance of FLC and genetic fuzzy ramp metering will be given and analyzed based on the simulation results. Also, Sensitivity Analysis will be introduced in this chapter as the first step of model optimization.

5.1 Aimsun 6 simulation environment

With the increase of popularity of traffic simulation software packages, a number of commercial traffic simulation packages, such as such as CORSIM (USA), PARAMICS (UK), AIMSUN (Spain) and VISSIM (Germany) are developed to analyze and predict traffic flow conditions.

Aimsun6, as a microscopic, stochastic traffic simulator, is used to be the simulation environment for testing the algorithms we proposed before. The microsimulator of Aimsun6 follows a microscopic approach [23] to continuously model each vehicle behavior in the network according to driver's behavior model such as car flowing and lane changing. The traffic simulation provides the collective behavior of all vehicle-driver units within the range of network geometries. In addition, most traffic equipment present in a real traffic network is also modeled in the microsimulator like traffic lights, traffic detectors, VMS (Variable Message Signs) and ramp metering devices, etc.

Figure 5.1 shows the layout of Aimsun environment [24].

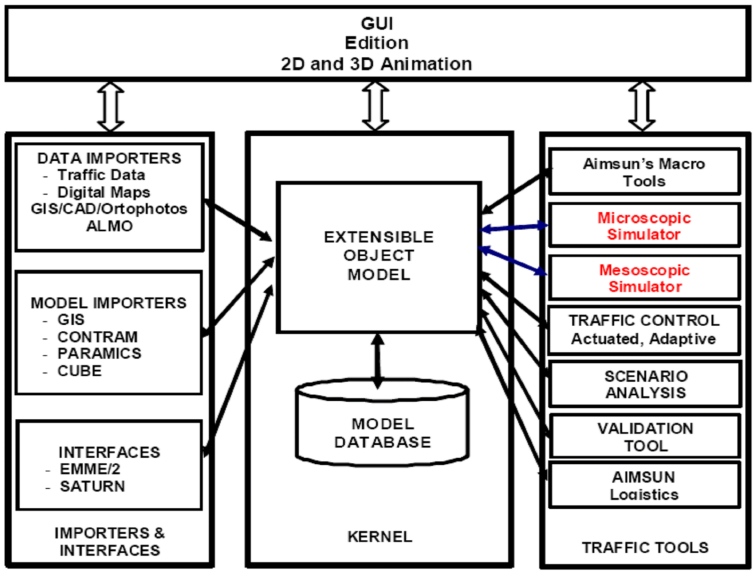


Figure 5.1 Aimsun environment [24]

5.2 Simulator Enhancements

Since Aimsun is unable to implement adaptive traffic control with the standard software pack, the Aimsun API module has been used to enable the communication between the Aimsun simulation model and a user-built control algorithm.

Figure 5.2 illustrates the conceptual structure of how Aimsun working with user application by means of Aimsun API module:

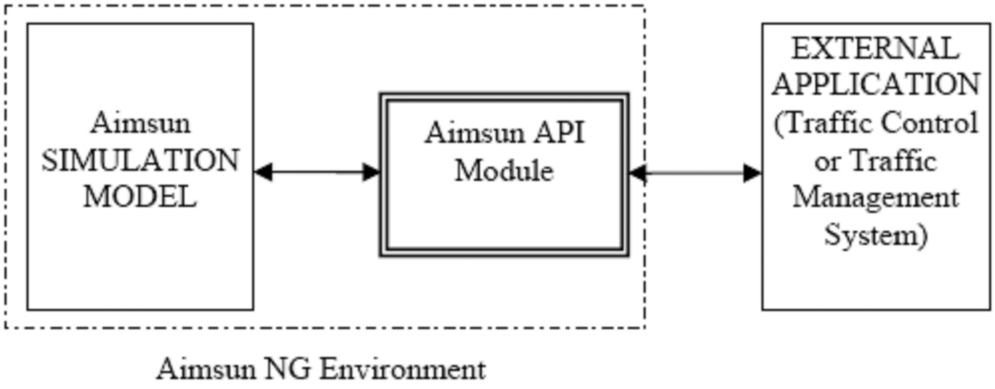


Figure 5.2 Conceptual structure of Aimsun API application [25]

The Aimsun API module provides a set of functions to collect the required data (e.g. flow, occupancy, etc.) from traffic simulation. Based on the collected information, the EXTERNAL APPLICATION (user-built control algorithm) makes some control decisions which will be applied to the simulation. Such a process completes the communication between the Aimsun simulation model and a user-built control algorithm.

The communication process is guaranteed by eight high level functions defined in Aimsun API module: AAPILoad, AAPIInit, AAPIManage, AAPIPostManage, AAPIFinish, AAPIUnLoad, AAPIEnterVehicle and AAPIExitVehicle [25].

- AAPILoad (): It is called when the module is loaded by Aimsun.
- AAPIInit (): It is called when Aimsun starts the simulation and can be used to initialise whatever the module needs.
- AAPIManage (): This is called in every simulation step at the beginning of the cycle, and can be used to request detector measures, vehicle information and interact with junctions, metering and VMS in order to implement the control and management policy.

- AAPIPostManage (): This is called in every simulation step at the end of the cycle, and can be used to request detector measures, vehicle information and interact with junctions, metering and VMS in order to implement the control and management policy.
- AAPIFinish (): It is called when Aimsun finish the simulation and can be used to finish whatever the module needs.
- AAPIUnLoad (): It is called when the module is unloaded by Aimsun.

The scheme of how Aimsun interacts with Aimsun API is shown in Figure 5.3.

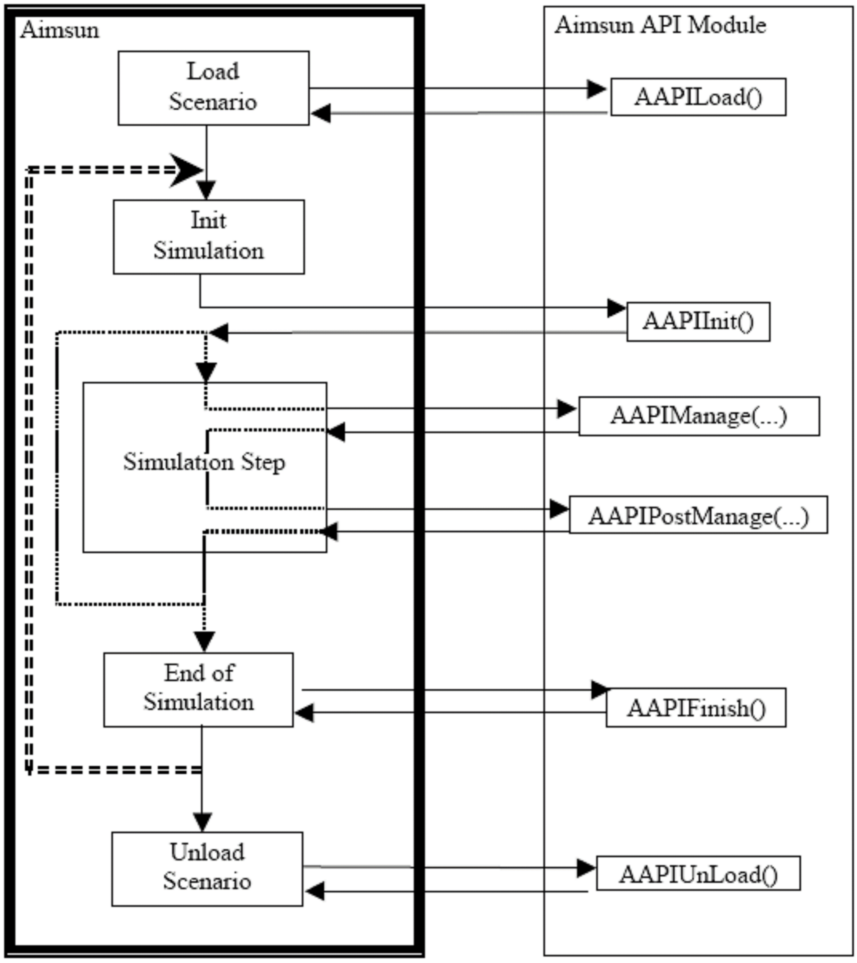


Figure 5.3 Interactions between Aimsun and Aimsun API [25]

The proposed ramp metering algorithms programmed in Microsoft Visual C++ will be implemented in Aimsun simulator through AAPIManage () and AAPIPostManage () function by means of Microsoft Visual Studio 2005, where a Dynamic Link Library (DLL) will be generated and integrated to the simulator.

5.3 Study area and model calibration

The study area is located at the southbound on-ramp of Constellation interchange in Auckland North motorway (Figure 5.4).

Figure 5.4 the southbound on-ramp of Constellation interchange in Aimsun

5.3.1 Road section information

The motorway geometric layout is acquired from the picture of Google map, and detailed information is obtained from the construction drawing provided by Transit New Zealand. Figure 5.5 shows the geometric information of the study place.

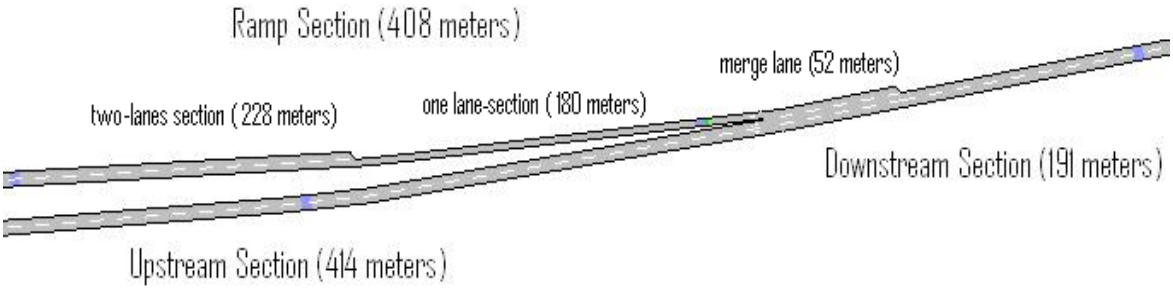


Figure 5.5 the geometric information of the on-ramp of Constellation Dr in Aimsun

The geometric information of the on-ramp model is defined in Aimsun as follows:

-The length of ramp section: 408 meters.

-The length of downstream motorway section: 191 meters.

-The length of upstream motorway section: 414 meters.

-The length of merge lane section: 52 meters.

Table 5-1 shows the basic road section parameters [26] [27]

Table 5.1 Basic road information

Section Type	Speed Limit	Capacity
Freeway	100km/hour	2500veh/hour/lane
On ramp	90km/hour	1600veh/hour/lane

5.3.2 Vehicle information

No bus are used in this simulation because since the mid of 2007, buses travelling to and from North shore have their own lanes which are independent from the motorway .

No high occupancy vehicles (HOVs) are allocated in this simulation since the presence of small percentage HOVs will not cause obvious variation of simulation results but seriously slow down the simulation process especially when genetic algorithm is already time-consuming.

The parameters of the general car are shown in Figure 5.6.

Name	Mean	Deviation	Min	Max	Units
Length	4.4	0.43	4	5	meters
Width	1.8	0.1	1.6	2.2	meters
Max Desired Speed	96	15.3	70	135	km/h
Max Acceleration	3.5	0.2	3	5	m/s2
Normal Deceleration	4	0	4	4	m/s2
Max Deceleration	6	0	6	6	m/s2
Speed Acceptance	1	0	1	1	
Min Distance Veh	1	0	1	1	meters
Give Way Time	20	5	0.01	30	Secs
Guidance Acceptance	100	0	100	100	%
Sensitivity Factor	1	0	1	1	
Minimum Headway	0	0	0	0	Secs

Figure 5.6 Vehicle parameters

5.3.3 Detector information

To implement the proposed control approaches, several detectors are installed on the road section. The detected information includes occupancy, vehicle speed and vehicle count. The detection interval is 1 minute.

Figure 5.7 shows the distribution of detectors installed on the motorway sections.

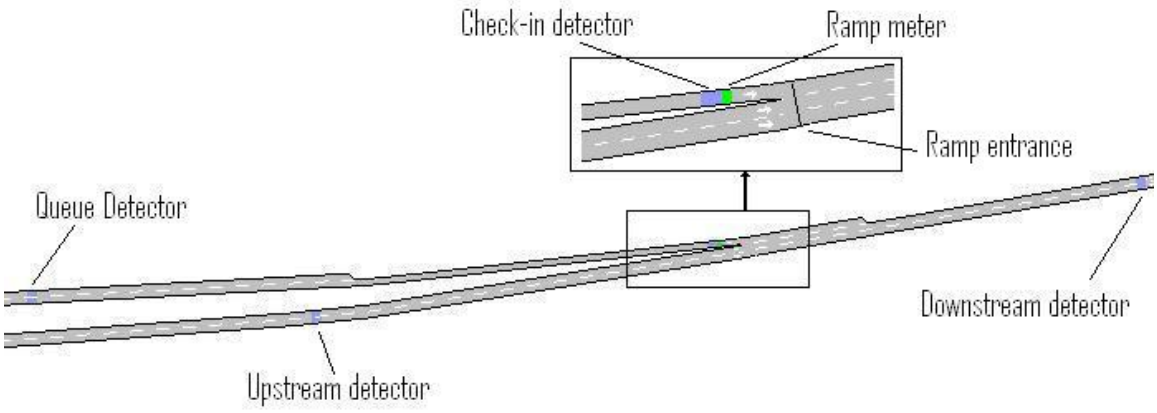


Figure 5.7 the distribution layout of detectors

The locations of detectors are set in Aimsun as the following information:

-Ramp Queue detector: 400 meters from on-ramp entrance.

-Ramp Check-in detector: 9 meters from on-ramp entrance.

-Upstream Detector: 200 meters from on-ramp entrance.

-Downstream Detector: 180 metres from on-ramp entrance.

5.3.4 Traffic Flow assumption

An Aimsun traffic network model could define the traffic demand data in two different ways, an O/D matrix or the traffic flows at the sections. This ramp model is using the traffic flows that include upstream traffic demand and ramp traffic demand. These traffic demands will change each 15 minutes and the duration of the simulation is 1 hour.

For an isolated on-ramp model, normally the congestion happens when the sum of the ramp traffic demand and upstream traffic demand exceeds or nearly reaches the downstream traffic capacity. Therefore, the total traffic demand (upstream traffic demand plus ramp traffic demand) will be set around 5000vehicles/hour to test the performance change of the proposed algorithms under the situation where congestion is going to be or already formed because downstream road capacity is 5000 vehicles/hour (lane capacity times two). By doing so, the simulation is able to find the specific ranges of traffic flow, within which, the control algorithms work properly.

Average ramp demand is from 1000vehicles/hour to 1600vehicles/hour (ramp capacity) with an increase of 200vehicles/hour. Table 5.2 to Table 5.5 shows the traffic demand information used in the Aimsun on-ramp model.

Table 5.2 Traffic demand data when average ramp demand is 1600vehicles/h

No.	Location (vehicles/h)	8:00am~ 8:15am	8:15am~ 8:30am	8:30am~ 8:45am	8:45am~ 9:00am	Average Demand	Total Demand
1	**Upstream Demand**	4400	3600	4200	3800	**4000**	**5600**
	Ramp Demand	1800	1400	1400	1800	**1600**	
2	**Upstream Demand**	4200	3400	4000	3600	**3800**	**5400**
	Ramp Demand	1800	1400	1400	1800	**1600**	
3	**Upstream Demand**	4000	3200	3800	3400	**3600**	**5200**
	Ramp Demand	1800	1400	1400	1800	**1600**	
4	**Upstream Demand**	3800	3000	3600	3200	**3400**	**5000**
	Ramp Demand	1800	1400	1400	1800	**1600**	
5	**Upstream Demand**	3600	2800	3400	3000	**3200**	**4800**
	Ramp Demand	1800	1400	1400	1800	**1600**	
6	**Upstream Demand**	3400	2600	3200	2800	**3000**	**4600**
	Ramp Demand	1800	1400	1400	1800	**1600**	

Table 5.3 Traffic demand data when average ramp demand is 1400vehicles/h

No.	Location (vehicles/h)	8:00am~ 8:15am	8:15am~ 8:30am	8:30am~ 8:45am	8:45am~ 9:00am	Average Demand	Total Demand
7	**Upstream Demand**	4600	3800	4400	4000	**4200**	**5600**
	Ramp Demand	1600	1200	1200	1600	**1400**	
8	**Upstream Demand**	4400	3600	4200	3800	**4000**	**5400**
	Ramp Demand	1600	1200	1200	1600	**1400**	
9	**Upstream Demand**	4200	3400	4000	3600	**3800**	**5200**
	Ramp Demand	1600	1200	1200	1600	**1400**	
10	**Upstream Demand**	4000	3200	3800	3400	**3600**	**5000**
	Ramp Demand	1600	1200	1200	1600	**1400**	
11	**Upstream Demand**	3800	3000	3600	3200	**3400**	**4800**
	Ramp Demand	1600	1200	1200	1600	**1400**	
12	**Upstream Demand**	3600	2800	3400	3000	**3200**	**4600**
	Ramp Demand	1600	1200	1200	1600	**1400**	

Table 5.4 Traffic demand data when average ramp demand is 1200vehicles/h

No.	Location (vehicles/h)	8:00am~ 8:15am	8:15am~ 8:30am	8:30am~ 8:45am	8:45am~ 9:00am	Average Demand	Total Demand
13	**Upstream Demand**	4600	3800	4400	4000	**4200**	**5400**
	Ramp Demand	1400	1000	1000	1400	**1200**	
14	**Upstream Demand**	4400	3600	4200	3800	**4000**	**5200**
	Ramp Demand	1400	1000	1000	1400	**1200**	
15	**Upstream Demand**	4200	3400	4000	3600	**3800**	**5000**
	Ramp Demand	1400	1000	1000	1400	**1200**	
16	**Upstream Demand**	4000	3200	3800	3400	**3600**	**4800**
	Ramp Demand	1400	1000	1000	1400	**1200**	
17	**Upstream Demand**	3800	3000	3600	3200	**3400**	**4600**
	Ramp Demand	1400	1000	1000	1400	**1200**	
18	**Upstream Demand**	3600	2800	3400	3000	**3200**	**4400**
	Ramp Demand	1400	1000	1000	1400	**1200**	

Table 5.5 Traffic demand data when average ramp demand is 1000vehicles/h

No.	Location (vehicles/h)	8:00am~ 8:15am	8:15am~ 8:30am	8:30am~ 8:45am	8:45am~ 9:00am	Average Demand	Total Demand
19	**Upstream Demand**	4600	3800	4400	4000	**4200**	**5200**
	Ramp Demand	1200	800	800	1200	**1000**	
20	**Upstream Demand**	4400	3600	4200	3800	**4000**	**5000**
	Ramp Demand	1200	800	800	1200	**1000**	
21	**Upstream Demand**	4200	3400	4000	3600	**3800**	**4800**
	Ramp Demand	1200	800	800	1200	**1000**	
22	**Upstream Demand**	4000	3200	3800	3400	**3600**	**4600**
	Ramp Demand	1200	800	800	1200	**1000**	
23	**Upstream Demand**	3800	3000	3600	3200	**3400**	**4400**
	Ramp Demand	1200	800	800	1200	**1000**	
24	**Upstream Demand**	3600	2800	3400	3000	**3200**	**4200**
	Ramp Demand	1200	800	800	1200	**1000**	

Totally four ramp demands is used in this simulation: 1000vehicles/hour, 1200vehicles/hour, 1400vehicles/hour and 1600vehicles/hour, which cover the possible range of ramp demands when the downstream congestion could happen. The situation about less than 1000vehicels/hour ramp demand will not be discussed in this paper for two reasons:

a) When average ramp demand is less than 1000vehicles/hour, average upstream demand should reach more than 4000vehicles/hour to cause downstream congestion. But from the observation of field data, average upstream demand barely exceeds 4200vehicles/hour, so this situation is not necessary to be discussed.

b) When average upstream demand is less than 4000vehicles/hour and average ramp demand is less than 1000vehicles/hour, the total traffic demand hardly reaches road capacity (5000 vehicles/hour), so the motorway is under the free flow condition, where ramp metering will not benefit traffic condition and may cause extra traffic delay.

5.3.5 The calculation of the objective function in Aimsun

As we discussed in the last chapter, the fuzzy genetic ramp metering control will adjust the ramp metering rates ($R_{evolution}$) based on an ideal metering rate (R_{ideal}) for the ramp section (Δx) by means of an objective function, $1/(R_{ideal}(t+\Delta t) - R_{evolution})^2$

And R_{ideal} is given as:

$$R_{ideal}(t+\Delta t) = 60 \times (N_{congestion} - N_u + N_d - N_r - N(t))/\Delta t \qquad (5.1)$$

Where:

$N_{congestion}$ is the number of vehicles staying in motorway section when congestion just happens.

N_r is the number of vehicles getting in the motorway section from the ramp.

N_d is the number of vehicles leaving the motorway section.

N_u is the number of vehicles getting in the motorway section from the upstream section.

N (t) is the number of vehicles staying motorway section.

In this simulation, Δx starts from the upstream detector and ends to the downstream detector (so the length of Δx is 380meters); Δt is given by 5 minutes; N_r is counted by the check-in detector; N_d is counted by the downstream detector; N_u is counted by the upstream detector; N (t) is counted by $N_u + N_d + N_r + N(t-1)$ during the whole simulation. Since the default jam density in Aimsun is 200vehicles/km, the critical density could be estimated by Greenshield's macroscopic stream model (Figure5.8), which is half of jam density, 100vehicles/km.

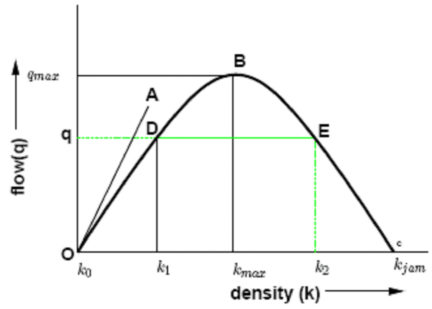


Figure 5.8 Relation between flow and density in Greenshield's macroscopic stream model

To further prevent the formation of congestion, the predefined density could be lower than the critical point, so the congestion density is set to 90vehicles/km. Then $N_{congetion}$ could be given as [(90/1000) ×380] ≈35.

Finally, the equation 5.1 is calculated as:

$$R_{ideal}(t+\Delta t) = (35 - (N_u - N_d + N_r + N(t))/5) \times 60 \qquad (5.2)$$

5.4 Simulation results and analysis

To evaluate the performance of the proposed ramp metering algorithms, the following Measures of Effectiveness (MOE) are selected: motorway downstream MOEs, ramp MOEs and system MOEs. The specific measures of effectiveness are:

Motorway Downstream Performance MOEs

1) Total Downstream Travel Time (seconds per vehicle): Total time experienced by all vehicle travelling on the motorway downstream section per kilometre.
2) Average Downstream Delay Time (seconds per vehicle): Average delay time per vehicle while travelling on the motorway downstream section.
3) Average Downstream Flow Rate (vehicles per hour): The number of vehicles travelling on the motorway downstream section during the simulation time (one hour).
4) Average Downstream Speed (kms per hour): Space mean speed for vehicles travelling one the motorway downstream section.

Ramp MOEs

5) Total Ramp Travel Time (seconds per vehicle): Total time experienced by all vehicle travelling on the ramp section per kilometre.
6) Average Ramp Delay (seconds per vehicle): Average delay time per vehicle while travelling on the ramp section.
7) Average Flow Rate (vehicles per hour): The number of vehicles travelling on the ramp section during the simulation (one hour).

System MOEs

8) Total Travel Time (hours): Total travel time accumulated by all vehicles travelling in the Aimsun traffic network.

9) Average Delay Time (seconds per vehicle per km): Average delay time per vehicle while travelling in the Aimsun traffic network per kilometre.

To show the difference of the performance of FLC ramp metering and Genetic fuzzy ramp metering, the simulation results will be given as the percentage change of MOEs based on the No Metering condition.

Also, since Total Travel Time (TTT), the time accumulated by all vehicles travelling in the traffic network, is a very good indicator of the traffic system's overall performance, it will be used to evaluate the performance of the proposed ramp metering algorithms. The percentage change of TTT based the No Metering condition will be used to illustrate the comparison of the performance of FLC and genetic fuzzy approaches.

5.4.1 The simulation results and analysis of traffic demand data - Table 5.2

Table 5.6 ~ Table 5.11 shows the results of the traffic demand data of Table 5.2.

Table 5.6 General measures of Effectiveness at traffic demand (3000vehs/h~1600vehs/h)

1	Measures of Effectiveness	No Metering	Fuzzy Metering	Genetic Fuzzy Metering	% Change		
					Fuzzy VS. No Metering	GA VS. No Metering	GA VS. Fuzzy
Downstream MOEs	Total Travel Time (seconds per km)	1185.94	720.63	820.93	-39.24%	-30.78%	13.92%
	Average Delay (seconds per vehicle)	10.26	4.39	5.59	-57.21%	-45.52%	27.33%
	Average Flow Rate (vehicles per hour)	4002.25	3605.20	3706.00	-9.92%	-7.40%	2.80%
	Average Speed (kms per hour)	54.14	68.70	65.97	26.89%	21.85%	-3.97%
Ramp MOEs	Total Ramp Travel Time (seconds per km)	4299.25	4604.50	4512.98	7.10%	4.97%	-1.99%
	Average Ramp Delay (seconds per vehicle)	203.15	427.17	369.76	110.27%	82.01%	-13.44%
	Average Flow Rate (vehicles per hour)	1032.75	620.75	706.00	-39.89%	-31.64%	13.73%
System MOEs	Total Travel Time (hours)	**166.00**	**171.40**	**169.60**	**3.25%**	**2.17%**	**-1.05%**
	Average Delay Time (seconds per vehicle per km)	**126.54**	**132.04**	**130.17**	**4.35%**	**2.87%**	**-1.42%**

Table 5-7 General measures of Effectiveness at traffic demand (3200vehs/h~1600vehs/h)

2	Measures of Effectiveness	No Metering	Fuzzy Metering	Genetic Fuzzy Metering	% Change		
					Fuzzy VS. No Metering	GA VS. No Metering	GA VS. Fuzzy
Downstream MOEs	Total Travel Time (seconds per km)	1212.72	778.91	775.68	-35.77%	-36.04%	-0.41%
	Average Delay (seconds per vehicle)	10.20	4.71	4.66	-53.82%	-54.31%	-1.06%
	Average Flow Rate (vehicles per hour)	4108.50	3802.50	3787.25	-7.45%	-7.82%	-0.40%
	Average Speed (kms per hour)	54.41	67.00	68.00	23.14%	24.98%	1.49%
Ramp MOEs	Total Ramp Travel Time (seconds per km)	4631.69	4517.49	4362.43	-2.47%	-5.81%	-3.43%
	Average Ramp Delay (seconds per vehicle)	253.04	435.88	459.49	72.26%	81.59%	5.42%
	Average Flow Rate (vehicles per hour)	973.00	613.50	602.25	-36.95%	-38.10%	-1.83%
System MOEs	Total Travel Time (hours)	**171.06**	**167.97**	**167.60**	**-1.81%**	**-2.02%**	**-0.22%**
	Average Delay Time (seconds per vehicle per km)	**131.63**	**128.67**	**128.30**	**-2.25%**	**-2.53%**	**-0.29%**

Table 5-8 General measures of Effectiveness at traffic demand (3400vehs/h~1600vehs/h)

3	Measures of Effectiveness	No Metering	Fuzzy Metering	Genetic Fuzzy Metering	% Change		
					Fuzzy VS. No Metering	GA VS. No Metering	GA VS. Fuzzy
Downstream MOEs	Total Travel Time (seconds per km)	1242.58	903.27	805.58	-27.31%	-35.17%	-10.82%
	Average Delay (seconds per vehicle)	9.96	5.86	4.66	-41.16%	-53.21%	-20.48%
	Average Flow Rate (vehicles per hour)	4268.25	4031.75	3933.25	-5.54%	-7.85%	-2.44%
	Average Speed (kms per hour)	54.68	62.87	67.53	14.98%	23.50%	7.41%
Ramp MOEs	Total Ramp Travel Time (seconds per km)	4405.40	4570.63	4563.03	3.75%	3.58%	-0.17%
	Average Ramp Delay (seconds per vehicle)	264.45	435.72	491.16	64.76%	85.73%	12.72%
	Average Flow Rate (vehicles per hour)	939.75	606.50	567.07	-35.46%	-39.66%	-6.50%
System MOEs	Total Travel Time (hours)	**178.61**	**169.45**	**165.99**	**-5.13%**	**-7.07%**	**-2.04%**
	Average Delay Time (seconds per vehicle per km)	**139.20**	**130.07**	**126.63**	**-6.56%**	**-9.03%**	**-2.64%**

Table 5.9 General measures of Effectiveness at traffic demand (3600vehs/h~1600vehs/h)

4	Measures of Effectiveness	No Metering	Fuzzy Metering	Genetic Fuzzy Metering	% Change		
					Fuzzy VS. No Metering	GA VS. No Metering	GA VS. Fuzzy
Downstream MOEs	Total Travel Time (seconds per km)	1278.54	954.02	855.66	-25.38%	-33.08%	-10.31%
	Average Delay (seconds per vehicle)	9.65	6.15	4.81	-36.27%	-50.16%	-21.79%
	Average Flow Rate (vehicles per hour)	4475.50	4194.75	4155.00	-6.27%	-7.16%	-0.95%
	Average Speed (kms per hour)	55.20	61.45	66.20	11.32%	19.93%	7.73%
Ramp MOEs	Total Ramp Travel Time (seconds per km)	4449.47	4608.15	4450.64	3.57%	0.03%	-3.42%
	Average Ramp Delay (seconds per vehicle)	267.42	442.11	503.92	65.32%	88.44%	13.98%
	Average Flow Rate (vehicles per hour)	907.75	603.25	539.75	-33.54%	-40.54%	-10.53%
System MOEs	Total Travel Time (hours)	**190.07**	**169.29**	**161.60**	**-10.93%**	**-14.98%**	**-4.54%**
	Average Delay Time (seconds per vehicle per km)	**150.72**	**130.00**	**122.36**	**-13.75%**	**-18.82%**	**-5.88%**

Table 5.10 General measures of Effectiveness at traffic demand (3800vehs/h~1600vehs/h)

5	Measures of Effectiveness	No Metering	Fuzzy Metering	Genetic Fuzzy Metering	% Change		
					Fuzzy VS. No Metering	GA VS. No Metering	GA VS. Fuzzy
Downstream MOEs	Total Travel Time (seconds per km)	1283.42	1041.89	842.24	-18.82%	-34.38%	-19.16%
	Average Delay (seconds per vehicle)	9.48	6.95	4.32	-26.69%	-54.43%	-37.84%
	Average Flow Rate (vehicles per hour)	4535.00	4320.50	4271.75	-4.73%	-5.80%	-1.13%
	Average Speed (kms per hour)	55.67	58.98	66.94	5.95%	20.24%	13.50%
Ramp MOEs	Total Ramp Travel Time (seconds per km)	4453.85	4637.15	4614.77	4.12%	3.61%	-0.48%
	Average Ramp Delay (seconds per vehicle)	282.35	445.56	572.87	57.80%	102.89%	28.57%
	Average Flow Rate (vehicles per hour)	897.25	602.50	501.25	-32.85%	-44.13%	-16.80%
System MOEs	Total Travel Time (hours)	**191.53**	**180.11**	**161.54**	**-5.96%**	**-15.66%**	**-10.31%**
	Average Delay Time (seconds per vehicle per km)	**152.20**	**140.80**	**121.86**	**-7.49%**	**-19.93%**	**-13.45%**

Table 5.11 General measures of Effectiveness at traffic demand (4000vehs/h~1600vehs/h)

6	Measures of Effectiveness	No Metering	Fuzzy Metering	Genetic Fuzzy Metering	% Change		
					Fuzzy VS. No Metering	GA VS. No Metering	GA VS. Fuzzy
Downstream MOEs	Total Travel Time (seconds per km)	1285.30	1112.00	870.21	-13.48%	-32.30%	-21.74%
	Average Delay (seconds per vehicle)	9.48	7.40	4.38	-21.94%	-53.80%	-40.81%
	Average Flow Rate (vehicles per hour)	4540.25	4478.25	4402.50	-1.37%	-3.03%	-1.69%
	Average Speed (kms per hour)	55.64	57.83	66.52	3.94%	19.55%	15.03%
Ramp MOEs	Total Ramp Travel Time (seconds per km)	4463.06	4600.01	4202.91	3.07%	-5.83%	-8.63%
	Average Ramp Delay (seconds per vehicle)	284.06	442.69	536.25	55.84%	88.78%	21.13%
	Average Flow Rate (vehicles per hour)	890.25	601.50	493.50	-32.43%	-44.57%	-17.96%
System MOEs	Total Travel Time (hours)	**193.31**	**186.16**	**151.35**	**-3.70%**	**-21.71%**	**-18.70%**
	Average Delay Time (seconds per vehicle per km)	**153.92**	**146.87**	**112.29**	**-4.58%**	**-27.05%**	**-23.54%**

Analyzing the simulation results of the traffic demand data - Table 5.2

Figure 5.9 shows the comparison of the change of TTT for FLC and genetic fuzzy ramp metering when the ramp demand is 1600 vehicles/hour.

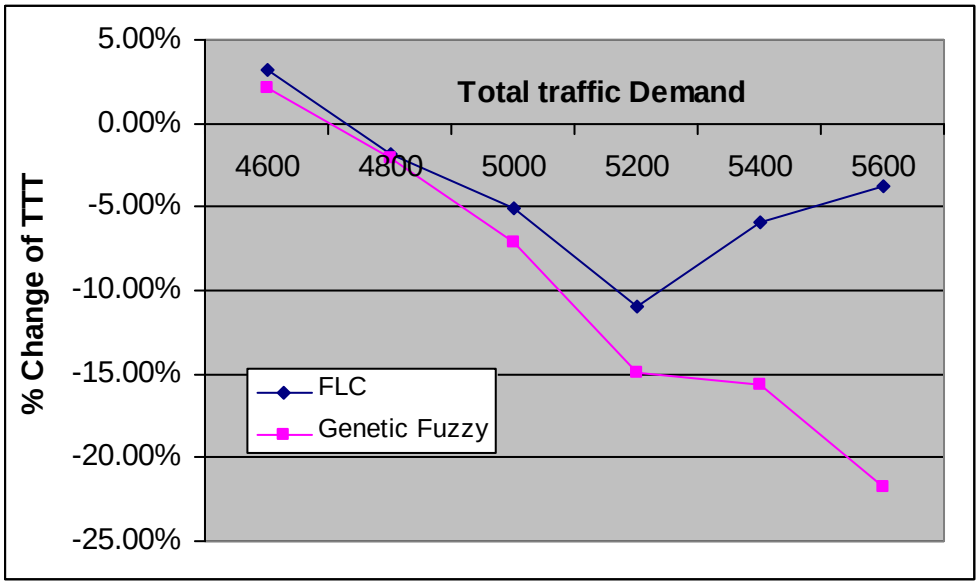


Figure 5.9 the percentage change of TTT when ramp demand is 1600 vehs/h

As Figure 5.9 shows, when total traffic demand is less than about 4700vehicles/hour, both FLC and genetic ramp metering fail to reduce the TTT, which means both ramp metering approaches do not work properly and even cause the extra delay for the traffic condition. And when total traffic demand is from 4600vehicles/hour to 4800 vehicles/hours, the performance of FLC and genetic fuzzy ramp metering is very close. When the total traffic demand reaches or nearly reaches the road capacity (4800 vehicles/h to 5000vehicles/hour), both FLC and genetic fuzzy ramp metering perform very well. FLC reduces TTT from -1.81% to -5.13% and genetic fuzzy have more significant reduction on TTT, which is from -2.02% to -7.07%. The situation keeps going well until total demand reaches 5200vehicles/h, where FLC reduces TTT to -10.93% and genetic fuzzy reduces TTT to -14.98%. After that, the performance of FLC turns to be worse and the corresponding TTT increases from -10.93% to -3.70% while the genetic fuzzy keeps working well and the relative TTT drops from -14.98% to -21.71%. The reason for that is because the objective function of genetic fuzzy ramp metering effectively keeps maintaining the flow density below the predefined density, while FLC ramp metering without the density restriction finally fails to prevent the formation of

congestion when total traffic demand goes too high. The change of traffic flow densities of the Aimsun traffic network shows the evidence why FLC and genetic fuzzy ramp metering performs differently at high traffic demand, which is shown in Figure 5.10 to Figure 5.12 (where NC means no control and GA-FLC means genetic fuzzy ramp metering).

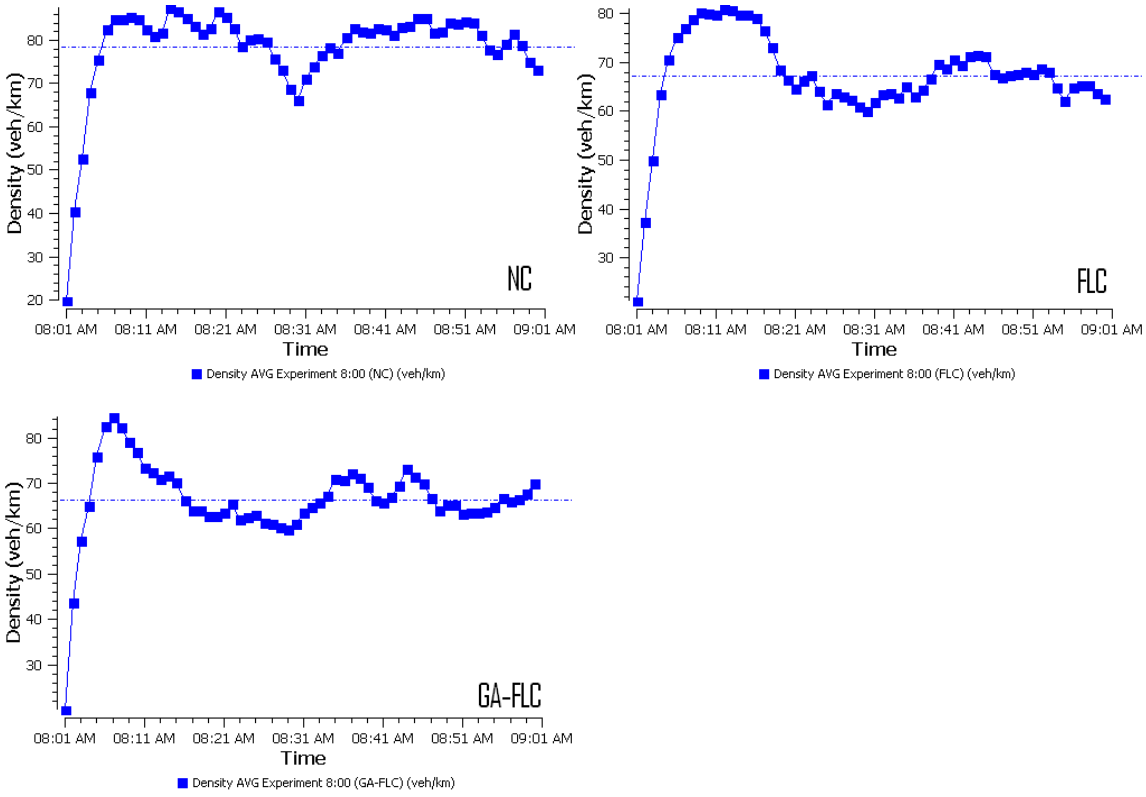


Figure 5.10 the change of average flow density when total demand is 5200 vehs/h

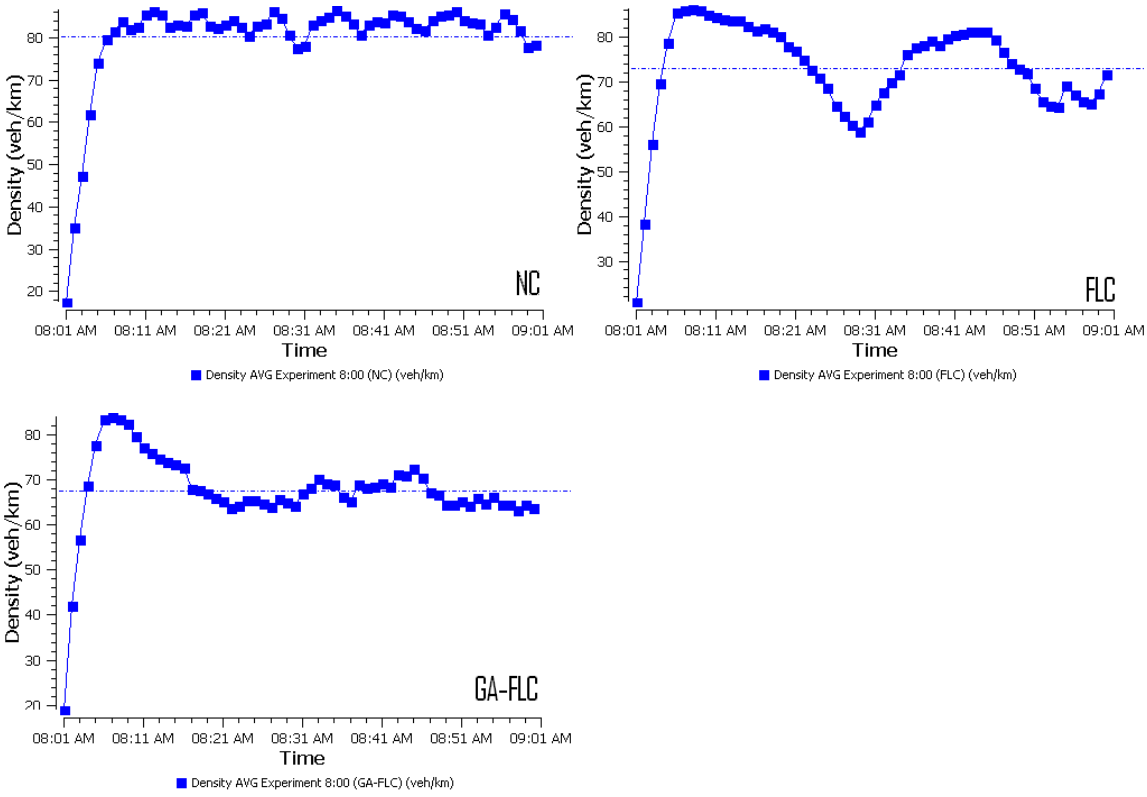


Figure 5.11 the change of average flow density when total demand is 5400 vehs/h

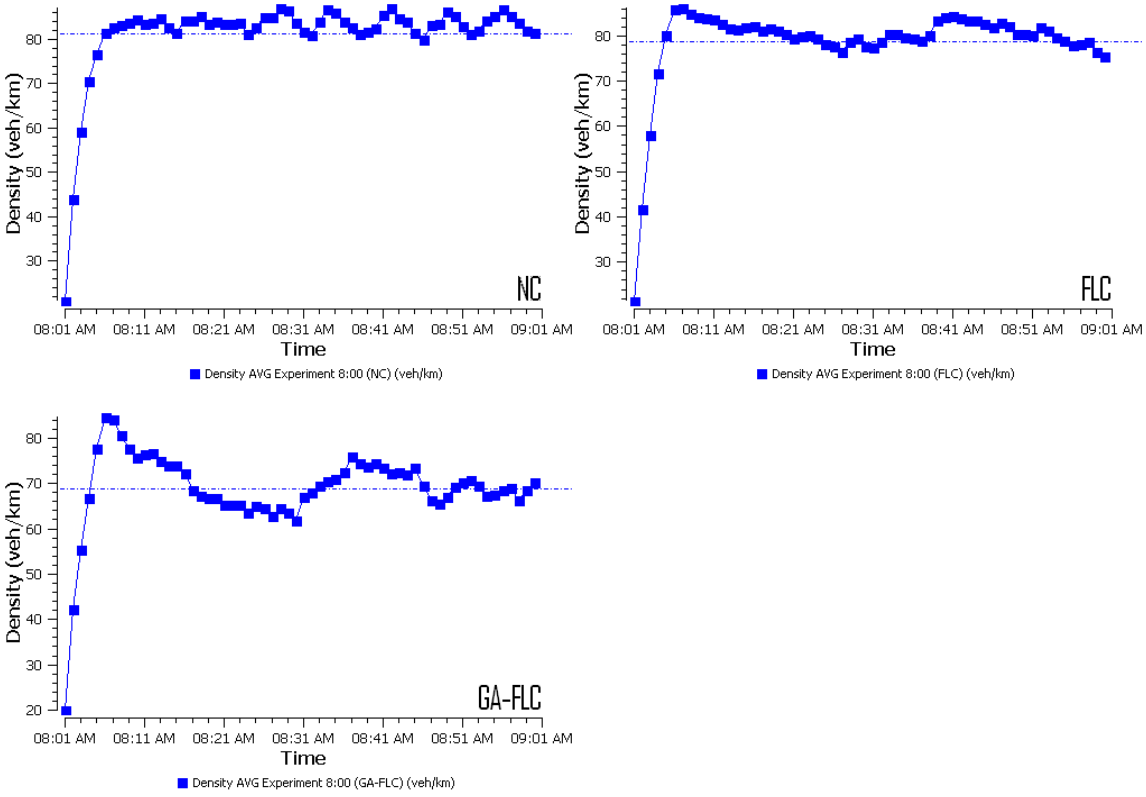


Figure 5.12 the change of average flow density when total demand is 5600 vehs/h

In Figure 5.10, both FLC and genetic fuzzy ramp metering effectively stabilize the flow density at about 70vehicles/km when the average flow density under NC situation almost reaches 80vehicles/km, so both the performance of FLC and genetic fuzzy metering are very well, which could be observed by the percentage changes of TTT in Figure 5.9, -10.93%(FLC) and -14.98%(genetic fuzzy).

In Figure 5.11, when the total traffic demand increases to 5400vehicles/hour, the oscillation of flow density under FLC is very obvious which means FLC is less effective to stabilize the flow density than genetic fuzzy ramp metering. Back to the percentage change of TTT under FLC ramp metering in Figure 5.9, there is a significant increase on TTT from -10.93% to -5.96% when total traffic demand increases from 5200vehicles/hour to 5400vehicles/hour, while the change of TTT under genetic fuzzy ramp metering is decreased from -14.98 to -15.66%.

In Figure 5.12, when the total traffic demand increases to 5600vehicles/hour, the change of the flow density under FLC shows no obvious difference with the change of traffic flow density under NC situation, which means FLC almost has less effect on controlling traffic condition and possible fails to prevent the formation of traffic congestion. And Figure 5.9 shows that the percentage change of TTT under FLC drops to -3%. Meanwhile, genetic fuzzy ramp metering still effectively maintains and stabilizes the flow density at about 70vehicles/km and Figure 5.9 shows that the percentage change of TTT under genetic fuzzy ramp metering increases to -23%.

5.4.2 The simulation results and analysis with traffic demand data - Table 5.3

Table 5.12 ~ Table 5.17 shows the results of the traffic demand data of Table 5.3.

Table 5.12 General measures of Effectiveness at traffic demand (3200~1400)

7	Measures of Effectiveness	No Metering	Fuzzy Metering	Genetic Fuzzy Metering	% Change		
					Fuzzy VS. No Metering	GA VS. No Metering	GA VS. Fuzzy
Downstream MOEs	Total Travel Time (seconds per km)	1214.69	822.99	756.11	-32.25%	-37.75%	-8.13%
	Average Delay (seconds per vehicle)	10.02	5.35	4.39	-46.61%	-56.19%	-17.94%
	Average Flow Rate (vehicles per hour)	4157.50	3800.75	3773.50	-8.58%	-9.24%	-0.72%
	Average Speed (kms per hour)	54.56	65.52	68.75	20.09%	26.01%	4.93%
Ramp MOEs	Total Ramp Travel Time (seconds per km)	4286.35	4577.50	4563.42	6.79%	6.46%	-0.31%
	Average Ramp Delay (seconds per vehicle)	244.11	432.61	470.07	77.22%	92.56%	8.66%
	Average Flow Rate (vehicles per hour)	986.75	611.50	586.75	-38.03%	-40.54%	-4.05%
System MOEs	Total Travel Time (hours)	**170.83**	**169.89**	**169.11**	**-0.55%**	**-1.01%**	**-0.46%**
	Average Delay Time (seconds per vehicle per km)	**131.39**	**130.57**	**129.80**	**-0.62%**	**-1.21%**	**-0.59%**

Table 5.13 General measures of Effectiveness at traffic demand (3400~1400)

8	Measures of Effectiveness	No Metering	Fuzzy Metering	Genetic Fuzzy Metering	% Change		
					Fuzzy VS. No Metering	GA VS. No Metering	GA VS. Fuzzy
Downstream MOEs	Total Travel Time (seconds per km)	1240.81	898.26	793.02	-27.61%	-36.09%	-11.72%
	Average Delay (seconds per vehicle)	9.82	5.89	4.59	-40.02%	-53.26%	-22.07%
	Average Flow Rate (vehicles per hour)	4295.5	4003.5	3903.5	-6.80%	-9.13%	-2.50%
	Average Speed (kms per hour)	54.71	62.87	67.8	14.92%	23.93%	7.84%
Ramp MOEs	Total Ramp Travel Time (seconds per km)	4368.89	4545.18	4516.57	4.04%	3.38%	-0.63%
	Average Ramp Delay (seconds per vehicle)	259.28	432.65	512.2	66.87%	97.55%	18.39%
	Average Flow Rate (vehicles per hour)	947.25	607.75	540.75	-35.84%	-42.91%	-11.02%
System MOEs	Total Travel Time (hours)	**177.33**	**168.55**	**164.57**	**-4.95%**	**-7.20%**	**-2.36%**
	Average Delay Time (seconds per vehicle per km)	**137.92**	**129.20**	**125.27**	**-6.32%**	**-9.17%**	**-3.04%**

Table 5.14 General measures of Effectiveness at traffic demand (3600~1400)

9	Measures of Effectiveness	No Metering	Fuzzy Metering	Genetic Fuzzy Metering	% Change		
					Fuzzy VS. No Metering	GA VS. No Metering	GA VS. Fuzzy
Downstream MOEs	Total Travel Time (seconds per km)	1277.55	979.52	815.57	-23.33%	-36.16%	-16.74%
	Average Delay (seconds per vehicle)	9.7	6.52	4.4	-32.78%	-54.64%	-32.52%
	Average Flow Rate (vehicles per hour)	4457.00	4187.75	4096.25	-6.04%	-8.09%	-2.18%
	Average Speed (kms per hour)	55.12	60.60	67.73	9.94%	22.88%	11.77%
Ramp MOEs	Total Ramp Travel Time (seconds per km)	4368.88	4552.18	4633.88	4.20%	6.07%	1.79%
	Average Ramp Delay (seconds per vehicle)	272.93	436.81	536.18	60.04%	96.45%	22.75%
	Average Flow Rate (vehicles per hour)	908.00	603.25	532.00	-33.56%	-41.41%	-11.81%
System MOEs	Total Travel Time (hours)	**186.04**	**172.52**	**164.04**	**-7.27%**	**-11.83%**	**-4.92%**
	Average Delay Time (seconds per vehicle per km)	**146.67**	**133.24**	**124.83**	**-9.16%**	**-14.89%**	**-6.31%**

Table 5.15 General measures of Effectiveness at traffic demand (3800~1400)

10	Measures of Effectiveness	No Metering	Fuzzy Metering	Genetic Fuzzy Metering	% Change		
					Fuzzy VS. No Metering	GA VS. No Metering	GA VS. Fuzzy
Downstream MOEs	Total Travel Time (seconds per km)	1287.11	1064.36	856.90	-17.31%	-33.42%	-19.49%
	Average Delay (seconds per vehicle)	9.55	7.08	4.53	-25.86%	-52.57%	-36.02%
	Average Flow Rate (vehicles per hour)	4525.75	4384.75	4269.00	-3.12%	-5.67%	-2.64%
	Average Speed (kms per hour)	55.59	58.69	66.39	5.58%	19.43%	13.12%
Ramp MOEs	Total Ramp Travel Time (seconds per km)	4410.37	4553.01	4637.30	3.23%	5.15%	1.85%
	Average Ramp Delay (seconds per vehicle)	280.84	438.00	566.99	55.96%	101.89%	29.45%
	Average Flow Rate (vehicles per hour)	893.00	602.25	508.5	-32.56%	-43.06%	-15.57%
System MOEs	Total Travel Time (hours)	**191.22**	**179.81**	**163.61**	**-5.97%**	**-14.44%**	**-9.01%**
	Average Delay Time (seconds per vehicle per km)	**151.88**	**140.57**	**124.34**	**-7.45%**	**-18.13%**	**-11.55%**

Table 5.16 General measures of Effectiveness at traffic demand (4000~1400)

11	Measures of Effectiveness	No Metering	Fuzzy Metering	Genetic Fuzzy Metering	% Change		
					Fuzzy VS. No Metering	GA VS. No Metering	GA VS. Fuzzy
Downstream MOEs	Total Travel Time (seconds per km)	1291.33	1118.9	875.92	-13.35%	-32.17%	-21.72%
	Average Delay (seconds per vehicle)	9.54	7.39	4.31	-22.54%	-54.82%	-41.68%
	Average Flow Rate (vehicles per hour)	4542.25	4514.25	4458.25	-0.62%	-1.85%	-1.24%
	Average Speed (kms per hour)	55.67	57.91	66.40	4.02%	19.27%	14.66%
Ramp MOEs	Total Ramp Travel Time (seconds per km)	4406.09	4558.77	4495.07	3.47%	2.02%	-1.40%
	Average Ramp Delay (seconds per vehicle)	279.3	438.57	568.77	57.02%	103.64%	29.69%
	Average Flow Rate (vehicles per hour)	891.25	602.00	494.75	-32.45%	-44.49%	-17.82%
System MOEs	Total Travel Time (hours)	**192.29**	**186.13**	**157.34**	**-3.20%**	**-18.18%**	**-15.47%**
	Average Delay Time (seconds per vehicle per km)	**152.92**	**146.92**	**118.10**	**-3.92%**	**-22.77%**	**-19.62%**

Table 5.17 General measures of Effectiveness at traffic demand (4200~1400)

12	Measures of Effectiveness	No Metering	Fuzzy Metering	Genetic Fuzzy Metering	% Change		
					Fuzzy VS. No Metering	GA VS. No Metering	GA VS. Fuzzy
Downstream MOEs	Total Travel Time (seconds per km)	1282.82	1119.99	898.07	-12.69%	-29.99%	-19.81%
	Average Delay (seconds per vehicle)	9.43	7.31	4.13	-22.48%	-56.20%	-43.50%
	Average Flow Rate (vehicles per hour)	4543.25	4541.25	4656.00	-0.04%	2.48%	2.53%
	Average Speed (kms per hour)	55.81	57.8	66.23	3.57%	18.67%	14.58%
Ramp MOEs	Total Ramp Travel Time (seconds per km)	4407.08	4530.52	4327.65	2.80%	-1.80%	-4.48%
	Average Ramp Delay (seconds per vehicle)	279.83	435.71	577.5	55.71%	106.38%	32.54%
	Average Flow Rate (vehicles per hour)	891.25	602.00	475.00	-32.45%	-46.70%	-21.10%
System MOEs	Total Travel Time (hours)	**192.05**	**184.90**	**150.11**	**-3.72%**	**-21.84%**	**-18.82%**
	Average Delay Time (seconds per vehicle per km)	**152.71**	**145.61**	**110.85**	**-4.65%**	**-27.41%**	**-23.87%**

Analyzing the simulation results of the traffic demand data - Table 5.3

Figure 5.13 shows the comparison of the change of TTT for FLC and genetic fuzzy ramp metering when the ramp demand is 1400 vehicles/hour.

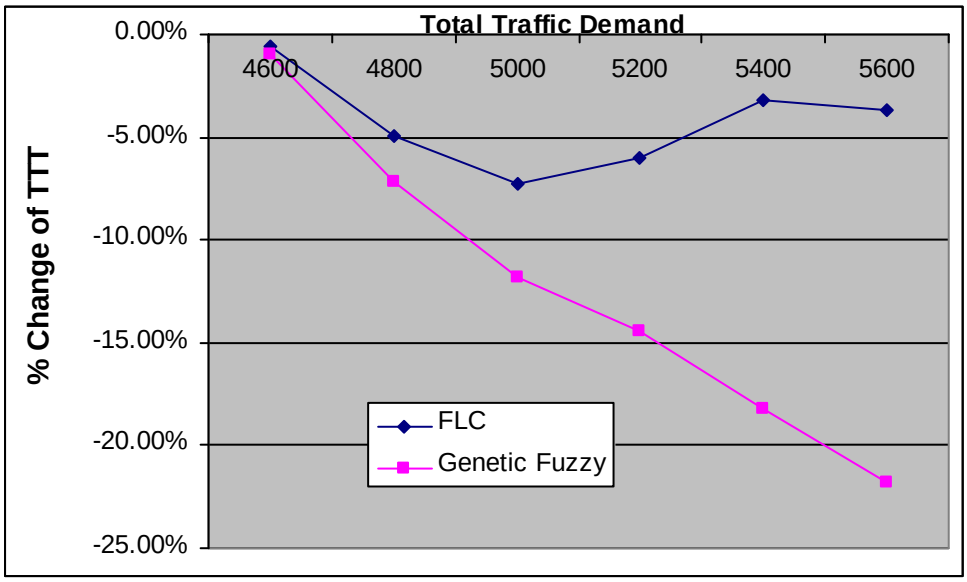


Figure 5.13 the percentage change of TTT when ramp demand is 1400 vehs/h

As Figure 5.13 shows, both FLC and genetic ramp metering start working properly when total traffic demand is more than about 4600vehicles. And when total traffic demand is from 4800vehicles/hour to road capacity (5000 vehicles/hours), both FLC and genetic fuzzy ramp metering perform very well. FLC reduces TTT from -4.95% to -7.27% and genetic fuzzy have more significant reduction on TTT, which is from -7.20% to -11.83%. The situation keeps going well until total demand reaches 5200vehicles/h, where FLC reduces TTT to -5.97% and genetic fuzzy reduces TTT to -14.44%. After that, the performance of FLC obviously turns to be worse and the corresponding TTT significantly increases from -5.97% to -3.20% while the genetic fuzzy keeps working well and the relative TTT drops from -14.44% to -18.18%. The reason for that is because the objective function of genetic fuzzy ramp metering effectively keeps maintaining the flow density below the predefined density. Again, the change of traffic flow densities of the Aimsun traffic network is used to show why FLC and genetic fuzzy ramp metering performs differently at high traffic demand, which is shown in Figure 5.14 to Figure 5.16 (where NC means no control and GA-FLC means genetic fuzzy ramp metering).

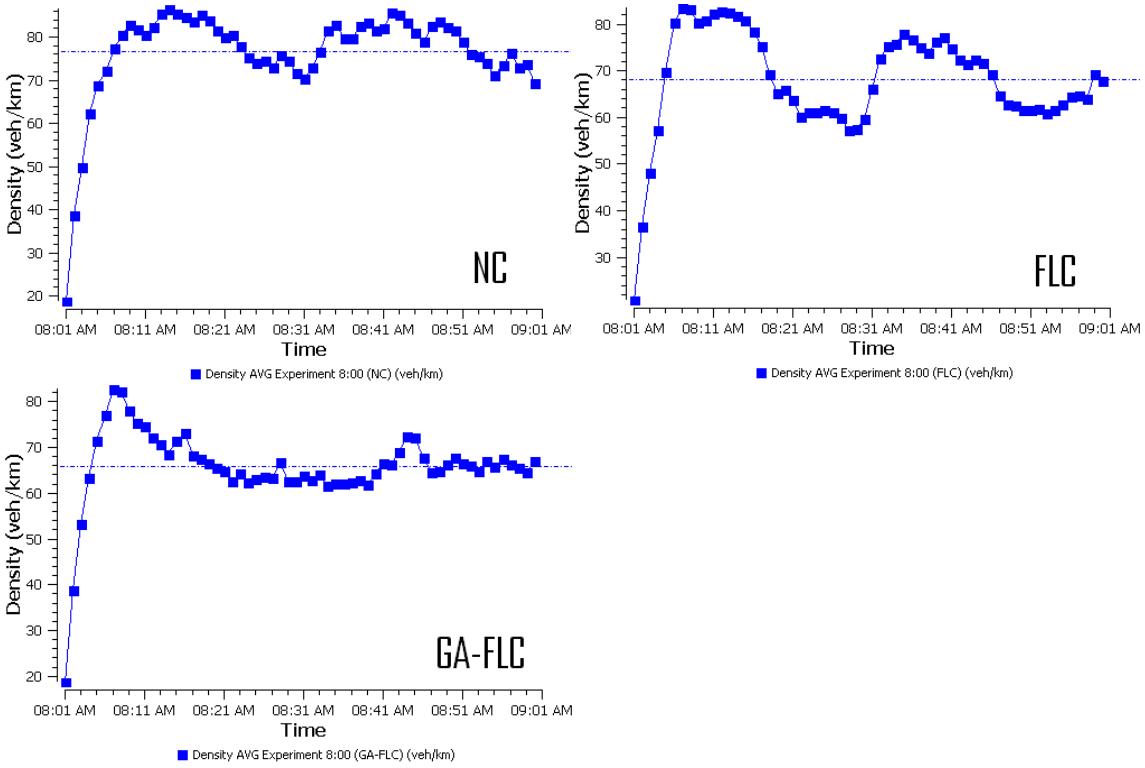


Figure 5.14 the change of average flow density when total demand is 5000 vehs/h

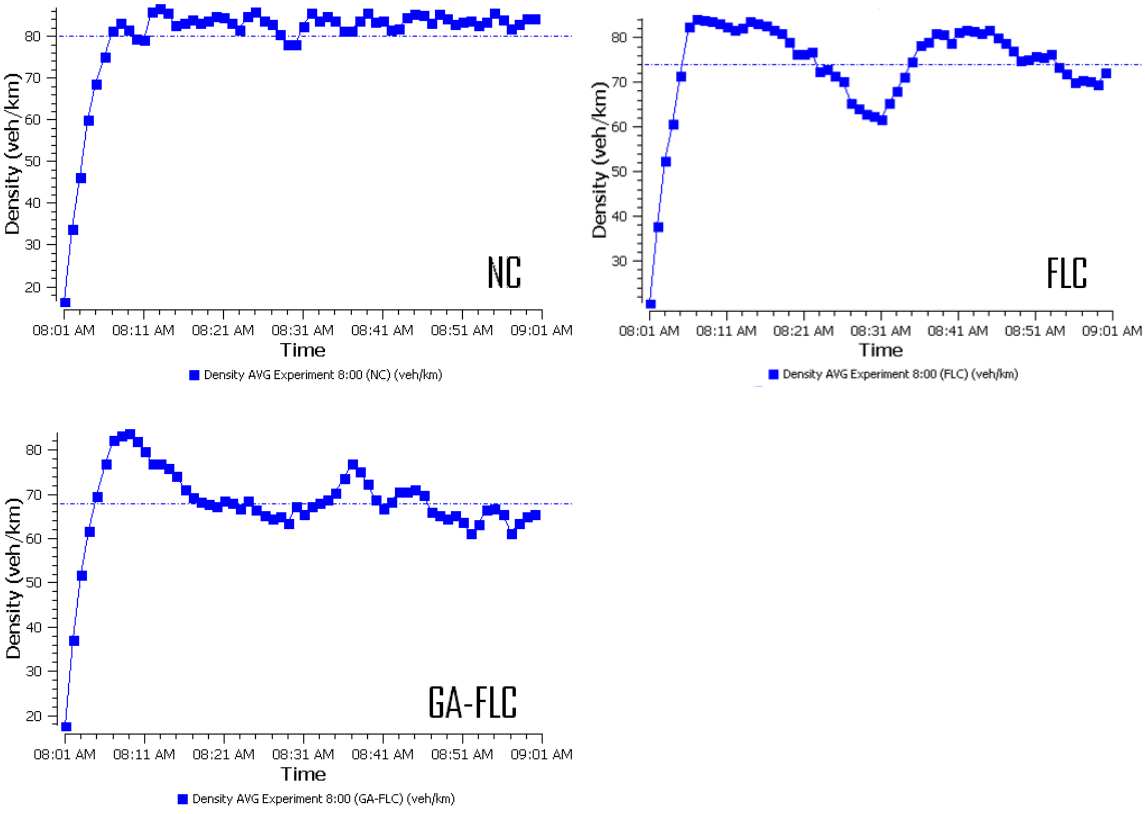


Figure 5.15 the change of average flow density when total demand is 5200 vehs/h

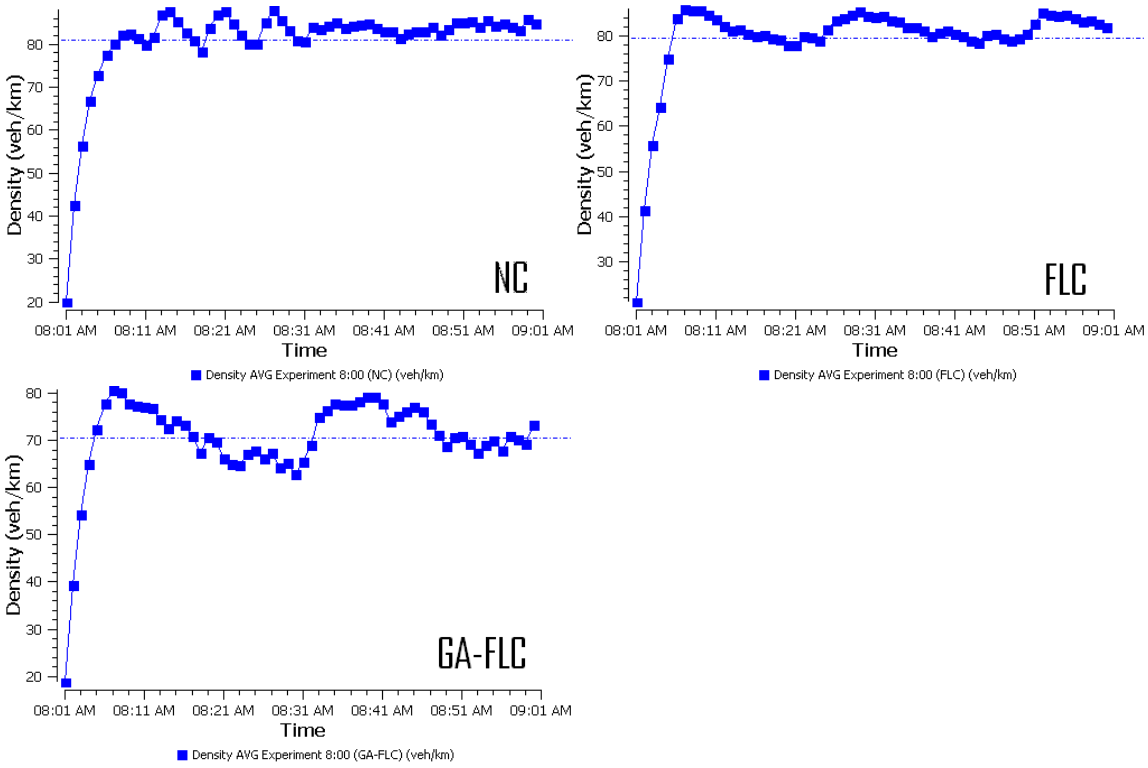


Figure 5.16 the change of average flow density when total demand is 5400 vehs/h

In Figure 5.14 and Figure 5.15, genetic fuzzy ramp metering effectively stabilize the flow density at about 65vehicles/km when the average flow density under NC situation nearly reaches 80vehicles/km while the change of the flow density under FLC is unstable and the average density is around 70vehicle/hour. Then back to Figure 5.13, we can see the TTT under FLC increase from -7.27% to -5.97% and the TTT under genetic fuzzy decrease from -11.83% to -14.44% when total traffic demand increases from 5000vehicles/hour to 5200vehicles/hour. Apparently, genetic fuzzy ramp metering with the stable flow density performs better than FLC although FLC also works well.

In Figure 5.16, when the total traffic demand increases to 5400vehicles/hour, the change of the flow density under FLC shows no obvious difference with the change of traffic flow density under NC situation, which means FLC almost has less effect on controlling traffic condition and possibly fails to prevent the formation of traffic congestion. And Figure 5.13 shows that the percentage change of TTT under FLC drops to -3.20%. Meanwhile, genetic fuzzy ramp metering still effectively maintains and stabilizes the flow density at about 70vehicles/km and Figure 5.13 shows that the percentage change of TTT under genetic fuzzy ramp metering increases to -18.18%.

5.4.3 The simulation results and analysis with traffic demand data - Table 5.4

Table 5.18 ~ Table 5.23 shows the results of the traffic demand data of Table 5.4.

Table 5.18 General measures of Effectiveness at traffic demand (3200~1200)

13	Measures of Effectiveness	No Metering	Fuzzy Metering	Genetic Fuzzy Metering	% Change		
					Fuzzy VS. No Metering	GA VS. No Metering	GA VS. Fuzzy
Downstream MOEs	Total Travel Time (seconds per km)	1207.78	803.77	780.36	-33.45%	-35.39%	-2.91%
	Average Delay (seconds per vehicle)	10.16	5.15	4.68	-49.31%	-53.94%	-9.13%
	Average Flow Rate (vehicles per hour)	4099.00	3781.50	3798.25	-7.75%	-7.34%	0.44%
	Average Speed (kms per hour)	54.59	66.08	67.70	21.05%	24.02%	2.45%
Ramp MOEs	Total Ramp Travel Time (seconds per km)	4191.24	4487.78	4587.8	7.08%	9.46%	2.23%
	Average Ramp Delay (seconds per vehicle)	240.99	420.58	464.76	74.52%	92.85%	10.50%
	Average Flow Rate (vehicles per hour)	971.25	614.75	599.25	-36.71%	-38.30%	-2.52%
System MOEs	Total Travel Time (hours)	**166.67**	**167.37**	**169.63**	**0.42%**	**1.78%**	**1.35%**
	Average Delay Time (seconds per vehicle per km)	**127.24**	**128.09**	**130.28**	**0.67%**	**2.39%**	**1.71%**

Table 5.19 General measures of Effectiveness at traffic demand (3400~1200)

14	Measures of Effectiveness	No Metering	Fuzzy Metering	Genetic Fuzzy Metering	% Change		
					Fuzzy VS. No Metering	GA VS. No Metering	GA VS. Fuzzy
Downstream MOEs	Total Travel Time (seconds per km)	1248.66	867.21	795.55	-30.55%	-36.29%	-8.26%
	Average Delay (seconds per vehicle)	9.96	5.46	4.51	-45.18%	-54.72%	-17.40%
	Average Flow Rate (vehicles per hour)	4288.5	3985.5	3946.75	-7.07%	-7.97%	-0.97%
	Average Speed (kms per hour)	54.64	64.11	67.82	17.33%	24.12%	5.79%
Ramp MOEs	Total Ramp Travel Time (seconds per km)	4200.48	4456.94	4594.77	6.11%	9.39%	3.09%
	Average Ramp Delay (seconds per vehicle)	251.72	422.54	492.63	67.86%	95.71%	16.59%
	Average Flow Rate (vehicles per hour)	938.75	608.75	565.00	-35.15%	-39.81%	-7.19%
System MOEs	Total Travel Time (hours)	**175.99**	**165.84**	**165.75**	**-5.77%**	**-5.82%**	**-0.05%**
	Average Delay Time (seconds per vehicle per km)	**136.61**	**126.53**	**126.44**	**-7.38%**	**-7.44%**	**-0.07%**

Table 5.20 General measures of Effectiveness at traffic demand (3600~1200)

15	Measures of Effectiveness	No Metering	Fuzzy Metering	Genetic Fuzzy Metering	% Change		
					Fuzzy VS. No Metering	GA VS. No Metering	GA VS. Fuzzy
Downstream MOEs	Total Travel Time (seconds per km)	1270.87	989.62	835.17	-22.13%	-34.28%	-15.61%
	Average Delay (seconds per vehicle)	9.62	6.55	4.75	-31.91%	-50.62%	-27.48%
	Average Flow Rate (vehicles per hour)	4451.5	4216.75	4073.25	-5.27%	-8.50%	-3.40%
	Average Speed (kms per hour)	55.21	60.39	66.72	9.38%	20.85%	10.48%
Ramp MOEs	Total Ramp Travel Time (seconds per km)	4281.46	4544.07	4677.74	6.13%	9.26%	2.94%
	Average Ramp Delay (seconds per vehicle)	265.41	434.81	537.59	63.83%	102.55%	23.64%
	Average Flow Rate (vehicles per hour)	911.75	604.25	530.00	-33.73%	-41.87%	-12.29%
System MOEs	Total Travel Time (hours)	**183.25**	**172.82**	**167.66**	**-5.69%**	**-8.51%**	**-2.99%**
	Average Delay Time (seconds per vehicle per km)	**143.85**	**133.57**	**128.40**	**-7.15%**	**-10.74%**	**-3.87%**

Table 5.21 General measures of Effectiveness at traffic demand (3800~1200)

16	Measures of Effectiveness	No Metering	Fuzzy Metering	Genetic Fuzzy Metering	% Change		
					Fuzzy VS. No Metering	GA VS. No Metering	GA VS. Fuzzy
Downstream MOEs	Total Travel Time (seconds per km)	1280.18	1060.81	848.93	-17.14%	-33.69%	-19.97%
	Average Delay (seconds per vehicle)	9.49	7.08	4.5	-25.40%	-52.58%	-36.44%
	Average Flow Rate (vehicles per hour)	4516.75	4363.5	4241.25	-3.39%	-6.10%	-2.80%
	Average Speed (kms per hour)	55.69	58.55	66.67	5.14%	19.72%	13.87%
Ramp MOEs	Total Ramp Travel Time (seconds per km)	4307.97	4492.07	4421.8	4.27%	2.64%	-1.56%
	Average Ramp Delay (seconds per vehicle)	277.36	431.75	545.61	55.66%	96.72%	26.37%
	Average Flow Rate (vehicles per hour)	881.5	601.75	503.5	-31.74%	-42.88%	-16.33%
System MOEs	Total Travel Time (hours)	**188.62**	**178.63**	**158.41**	**-5.30%**	**-16.02%**	**-11.32%**
	Average Delay Time (seconds per vehicle per km)	**149.26**	**139.34**	**119.15**	**-6.65%**	**-20.17%**	**-14.49%**

Table 5.22 General measures of Effectiveness at traffic demand (4000~1200)

17	Measures of Effectiveness	No Metering	Fuzzy Metering	Genetic Fuzzy Metering	% Change		
					Fuzzy VS. No Metering	GA VS. No Metering	GA VS. Fuzzy
Downstream MOEs	Total Travel Time (seconds per km)	1287.12	1108.58	861.39	-13.87%	-33.08%	-22.30%
	Average Delay (seconds per vehicle)	9.48	7.25	4.28	-23.52%	-54.85%	-40.97%
	Average Flow Rate (vehicles per hour)	4543.75	4508.25	4397.25	-0.78%	-3.22%	-2.46%
	Average Speed (kms per hour)	55.66	58	66.9	4.20%	20.19%	15.34%
Ramp MOEs	Total Ramp Travel Time (seconds per km)	4339.71	4456.42	4430.3	2.69%	2.09%	-0.59%
	Average Ramp Delay (seconds per vehicle)	276.99	428.42	565.08	54.67%	104.01%	31.90%
	Average Flow Rate (vehicles per hour)	883.25	601.5	487.75	-31.90%	-44.78%	-18.91%
System MOEs	Total Travel Time (hours)	**190.35**	**183.56**	**154.48**	**-3.57%**	**-18.84%**	**-15.84%**
	Average Delay Time (seconds per vehicle per km)	**150.99**	**144.26**	**115.22**	**-4.46%**	**-23.69%**	**-20.13%**

Table 5-23 General measures of Effectiveness at traffic demand (4200~1200)

18	Measures of Effectiveness	No Metering	Fuzzy Metering	Genetic Fuzzy Metering	% Change		
					Fuzzy VS. No Metering	GA VS. No Metering	GA VS. Fuzzy
Downstream MOEs	Total Travel Time (seconds per km)	1286.53	1111.96	910.67	-13.57%	-29.22%	-18.10%
	Average Delay (seconds per vehicle)	9.49	7.2	4.20	-24.13%	-55.74%	-41.67%
	Average Flow Rate (vehicles per hour)	4537.25	4541.25	4698.25	0.09%	3.55%	3.46%
	Average Speed (kms per hour)	55.78	58.09	65.69	4.14%	17.77%	13.08%
Ramp MOEs	Total Ramp Travel Time (seconds per km)	4339.14	4492.83	4200.68	3.54%	-3.19%	-6.50%
	Average Ramp Delay (seconds per vehicle)	279.26	432.04	558.01	54.71%	99.82%	29.16%
	Average Flow Rate (vehicles per hour)	875.75	601.50	477.25	-31.32%	-45.50%	-20.66%
System MOEs	Total Travel Time (hours)	**190.35**	**183.73**	**147.20**	**-3.48%**	**-22.67%**	**-19.88%**
	Average Delay Time (seconds per vehicle per km)	**150.95**	**144.41**	**107.94**	**-4.33%**	**-28.49%**	**-25.25%**

Analyzing the simulation results of the traffic demand data - Table 5.4

Figure 5.17 shows the comparison of the change of TTT for FLC and genetic fuzzy ramp metering when the ramp demand is 1200 vehicles/hour.

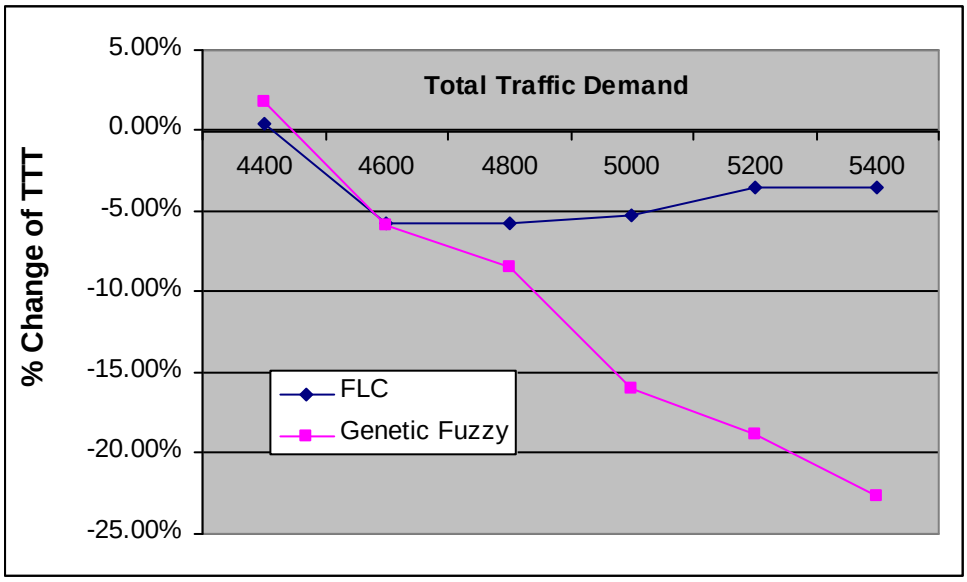


Figure 5.17 the percentage change of TTT when ramp demand is 1200 vehs/h

As Figure 5.17 shows, when total traffic demand is less than about 4400vehicles/hour, both FLC and genetic ramp metering fail to reduce the TTT, which means both ramp metering approaches do not work properly and even cause the extra delay for the traffic condition. And when total traffic demand is from 4400vehicles/hour to 4600 vehicles/hours, the performance of FLC and genetic fuzzy ramp metering is very close. When the total traffic demand reaches or nearly reaches the road capacity (4600 vehicles/h to 5000vehicles/hour), both FLC and genetic fuzzy ramp metering perform very well. FLC reduces TTT from -5.77% to -5.30% and genetic fuzzy have more significant reduction on TTT, which is from -5.82% to -16.02%. After that, the performance of FLC turns to be worse and the corresponding TTT increases from -5.30% to -3.57% while the genetic fuzzy keeps working well and the relative TTT drops from -16.02% to -18.84%. The change of traffic flow densities of the Aimsun traffic network shows why FLC and genetic fuzzy ramp metering performs differently at high traffic demand, which is shown in Figure 5.18 to Figure 5.20 (where NC means no control and GA-FLC means genetic fuzzy ramp metering).

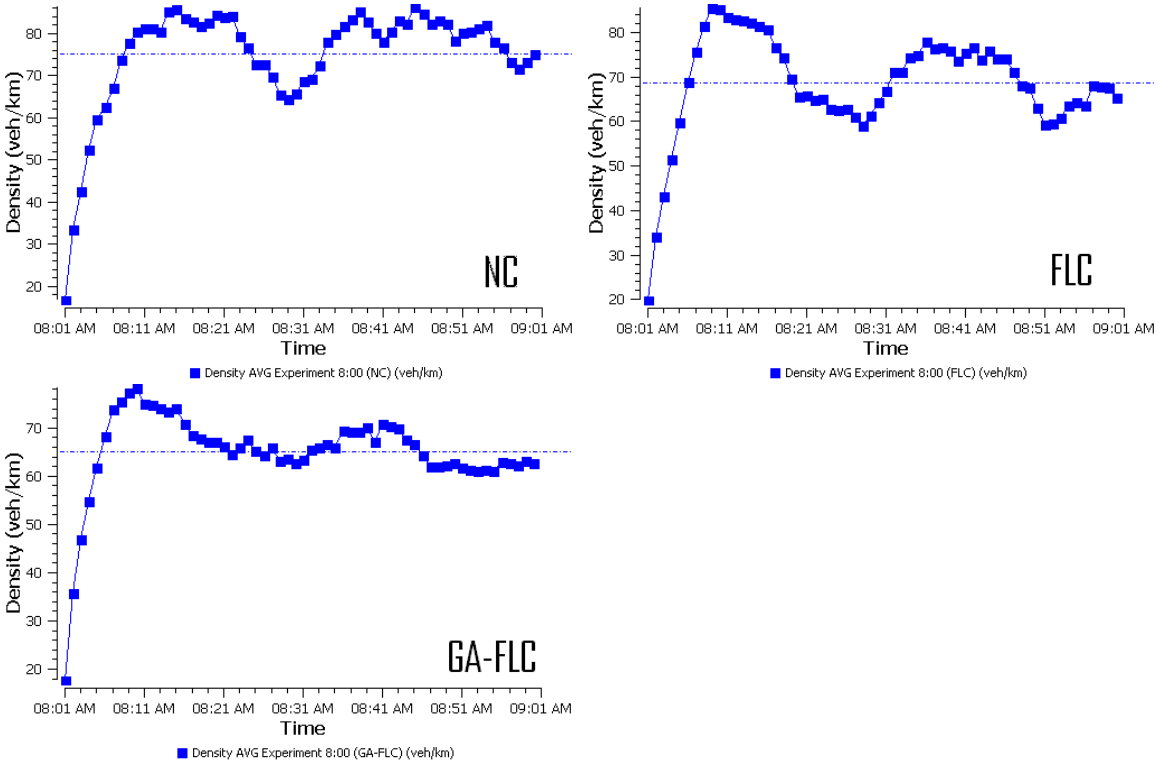


Figure 5.18 the change of average flow density when total demand is 4800 vehs/h

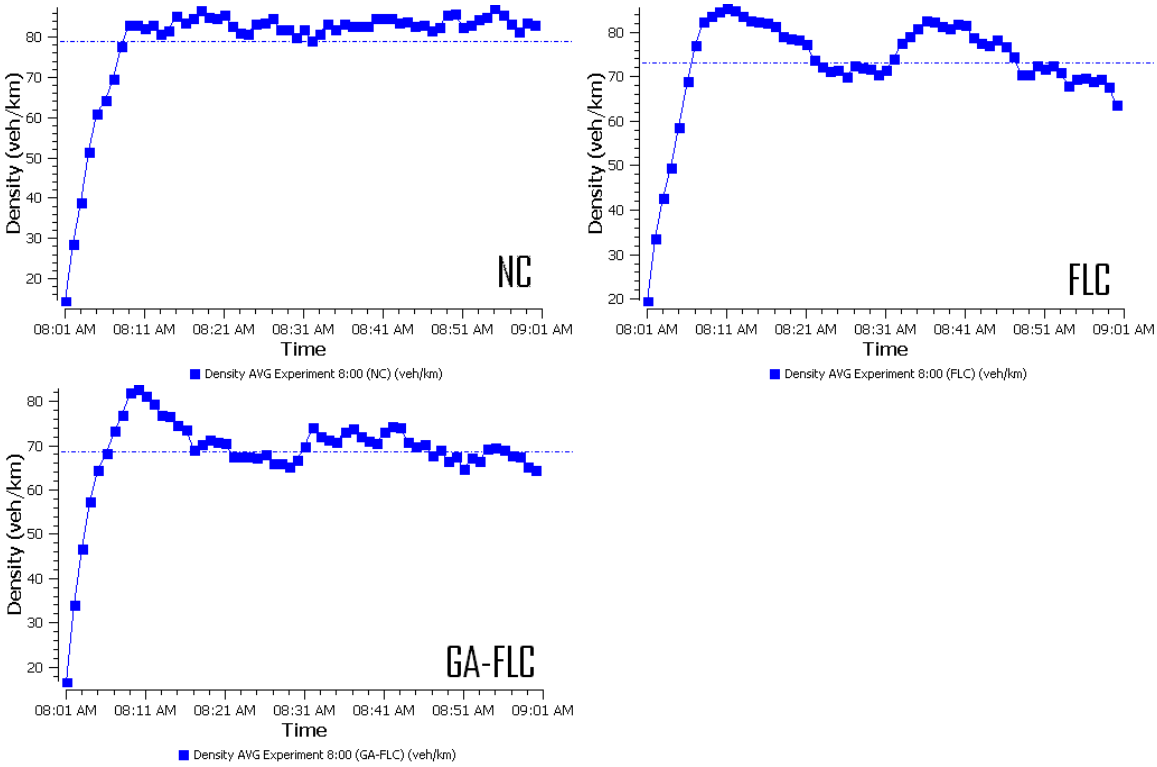


Figure 5.19 the change of average flow density when total demand is 5000 vehs/h

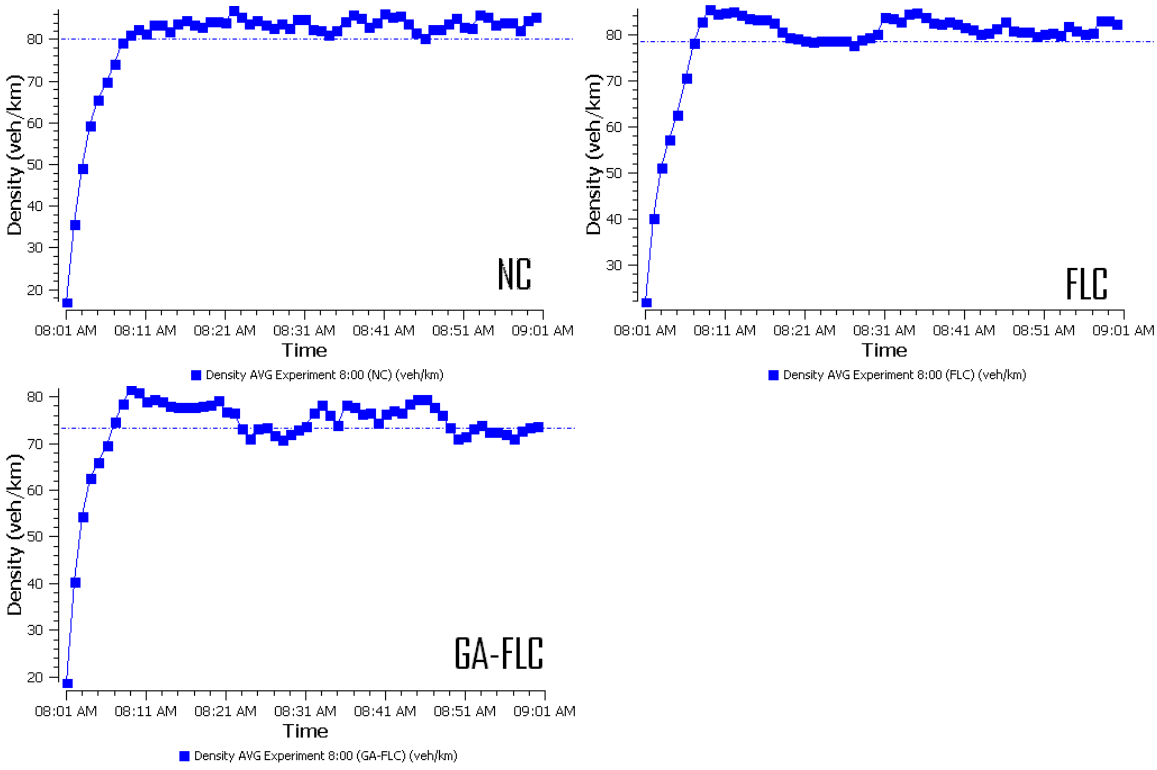


Figure 5.20 the change of average flow density when total demand is 5200 vehs/h

In Figure 5.18 and Figure 5.19, genetic fuzzy ramp metering effectively stabilize the flow density at about 66vehicles/km when the average flow density under NC situation nearly reaches 80vehicles/km, while the change of the flow density under FLC is unstable and the average flow density is around 70vehicles/hour. Then back to Figure 5.17, we can see the TTT under FLC slightly increases from -5.69% to -5.30% and the TTT under genetic fuzzy decrease from -8.51% to -16.02% when total traffic demand increases from 4800vehicles/hour to 5000vehicles/hour. Apparently, genetic fuzzy ramp metering with the stable flow density performs better than FLC although FLC also works well.

In Figure 5.20, when the total traffic demand increases to 5200vehicles/hour, the change of the flow density under FLC shows no obvious difference with the change of traffic flow density under NC situation, which means FLC almost has less effect on controlling traffic condition and possibly fails to prevent the formation of traffic congestion. And Figure 5.17 shows that the percentage change of TTT under FLC drops to -3.57%. Meanwhile, genetic fuzzy ramp metering still effectively maintains and stabilizes the flow density at about 70vehicles/km and Figure 5.17 shows that the percentage change of TTT under genetic fuzzy ramp metering increases to -18.84%.

5.4.4 The simulation results and analysis with traffic demand data - Table 5.5

Table 5.24 ~ Table 5.29 shows the results of the traffic demand data of Table 5.5.

Table 5.24 General measures of Effectiveness at traffic demand (3200~1000)

19	Measures of Effectiveness	No Metering	Fuzzy Metering	Genetic Fuzzy Metering	% Change		
					Fuzzy VS. No Metering	GA VS. No Metering	GA VS. Fuzzy
Downstream MOEs	Total Travel Time (seconds per km)	1127.07	825.83	775.99	-26.73%	-31.15%	-6.04%
	Average Delay (seconds per vehicle)	9.18	5.40	4.75	-41.18%	-48.26%	-12.04%
	Average Flow Rate (vehicles per hour)	4038.75	3800.50	3752.25	-5.90%	-7.09%	-1.27%
	Average Speed (kms per hour)	56.24	65.03	67.86	15.63%	20.66%	4.35%
Ramp MOEs	Total Ramp Travel Time (seconds per km)	1607.10	4347.38	4435.97	170.51%	176.02%	2.04%
	Average Ramp Delay (seconds per vehicle)	78.93	407.38	448.81	416.13%	468.62%	10.17%
	Average Flow Rate (vehicles per hour)	948.00	613.25	594.75	-35.31%	-37.26%	-3.02%
System MOEs	Total Travel Time (hours)	**100.425**	**164.65**	**166.21**	**63.95%**	**65.51%**	**0.95%**
	Average Delay Time (seconds per vehicle per km)	**60.99**	**125.31**	**126.88**	**105.46%**	**108.03%**	**1.25%**

Table 5.25 General measures of Effectiveness at traffic demand (3400~1000)

20	Measures of Effectiveness	No Metering	Fuzzy Metering	Genetic Fuzzy Metering	% Change		
					Fuzzy VS. No Metering	GA VS. No Metering	GA VS. Fuzzy
Downstream MOEs	Total Travel Time (seconds per km)	1221.16	885.44	784.05	-27.49%	-35.79%	-11.45%
	Average Delay (seconds per vehicle)	9.75	5.77	4.47	-40.82%	-54.15%	-22.53%
	Average Flow Rate (vehicles per hour)	4249.50	3971.25	3894.00	-6.55%	-8.37%	-1.95%
	Average Speed (kms per hour)	55.21	63.23	68.37	14.53%	23.84%	8.13%
Ramp MOEs	Total Ramp Travel Time (seconds per km)	2435.23	4395.76	4407.18	80.51%	80.98%	0.26%
	Average Ramp Delay (seconds per vehicle)	139.98	417.21	482.36	198.05%	244.59%	15.62%
	Average Flow Rate (vehicles per hour)	913.25	606.75	560.50	-33.56%	-38.63%	-7.62%
System MOEs	Total Travel Time (hours)	**132.24**	**165.31**	**162.51**	**25.01%**	**22.89%**	**-1.69%**
	Average Delay Time (seconds per vehicle per km)	**92.87**	**125.96**	**123.32**	**35.63%**	**32.79%**	**-2.10%**

Table 5.26 General measures of Effectiveness at traffic demand (3600~1000)

21	Measures of Effectiveness	No Metering	Fuzzy Metering	Genetic Fuzzy Metering	% Change		
					Fuzzy VS. No Metering	GA VS. No Metering	GA VS. Fuzzy
Downstream MOEs	Total Travel Time (seconds per km)	1274.74	984.69	834.76	-22.75%	-34.52%	-15.23%
	Average Delay (seconds per vehicle)	9.74	6.46	4.64	-33.68%	-52.36%	-28.17%
	Average Flow Rate (vehicles per hour)	4435.25	4225.50	4100.50	-4.73%	-7.55%	-2.96%
	Average Speed (kms per hour)	55.24	60.50	66.83	9.52%	20.98%	10.46%
Ramp MOEs	Total Ramp Travel Time (seconds per km)	2782.72	4356.58	4440.46	56.56%	59.57%	1.93%
	Average Ramp Delay (seconds per vehicle)	166.50	416.32	514.59	150.04%	209.06%	23.60%
	Average Flow Rate (vehicles per hour)	904.25	603.25	531.50	-33.29%	-41.22%	-11.89%
System MOEs	Total Travel Time (hours)	**150.61**	**168.27**	**162.70**	**11.73%**	**8.03%**	**-3.31%**
	Average Delay Time (seconds per vehicle per km)	**111.24**	**128.98**	**123.43**	**15.95%**	**10.96%**	**-4.30%**

Table 5.27 General measures of Effectiveness at traffic demand (3800~1000)

22	Measures of Effectiveness	No Metering	Fuzzy Metering	Genetic Fuzzy Metering	% Change		
					Fuzzy VS. No Metering	GA VS. No Metering	GA VS. Fuzzy
Downstream MOEs	Total Travel Time (seconds per km)	1280.26	1053.94	845.14	-17.68%	-33.99%	-19.81%
	Average Delay (seconds per vehicle)	7.13	4.45	9.41	-37.59%	31.98%	111.46%
	Average Flow Rate (vehicles per hour)	4538.75	4321.75	4230.25	-4.78%	-6.80%	-2.12%
	Average Speed (kms per hour)	55.92	58.72	66.67	5.01%	19.22%	13.54%
Ramp MOEs	Total Ramp Travel Time (seconds per km)	2883.02	4362.24	4406.52	51.31%	52.84%	1.02%
	Average Ramp Delay (seconds per vehicle)	176.74	418.34	549.49	136.70%	210.90%	31.35%
	Average Flow Rate (vehicles per hour)	888.75	601.50	500.25	-32.32%	-43.71%	-16.83%
System MOEs	Total Travel Time (hours)	**157.62**	**174.82**	**158.75**	**10.91%**	**0.72%**	**-9.19%**
	Average Delay Time (seconds per vehicle per km)	**118.28**	**135.54**	**119.46**	**14.59%**	**1.00%**	**-11.86%**

Table 5.28 General measures of Effectiveness at traffic demand (4000~1000)

23	Measures of Effectiveness	No Metering	Fuzzy Metering	Genetic Fuzzy Metering	% Change		
					Fuzzy VS. No Metering	GA VS. No Metering	GA VS. Fuzzy
Downstream MOEs	Total Travel Time (seconds per km)	1288.18	1117.27	861.05	-13.27%	-33.16%	-22.93%
	Average Delay (seconds per vehicle)	9.50	7.39	4.23	-22.21%	-55.47%	-42.76%
	Average Flow Rate (vehicles per hour)	4540.00	4503.25	4406.75	-0.81%	-2.94%	-2.14%
	Average Speed (kms per hour)	55.65	57.89	66.98	4.03%	20.36%	15.70%
Ramp MOEs	Total Ramp Travel Time (seconds per km)	3365.25	4389.20	4480.10	30.43%	33.13%	2.07%
	Average Ramp Delay (seconds per vehicle)	209.73	421.71	564.42	101.07%	169.12%	33.84%
	Average Flow Rate (vehicles per hour)	885.75	600.75	495.75	-32.18%	-44.03%	-17.48%
System MOEs	Total Travel Time (hours)	**169.75**	**182.17**	**156.56**	**7.32%**	**-7.77%**	**-14.06%**
	Average Delay Time (seconds per vehicle per km)	**130.37**	**142.89**	**117.29**	**9.60%**	**-10.03%**	**-17.92%**

Table 5.29 General measures of Effectiveness at traffic demand (4200~1000)

24	Measures of Effectiveness	No Metering	Fuzzy Metering	Genetic Fuzzy Metering	% Change		
					Fuzzy VS. No Metering	GA VS. No Metering	GA VS. Fuzzy
Downstream MOEs	Total Travel Time (seconds per km)	1286.26	1113.46	906.25	-13.43%	-29.54%	-18.61%
	Average Delay (seconds per vehicle)	9.44	7.29	4.27	-22.78%	-54.77%	-41.43%
	Average Flow Rate (vehicles per hour)	4549.50	4522.25	4647.25	-0.60%	2.15%	2.76%
	Average Speed (kms per hour)	55.78	57.89	65.57	3.78%	17.55%	13.27%
Ramp MOEs	Total Ramp Travel Time (seconds per km)	3526.81	4357.19	4059.71	23.54%	15.11%	-6.83%
	Average Ramp Delay (seconds per vehicle)	222.03	418.53	530.26	88.50%	138.82%	26.70%
	Average Flow Rate (vehicles per hour)	882.00	600.75	480.25	-31.89%	-45.55%	-20.06%
System MOEs	Total Travel Time (hours)	**172.43**	**182.10**	**144.96**	**5.61%**	**-15.93%**	**-20.40%**
	Average Delay Time (seconds per vehicle per km)	**133.08**	**142.27**	**105.67**	**6.91%**	**-20.60%**	**-25.73%**

Analyzing the simulation results of the traffic demand data - Table 5.5

Figure 5.21 shows the comparison of the change of TTT for FLC and genetic fuzzy ramp metering when the ramp demand is 1000 vehicles/hour.

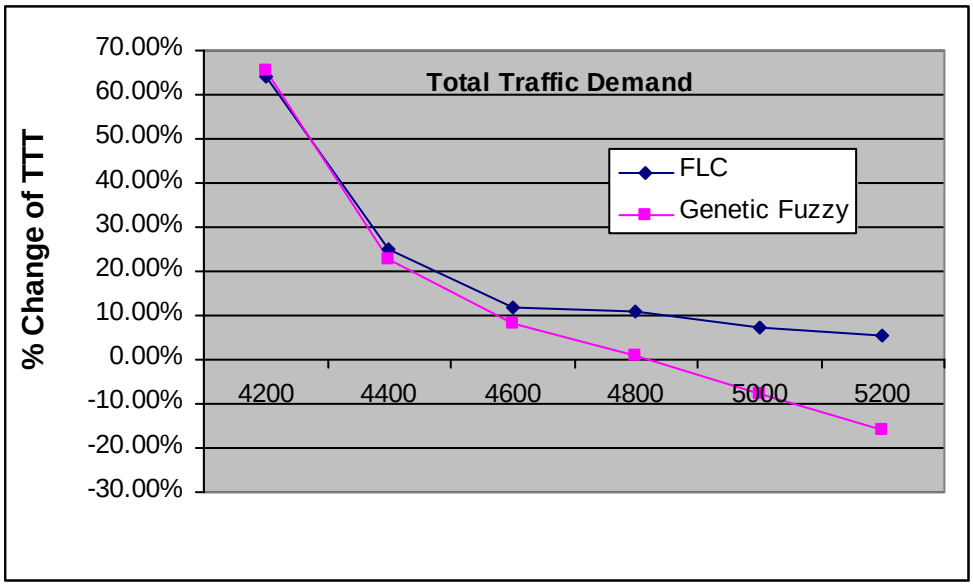


Figure 5.21 the percentage change of TTT when ramp demand is 1000 vehs/h

In Figure 5.21, FLC ramp metering causes the extra delay for the traffic flow and the percentage changes of TTT under FLC are all positive values, while genetic fuzzy ramp metering starts working efficiently when total traffic demand nearly reaches road capacity (5000vehicles/hour) where the reduction of TTT is from -7.77% to -15.93%. The reason why FLC fails to reduce TTT is because ramp demand is relative low and the motorway is under free flow condition in most cases. Although total traffic demand could sometimes exceeds the road capacity and cause traffic delay, low ramp demand hardly continuously interrupt the traffic platoon to make congestion finally formed. FLC metering unable to response the change of traffic flow density could generate a strict metering rate that cause extra traffic delay and finally make the traffic condition worse. The change of traffic flow densities of the Aimsun traffic network shows why FLC and genetic fuzzy ramp metering performs differently when total traffic demand is around road capacity, which is shown in Figure 5.22 ~ Figure 5.24 (where NC means no control and GA-FLC means genetic fuzzy ramp metering).

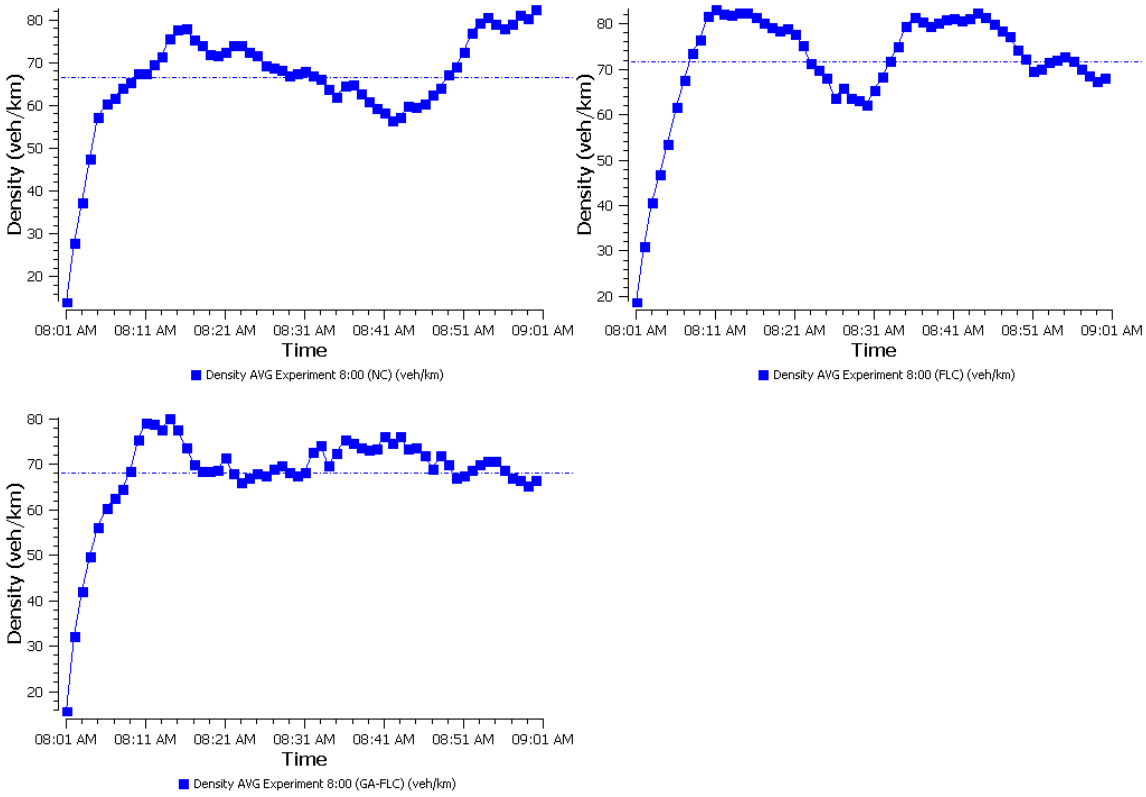


Figure 5.22 the change of average flow density when total demand is 4800 vehs/h

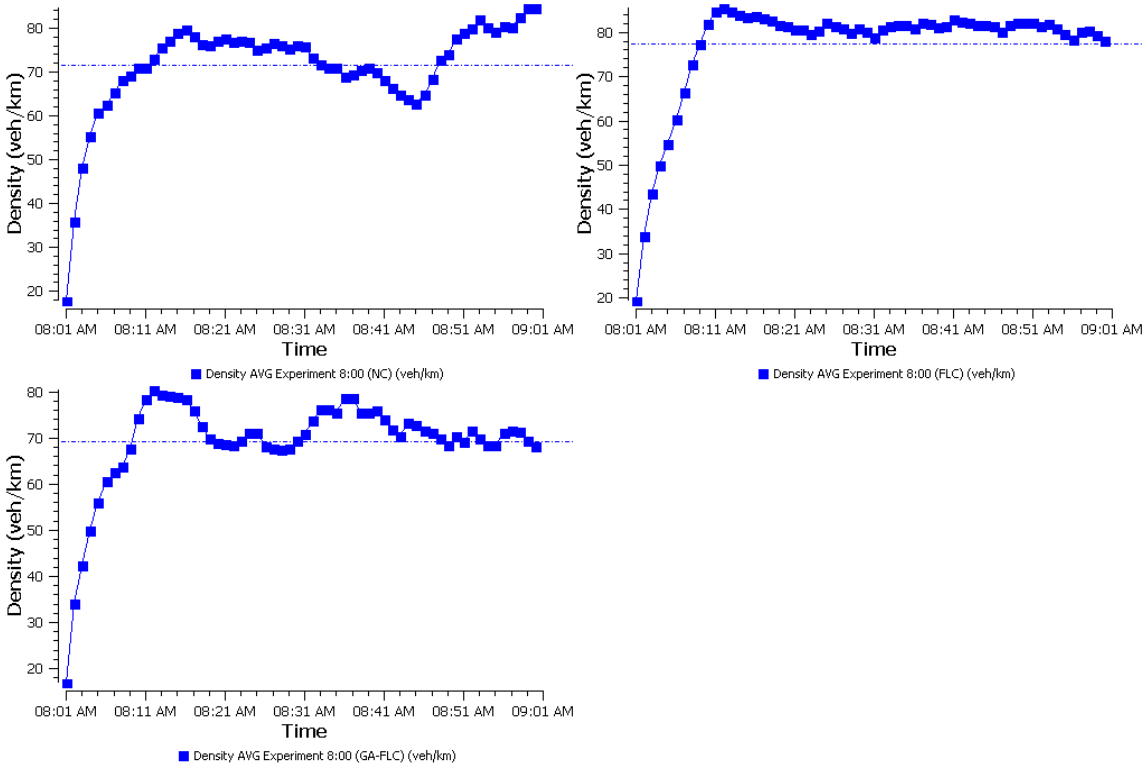


Figure 5.23 the change of average flow density when total demand is 5000 vehs/h

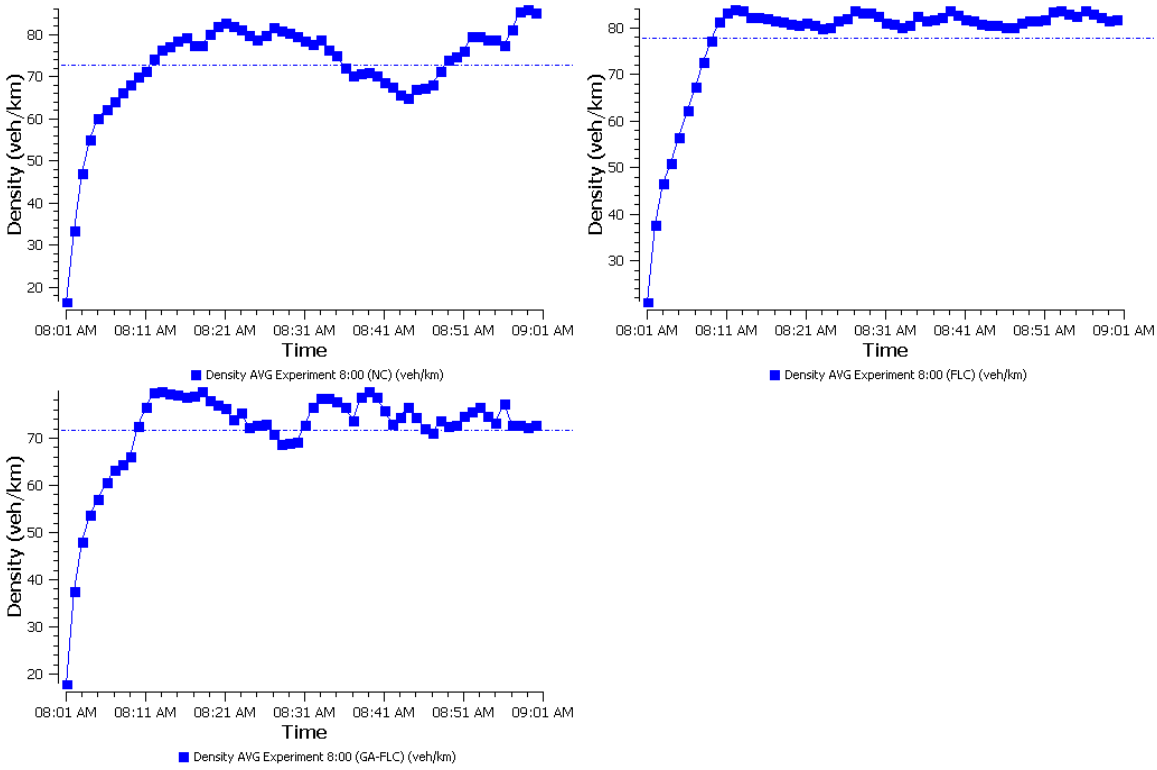


Figure 5.24 the change of average flow density when total demand is 5200 vehs/h

In Figure 5.22, both genetic fuzzy and FLC ramp metering generate a higher average flow density (70vehicles/km) than NC situation (66vehicles/km), which means both of them generate too strict metering rates to benefit traffic condition. Back to Figure 5.21, we can see the fact that the percentage change of TTT under both of the ramp metering approaches is positive (10.91% and 0.72%). In this case, ramp metering will not be necessary since motorway is under free flow condition.

In Figure 5.23 and Figure 5.24, the average flow density under FLC (78vehicles/km) is much higher than the average traffic flow density (72vehicels/km) under NC situation, which means FLC generates the too strict metering rates and already causes extra delay for the traffic flow. And Figure 5.21 shows that the percentage changes of TTT under FLC are positive (7.32% and 5.61%). Meanwhile, genetic fuzzy ramp metering still effectively maintains and stabilizes the flow density at about 70vehicles/km and Figure 5.21 shows that the percentage changes of TTT under genetic fuzzy ramp metering increase to -7.77% and -15.93%.

5.4.5 Overall Result Analysis

So far, the performances of FLC and genetic fuzzy ramp metering have been compared when ramp demand ranges from 1100vehicles/hour to 1600vehicles/hour. Basically, both genetic fuzzy and FLC ramp metering perform well when total traffic demand reaches or nearly reaches road capacity. Once total traffic demand is much higher than road capacity (5400vehicles/hour ~5600vehicles/hour), FLC becomes less efficient to prevent traffic congestion while genetic fuzzy ramp metering sill keep working well due to the objective function based on density-flow relationship. Besides, when ramp demand is relative low (about 1000vheciles/hour), FLC ramp metering can not benefit traffic condition and may cause extra traffic delay, while genetic fuzzy ramp metering still response well when total traffic demand exceeds road capacity.

5.5 Sensitivity Analysis

Sensitivity analysis is the study to show how "sensitive" a model is to the changes of the parameters of the model and to the changes of the structure of the model. In this section, we focus on the parameter sensitivity of the proposed on-ramp model in Aimsun. Parameter sensitivity analysis is usually performed as a series of tests in which the modeler sets different parameter values to study how the model behaves in response to the changes of the parameter values. OFAT (One-Factor-At-a-Time), as one of the simplest ways of investigating the parameter sensitivity of a model, has been applied to the proposed on-ramp model in Aimsun.

Since the all fuzzy parameters in the on-ramp model will be updated by genetic algorithm periodically, it appears unnecessary to analyze them. Therefore, the positions of detectors will be main parameters to be discussed in this section, the current values of which are given by:

The position of the upstream detector: 200m (the distance to the ramp entrance)

The position of the downstream detector: 180m (the distance to the ramp entrance)

The positions of check-in detector and ramp detector are fixed at the end and beginning of the ramp, so they will not be analyzed.

To visualize the effects of the change of detector positions, the positions of detectors that are currently used in the simulation model are considered as base values. The curves of

parameter sensitivity are developed by the percentage change of STTT (System Total Travel Time) based on those values. The small threshold value of 0.5% is used to screen the parameters.

Figure 5.25 and Figure 5.26 shows the sensitivity curve of the parameters.

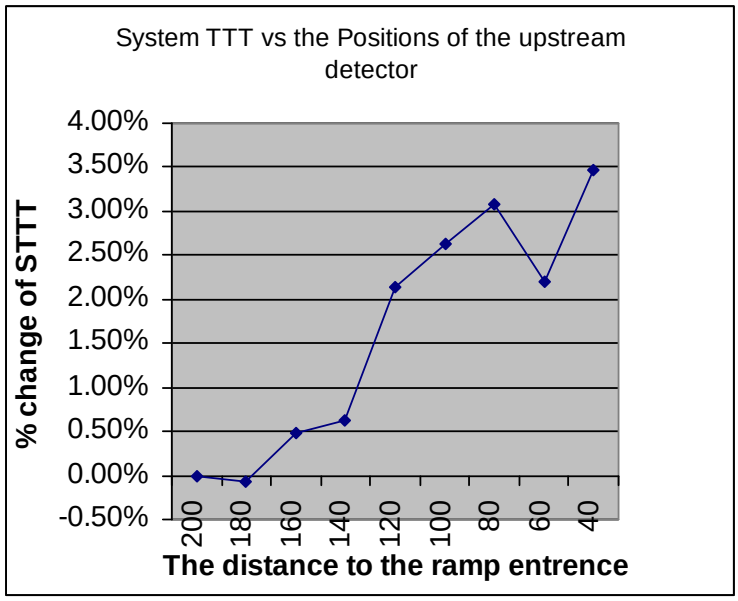


Figure 5.25 % change of TTT vs. the Positions of the upstream detector

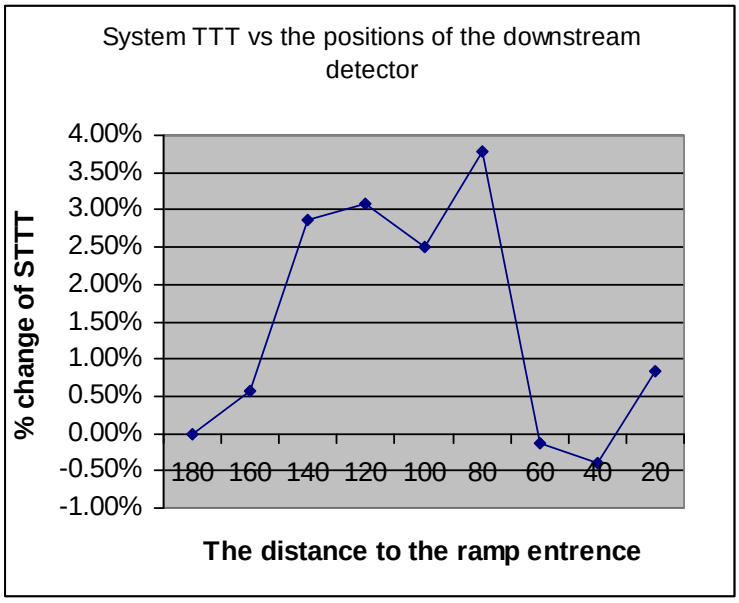


Figure 5.26 % change of TTT vs. the Positions of the downstream detector

5.6 Conclusion

In this chapter, both FLC and genetic fuzzy ramp metering are programmed and tested in Aimsun environment, and totally twenty four traffic scenarios have been simulated to analyze the performance change of the proposed algorithms in different traffic demand. From the comparison of simulation results, genetic fuzzy ramp metering shows more significant improvement on TTT when total traffic demand exceeds 4% of road capacity (about 5200vehicles/hour), while FLC ramp metering tends to be less effective when total traffic demand is much higher than road capacity. The change of system traffic flow density shows the reason why genetic fuzzy perform better than FLC ramp meter in the situation of high traffic demand, which is because the objective function of genetic fuzzy based on the traffic flow-density relationship effectively maintain and stabilize the flow density around the predefined value to avoid the formation of congestion.

Chapter 6 Conclusions and recommendation

6.1 Conclusion

This research proposes a genetic fuzzy approach to design a traffic-responsive ramp control algorithm for an isolated on-ramp. A traditional fuzzy ramp meter is based on the theory of fuzzy logic control, which resembles the approximate reasoning characteristics of human decision making by means of a traffic knowledge base in the form of if-then rules. The common limitation of FLC ramp metering is the dependence on how good the rules could be made by human experts. Therefore, the optimization of FLC ramp metering is actually the optimization of rule-base. Instead of making new rules, tuning fuzzy parameters for fuzzy sets is easy to be implemented due to the avoidance of changing the structure of fuzzy rule-base, so genetic algorithm is used as an evaluation algorithm to optimize the fuzzy parameters based on given evaluation criteria.

The simulation in Aimsun on-ramp model shows significant improvement on TTT when fuzzy parameters are properly tuned, and some conclusions for the achievement of such improvement could be summarized as follows:

a) Although fuzzy logic control based on inexact input information do not need a mathematic model, to properly tune fuzzy parameters, the objective function must have an exact mathematic expression. In this report, a function derived from traffic flow-density relationship of an isolated on-ramp model is used as objective function.

b) To prevent the ramp metering rate neither to be too permissive nor to be too restrictive, a macroscopic traffic model should be taken into consideration. The assumption of Greenshield's macroscopic stream model is involved in the Aimsun on-ramp mode of this paper.

c) To proper tune the fuzzy parameters, a proper evolutionary algorithm should be selected. Genetic algorithm has been proved to be an easy-implementing and simple effective way to optimize the FLC ramp metering, and the genetic fuzzy ramp metering shows the stronger stability of maintaining system traffic flow density especially when handling high traffic demand situation.

6.2 Recommendation

Although the genetic fuzzy ramp metering shows better performance to control the mainstream traffic condition in Aimsun simulation, the simulation results also show it might cause longer ramp delay time than FLC or NC. The reason for that is because the objective function only focuses on the mainstream flow density. In other words, the consideration about the mainstream traffic condition is the first priority for the ramp control algorithm, and the limit of ramp queue length is out of consideration in the objective function and only restricted by fuzzy rules. Certainly, a long ramp queue will not affect the traffic condition on freeway, but it might cause the congestion on surface streets due to the spill-back of the long ramp queue, so a ramp queue control function is recommended to be involved in the objective function if the simulation scenario is not a isolated on-ramp model but a complete traffic network.

Also, for a motorway network including several on-ramps, the genetic fuzzy ramp metering algorithm, as a local traffic responsive ramp metering algorithm, might not be able to optimize traffic condition for the whole network. Then a system objective function is going to be necessary for the system optimization to tune the fuzzy parameters. The objective function could be derived by a second order traffic model representing the traffic network, which could be found in many publications [1] [15] [28]. With this system objective function, the fuzzy genetic ramp metering algorithm could be tuned as a coordinated ramp metering algorithm to calculate a system metering rate for each ramp. Meanwhile, the fuzzy genetic algorithm could also work as a local traffic responsive ramp meter to calculate a local ramp metering rate based on the local traffic condition as we proposed before. The more restrictive one between the local and system metering rate could be finally applied to the corresponding ramp meter. This is what we call the competitive ramp metering algorithm. It must be very interesting to see the feasibility of genetic fuzzy algorithm working on such an assumption.

REFERENCES

[1] Kachroo, P., Ozbay, K., (2003), "Feedback Ramp Metering in Intelligent Transportation Systems," Kluwer Academic 2003, 38-45.

[2] Deliverable D7.5 Handbook of Ramp Metering, IBI Group UK Ltd, 2004.

[3] Guang, Y. X., Lei, N., (2006), "Research on Evaluation of Expressway Ramp Isolated Metering Strategy in Shanghai," ITS Telecommunications Proceedings, 2006 6th International Conference.

[4] Bogenberger, K., and May, A.D., (1999), "Advanced Coordinated Traffic Responsive Ramp Metering Strategies," Berkeley.

[5] Nadeem, A., Zongzhong, T., Messer, C. J., and Chu, C. L., (2003), "Ramp Metering Algorithms and Approaches for Texas," Texas Transportation Institute.

[6] Horowitz, R., May, A.D., Skabardonis, A., Varaiya, P., Zhang, M., Gomes, G., Muñoz, L., Sun, X.T., Sun, D.F., (2005), "Field Implementation and Evaluation of Adaptive Ramp Metering Algorithms," Research Report, University of California.

[7] Joseph, R., (2003), "Evaluation of Coordinated and Local Ramp Metering Algorithms using Microscopic Traffic Simulation", Master Thesis, University of Rhode Island.

[8] Jin, W. L., Zhang, M., (2000), "Evaluation of On-ramp Control Algorithms", Interim Report, University of California at Davis.

[9] Taylor, C., and Meldrum, D., (1995), "Simulation testing of a Fuzzy Neural Ramp Metering Algorithm," final technical report, Washington State Department of Transportation, National Technical Information Service, WA-RD 395.1.

[10] Bogenberger, K., (2000), "Adaptive Fuzzy Systems for Traffic Responsive and Coordinated Ramp Metering," Ph.D. Thesis at the Fachgebiet.

[11] Taylor, C., Meldrum, D., Jacobson, L., (1995), "Fuzzy Ramp Metering - Design Overview and Simulation Results", Transportation Research Record 1634, Washington.

[12] Tale, H., Slager, J., Rosloot, J., (1996), "The Assessment of Ramp Metering Based on Fuzzy Logic," 3rd ITS World Congress in Orlando.

[13] Taylor, C., and Meldrum, D., (2000), "Evaluation of a Fuzzy Logic Ramp Metering Algorithm: A Comparative Study between Three Ramp Metering Algorithms used in

the Greater Seattle Area," WA-RD Technical Report to be published, Washington State Department of Transportation, National Technical Information Service.

[14] Bogenberger, K., Keller, H., (2001), "An Evolutionary Fuzzy System for Coordinated and Traffic Responsive Ramp Metering," Annual Hawaiian International Conference on System Sciences, USA.

[15] Bogenberger, K., Vukanovic, S., Keller, H., (2001), "A Neuro-Fuzzy Algorithm for Coordinated Traffic Responsive Ramp Metering," IEEE 4th International Conference on Intelligent Transportation Systems, California.

[16] Taylor, C., and Meldrum, D., (2000), "Algorithm Design, User Interface, and Optimization Procedure for a Fuzzy Logic Ramp Metering Algorithm: A Training Manual for Motorway Operations Engineers," WA-RD Technical Report to be published, Washington State Department of Transportation, National Technical Information Service.

[17] Roger, J.-S., (1993), "ANFIS: Adaptive-Network-Based Fuzzy Inference Systems," IEEE Transactions on Systems, Man, and Cybernetics, Vol. 23, No. 03, pp 665-685.

[18] Lee, C., (1990), "Fuzzy logic in control systems: fuzzy logic controller-part 1 and 2," IEEE Transactions on Systems, Man, and Cybernetics.

[19] Fuzzy Logic Toolbox User's Guide, The MathWorks, 2008.

[20] Hoffmann, F., (2001), "Evolutionary Algorithm for Fuzzy Control System Design," Proceedings of the IEEE, VOL.89, NO. 9.

[21] Mohammadian, M., Stonier, R. J., (1994), "Tuning and Optimisation of Membership functions of Fuzzy Logic Controllers by Genetic Algorithm", IEEE International Workshop on Robot and Human Communication 0-7803-2002-6/94.

[22] Steeb, W.-H., (2002), "The Nonlinear Workbook," Singapore: World Scientific Publishing Co. Pte. Ltd.

[23] Sun, X.T., (2005), "Modelling, Estimation, and Control of Motorway Traffic," Ph.D. Thesis at the University of California, Berkeley.

[24] Microsimulator and Mesosimulator in Aimsun 6 User's Manual, TSS-Transport Simulation Systems, 2008,

[25] Aimsun6 API Manual, TSS-Transport Simulation Systems, 2008.

[26] Hughes, J., (1998), “Intensive Traffic Data Collection for Simulation of a Congested Auckland Motorway ,” Proceedings 19th ARRB Transport Research Conference, Sydney, Australia, 15pp.

[27] Hughes, J., (2000), “AIMSUN2 Simulation of a Congested Auckland Freeway,” Master thesis at the University of Auckland.

[28] Ghods, A.H., Kian, A. R., Tabibi, M., (2007), “A Genetic-Fuzzy Control Application to Ramp Metering and Variable Speed Limit Control,” IEEE International conference on Systems, Man and Cybernetics, ISIC.

[29] Papageorgiou, M., and Kotsialos, A., (2002), “Motorway ramp metering: An overview,” IEEE Transactions on Intelligent Transportation Systems.

Appendix A: Fuzzy logic control coding for ramp metering

- AAPI.CXX

```
#include "AKIProxie.h"
#include "CIProxie.h"
#include "ANGConProxie.h"
#include "AAPI.h"
#include <stdio.h>
#include "FLC.h"
// Procedures could be modified by the user
char astring[128];

int AAPILoad()
{
      // AKIPrintString("LOAD");
         return 0;
}

int AAPIInit()
{
      // AKIPrintString("\tInit");
      ANGConnEnableVehiclesInBatch(true);
      return 0;
}

int AAPIManage(double time, double timeSta, double timTrans, double
acicle)
{
//detectors setup
float D_v_s, D_up_flow, D_down_flow, D_up_occ, D_up_speed, D_down_speed,
D_queue_occ, D_checkin_occ;

//read detecor
D_up_occ=(float)(AKIDetGetTimeOccupedAggregatedbyId(248,NULL));
D_up_speed=(float)(AKIDetGetSpeedAggregatedbyId(248,NULL));
D_up_flow=(float)(60*(AKIDetGetCounterAggregatedbyId(248,NULL)));
D_down_speed=(float)(AKIDetGetSpeedAggregatedbyId(250,NULL));
D_queue_occ=(float)(AKIDetGetTimeOccupedAggregatedbyId(249,NULL));
D_checkin_occ=(float)(AKIDetGetTimeOccupedAggregatedbyId(247,NULL));
D_down_flow=(float)(60*(AKIDetGetCounterAggregatedbyId(250,NULL)));
D_v_s=(float)(D_down_flow/3990);

      //reading display
      /*sprintf_s(astring,"D_up_occ is %f\n",D_up_occ);
      AKIPrintString(astring);
      sprintf_s(astring,"D_up_speed is %f\n",D_up_speed);
      AKIPrintString(astring);
      sprintf_s(astring,"D_up_flow is %f\n",D_up_flow);
      AKIPrintString(astring);
      sprintf_s(astring,"D_down_speed is %f\n",D_down_speed);
      AKIPrintString(astring);
      sprintf_s(astring,"D_queue_occ is %f\n",D_queue_occ);
      AKIPrintString(astring);
      sprintf_s(astring,"D_checkin_occ is %f\n",D_checkin_occ);
      AKIPrintString(astring);
      sprintf_s(astring,"D_down_flow is %f\n",D_down_flow);
      AKIPrintString(astring);
      sprintf_s(astring,"D_V_S is %f\n",D_v_s);
```

```
      AKIPrintString(astring);*/

      //initialize the inputs
float local_occ[7], local_speed[7], local_flow[7], downstream_vc[3],
downstream_speed[3], checkin_occ[3], queue_occ[3];
      //input 1
      if(D_up_speed>=0)
      {local_speed[0]=D_up_speed;}
      else
      {local_speed[0] = 0;}
      local_speed[1]=21.5; local_speed[2]=0;
      local_speed[3]=21.5; local_speed[4]=50;
      local_speed[5]=21.5; local_speed[6]=100;
      //input 2
      if(D_up_flow>=0)
      {local_flow[0]=D_up_flow;}
      else
      {local_flow[0] = 0;}
      local_flow[1]=850; local_flow[2]=0;
      local_flow[3]=850; local_flow[4]=2000;
      local_flow[5]=850; local_flow[6]=4000;
      //input 3
      if(D_up_occ>=0)
      {local_occ[0]=D_up_occ;}
      else
      {local_occ[0] = 0;}
      local_occ[1]=6.4f; local_occ[2]=0;
      local_occ[3]=6.4f; local_occ[4]=15;
      local_occ[5]=6.4f; local_occ[6]=30;
      //input 4
      if(D_v_s>=0)
      {downstream_vc[0]=D_v_s;}
      else
      {downstream_vc[0] = 0;}
      downstream_vc[1]=6.5; downstream_vc[2]=0.5;
      //input 5
      if(D_down_speed>=0)
      {downstream_speed[0]=D_down_speed;}
      else
      {downstream_speed[0] = 0;}
      downstream_speed[1]=-0.25; downstream_speed[2]=65;
      //input 6
      if(D_checkin_occ>=0)
      {checkin_occ[0]=D_checkin_occ;}
      else
      {checkin_occ[0] = 0;}
      checkin_occ[1]=0.4f; checkin_occ[2]=20;
      //input 7
      if(D_queue_occ>=0)
      {queue_occ[0]=D_queue_occ;}
      else
      {queue_occ[0] = 0;}
      queue_occ[1]=0.4f; queue_occ[2]=20;

      //calculating FLC metering rate
float flow_rate;
```

```
flow_rate=flcMeterRate(local_occ,  local_speed,  local_flow,downstream_vc,
downstream_speed, checkin_occ, queue_occ);

      static int i=1;
      if(i==80)
      {
            ECIChangeParametersFlowMeteringById(245,timeSta,flow_rate,
            flow_rate,flow_rate);
                  i=0;
      }
      i=i+1;

      //sprintf_s(astring,"meter_rate is %f\n",flow_rate);
      //AKIPrintString(astring);

      return 0;
}


int AAPIPostManage(double time, double timeSta, double timTrans, double
acicle)
{
      // AKIPrintString("\tPostManage");
      return 0;
}

int AAPIFinish()
{
      // AKIPrintString("\tFinish");
      return 0;
}

int AAPIUnLoad()
{
      // AKIPrintString("UNLOAD");
      return 0;

}
```

- FLC

```
#include <stdio.h>
#include <iostream>
#include <math.h>
#include "FLC.h"

//input1 definition
float local_speed_med(float local_speed, float q /*initial value 25.5*/,
float c/*intial value 60*/)
{ double u;
  float v;
  if(local_speed<0)
      {return(0);}
  if(local_speed>100)
      {return(0);}
  if(local_speed>=0&&local_speed<=100)
      {v=-((local_speed-c)*(local_speed-c))/(2*q*q);
      u=exp(double(v));
      return (float(u));}
  return -1;
}
float local_speed_high(float local_speed, float q, float c)
{ double u;
  float v;
  if(local_speed<0)
      {return(0);}
  if(local_speed>100)
      {return(1);}
  if(local_speed>=0&&local_speed<=100)
      {v=-((local_speed-c)*(local_speed-c))/(2*q*q);
      u=exp(double(v));
      return (float(u));}
  return -1;
}
float local_speed_low(float local_speed, float q, float c)
{ double u;
  float v;
  if(local_speed<0)
      {return(1);}
  if(local_speed>100)
      {return(0);}
  if(local_speed>=0&&local_speed<=100)
      {v=-((local_speed-c)*(local_speed-c))/(2*q*q);
      u=exp(double(v));
      return (float(u));}
  return -1;
}

//input2 definition
float local_flow_med(float local_flow, float q/*initial value 850 */,
float c/*intial value 2000*/)
{ double u;
  float v;
  if(local_flow<0)
      {return(0);}
  if(local_flow>4000)
      {return(0);}
```

```
  if(local_flow>=0&&local_flow<=4000)
      {v=-((local_flow-c)*(local_flow-c))/(2*q*q);
      u=exp(double(v));
      return (float(u));}
  return -1;
}
float local_flow_high(float local_flow, float q, float c)
{ double u;
  float v;
  if(local_flow<0)
      {return(0);}
  if(local_flow>4000)
      {return(1);}
  if(local_flow>=0&&local_flow<=4000)
      {v=-((local_flow-c)*(local_flow-c))/(2*q*q);
      u=exp(double(v));
      return (float(u));}
  return -1;
}
float local_flow_low(float local_flow, float q, float c)
{ double u;
  float v;
  if(local_flow<0)
      {return(1);}
  if(local_flow>4000)
      {return(0);}
  if(local_flow>=0&&local_flow<=4000)
      {v=-((local_flow-c)*(local_flow-c))/(2*q*q);
      u=exp(double(v));
      return (float(u));}
  return -1;
}

//input3 definition
float local_occ_med(float local_occ, float q/*initial value 6.4 */,
float c/*intial value 15*/)
{ double u;
  float v;
  if(local_occ<0)
      {return(0);}
  if(local_occ>30)
      {return(0);}
  if(local_occ>=0&&local_occ<=30)
      {v=-((local_occ-c)*(local_occ-c))/(2*q*q);
      u=exp(double(v));
      return (float(u));}
  return -1;
}
float local_occ_high(float local_occ, float q, float c)
{ double u;
  float v;
  if(local_occ<0)
      {return(0);}
  if(local_occ>30)
      {return(1);}
  if(local_occ>=0&&local_occ<=30)
      {v=-((local_occ-c)*(local_occ-c))/(2*q*q);
```

```
      u=exp(double(v));
      return (float(u));}
  return -1;
}
float local_occ_low(float local_occ, float q, float c)
{ double u;
  float v;
  if(local_occ<0)
      {return(1);}
  if(local_occ>30)
      {return(0);}
  if(local_occ>=0&&local_occ<=30)
      {v=-((local_occ-c)*(local_occ-c))/(2*q*q);
      u=exp(double(v));
      return (float(u));}
  return -1;
}
//input4 definition
float downstream_vc_high(float downstream_vc, float q/*initial value
6.5*/, float c)
{double u;
  float v;
  if(downstream_vc<0)
      {return(0);}
  if(downstream_vc>1)
      {return(1);}
  if(downstream_vc>=0&&downstream_vc<=1)
      {v=-q*(downstream_vc-c);
      u=1/(1+exp(double(v)));
      return (float(u));}
  return -1;
}



//input5 definition
float downstream_speed_low(float downstream_speed, float q/*initial
value -0.25*/, float c)
{double u;
  float v;
  if(downstream_speed<0)
      {return(1);}
  if(downstream_speed>100)
      {return(0);}
  if(downstream_speed>=0&&downstream_speed<=100)
      {v=-q*(downstream_speed-c);
      u=1/(1+exp(double(v)));
      return (float(u));}
  return -1;
}
//input6 definition
float checkin_occ_high(float checkin_occ, float q/*initial value 0.4*/,
float c)
{double u;
  float v;
  if(checkin_occ<0)
      {return(0);}
```

```
  if(checkin_occ>50)
      {return(1);}
  if(checkin_occ>=0&&checkin_occ<=50)
      {v=-q*(checkin_occ-c);
       u=1/(1+exp(double(v)));
       return (float(u));}
  return -1;
}
//input7 definition
float queue_occ_high(float queue_occ, float q/*initial value 0.4*/,
float c)
{double u;
  float v;
  if(queue_occ<0)
      {return(0);}
  if(queue_occ>50)
      {return(1);}
  if(queue_occ>=0&&queue_occ<=50)
      {v=-q*(queue_occ-c);
       u=1/(1+exp(double(v)));
       return (float(u));}
  return -1;
}
// FLC code
float flcMeterRate(float *local_occ, float *local_speed, float
*local_flow, float *downstream_vc, float *downstream_speed, float
*checkin_occ, float *queue_occ)
{
//evaluate each rule
float rule[9];
rule[0]=local_occ_low(*local_occ,*(local_occ+1),*(local_occ+2));
rule[1]=local_occ_med(*local_occ,*(local_occ+3),*(local_occ+4));
rule[2]=local_occ_high(*local_occ,*(local_occ+5),*(local_occ+6));
rule[3]=MIN(local_speed_low(*local_speed,*(local_speed+1),*(local_speed
+2)),local_flow_high(*local_flow,*(local_flow+5),*(local_flow+6)));
rule[4]=MIN(local_speed_med(*local_speed,*(local_speed+3),*(local_speed
+4)),local_occ_high(*local_occ,*(local_occ+5),*(local_occ+6)));
rule[5]=MIN(local_speed_med(*local_speed,*(local_speed+3),*(local_speed
+4)),local_occ_low(*local_occ,*(local_occ+1),*(local_occ+2)));
rule[6]=MIN(local_speed_high(*local_speed,*(local_speed+5),*(local_spee
d+6)),local_flow_low(*local_flow,*(local_flow+1),*(local_flow+2)));
rule[7]=MIN(downstream_speed_low(*downstream_speed,*(downstream_speed+1
),*(downstream_speed+2)),downstream_vc_high(*downstream_vc,*(downstream
_vc+1),*(downstream_vc+2)));
rule[8]=MAX(checkin_occ_high(*checkin_occ,*(checkin_occ+1),*(checkin_oc
c+2)),queue_occ_high(*queue_occ,*(queue_occ+1),*(queue_occ+2)));
      //the weighted sum of each rule class outcome
      //meter_rate_high is 0, meter_rate_low is 1, meter_rate_med is 2
      float meter_rate_class[3];
      meter_rate_class[0]=rule[0]*(3/2)+rule[5]*1+rule[6]*1+rule[8]*3;
      meter_rate_class[1]=rule[2]*2+rule[3]*2+rule[7]*3;
      meter_rate_class[2]=rule[1]*(3/2)+rule[4]*1;
      //defuzzification (discreted centroids method)
      float base=0.5;
      float centroid=0;
      float area=0;
      float num=0;
```

```
        float den=0;
        float LL=240;
        float HL=900;
        float meter_rate;

        for(int i=0;i<3;i++)
        {
              if(i==0)
              { area=base/2;
                centroid=1-base/3;}
              else if(i==1)
              { area=base/2;
                centroid=base/3;}
              else
              { area=base;
                centroid=base;}
                num+=meter_rate_class[i]*area*centroid;
                den+=meter_rate_class[i]*area;
               }
        // calculate metering rate and rescale to LL and HH range
         meter_rate = (HL-LL)*(num/den+LL/(HL-LL));
        //std::cout << "meter_rate\t"<< meter_rate << std::endl;
        return(meter_rate);

}
```

Appendix B: Genetic fuzzy control coding for ramp metering

- AAPI.CXX

```
// Genetic fuzzy control coding for ramp metering
// Yu Xue Feng - MASTER STUDENT OF ENGINEERING IN MECHATRONICS
// SEAT - MASSEY UNIVERSITY - AUCKLAND - NEW ZEALAND
// 2008-2009 - All rights reserved

#include "AKIProxie.h"
#include "CIProxie.h"
#include "ANGConProxie.h"
#include "AAPI.h"
#include <stdio.h>
#include "FLC.h"
#include "GA_fuzzy.h"
#include <math.h>
#include <time.h>
// Procedures could be modified by the user
char astring[128];
int up_count, down_count, ramp_count;
float average_count;
//detectors setup
float D_v_s=0, D_up_flow=0, D_down_flow=0, D_up_occ=0, D_up_speed=0,
D_down_speed=0, D_queue_occ=0, D_checkin_occ=0;
//average value
float V_C=0, Upflow=0, Downflow=0, Upocc=0, Upspeed=0, Downspeed=0,
Queueocc=0, Checkinocc=0;
//initialize the inputs
float local_occ[7], local_speed[7], local_flow[7], downstream_vc[3],
downstream_speed[3], checkin_occ[3], queue_occ[3];
double L_speed=0, M_speed=50, H_speed=100,
       L_flow=0, M_flow=2000, H_flow=4000,
       L_occ=0, M_occ=15, H_occ=30,  V_c=0.5,
       D_speed=65,  Check_in=20, Q_occ=20;
double *Ls=&L_speed, *Ms=&M_speed, *Hs=&H_speed,
       *Lf=&L_flow, *Mf=&M_flow, *Hf=&H_flow,
       *Lo=&L_occ, *Mo=&M_occ, *Ho=&H_occ, *Vc=&V_c,
       *Ds=&D_speed, *Checkin=&Check_in, *Qocc=&Q_occ;

int AAPILoad()
{
      // AKIPrintString("LOAD");
         return 0;
}

int AAPIInit()
{
      // AKIPrintString("\tInit");
      ANGConnEnableVehiclesInBatch(true);
      return 0;
}

int AAPIManage(double time, double timeSta, double timTrans, double
acicle)
{
      //AKIPrintString("\tManage");
```

```
//read detecor
D_up_occ=(float)(AKIDetGetTimeOccupedAggregatedbyId(248,NULL));
D_up_speed=(float)(AKIDetGetSpeedAggregatedbyId(248,NULL));
D_up_flow=(float)(60*(AKIDetGetCounterAggregatedbyId(248,NULL)));
up_count=AKIDetGetCounterAggregatedbyId(248,NULL);
D_down_speed=(float)(AKIDetGetSpeedAggregatedbyId(250,NULL));
D_queue_occ=(float)(AKIDetGetTimeOccupedAggregatedbyId(249,NULL));
D_checkin_occ=(float)(AKIDetGetTimeOccupedAggregatedbyId(247,NULL));
D_down_flow=(float)(60*(AKIDetGetCounterAggregatedbyId(250,NULL)));
down_count=AKIDetGetCounterAggregatedbyId(250,NULL);
ramp_count=AKIDetGetCounterAggregatedbyId(247,NULL);
D_v_s=(float)(D_down_flow/4200);

      //reading display
      /*sprintf_s(astring,"D_up_occ is %f\n",D_up_occ);
      AKIPrintString(astring);
      sprintf_s(astring,"D_up_speed is %f\n",D_up_speed);
      AKIPrintString(astring);
      sprintf_s(astring,"D_up_flow is %f\n",D_up_flow);
      AKIPrintString(astring);
      sprintf_s(astring,"D_down_speed is %f\n",D_down_speed);
      AKIPrintString(astring);
      sprintf_s(astring,"D_queue_occ is %f\n",D_queue_occ);
      AKIPrintString(astring);
      sprintf_s(astring,"D_checkin_occ is %f\n",D_checkin_occ);
      AKIPrintString(astring);
      sprintf_s(astring,"D_down_flow is %f\n",D_down_flow);
      AKIPrintString(astring);
      sprintf_s(astring,"D_V_S is %f\n",D_v_s);
      AKIPrintString(astring);*/

      //input 1
      if(D_up_speed>=0)
      {local_speed[0]=D_up_speed;}
      else
      {local_speed[0] = 0;}
      local_speed[1]=21.5; local_speed[2]=float(L_speed);
      local_speed[3]=21.5; local_speed[4]=float(M_speed);
      local_speed[5]=21.5; local_speed[6]=float(H_speed);
      //input 2
      if(D_up_flow>=0)
      {local_flow[0]=D_up_flow;}
      else
      {local_flow[0] = 0;}
      local_flow[1]=850; local_flow[2]=float(L_flow);
      local_flow[3]=850; local_flow[4]=float(M_flow);
      local_flow[5]=850; local_flow[6]=float(H_flow);
      //input 3
      if(D_up_occ>=0)
      {local_occ[0]=D_up_occ;}
      else
      {local_occ[0] = 0;}
      local_occ[1]=6.4f; local_occ[2]=float(L_occ);
      local_occ[3]=6.4f; local_occ[4]=float(M_occ);
      local_occ[5]=6.4f; local_occ[6]=float(H_occ);
      //input 4
      if(D_v_s>=0)
```

```
{downstream_vc[0]=D_v_s;}
else
{downstream_vc[0] = 0;}
downstream_vc[1]=6.5; downstream_vc[2]=float(V_c);
//input 5
if(D_down_speed>=0)
{downstream_speed[0]=D_down_speed;}
else
{downstream_speed[0] = 0;}
downstream_speed[1]=-0.25; downstream_speed[2]=float(D_speed);
//input 6
if(D_checkin_occ>=0)
{checkin_occ[0]=D_checkin_occ;}
else
{checkin_occ[0] = 0;}
checkin_occ[1]=0.4f; checkin_occ[2]=float(Check_in);
//input 7
if(D_queue_occ>=0)
{queue_occ[0]=D_queue_occ;}
else
{queue_occ[0] = 0;}
queue_occ[1]=0.4f; queue_occ[2]=float(Q_occ);


//calculating FLC metering rate
if(average_count>15)
{
      float flow_rate;
      flow_rate=flcMeterRate(local_occ, local_speed,local_flow,
      downstream_vc, downstream_speed, checkin_occ, queue_occ);

      static int i=1;
      if(i==80)
      {

            ECIChangeParametersFlowMeteringById(245,timeSta,flow_
            rate,flow_rate,flow_rate);
            i=0;
            //sprintf_s(astring,"meter_rate is %f\n",flow_rate);
            //AKIPrintString(astring);
      }
      i=i+1;

}
else
{
      float flow_rate;
      flow_rate=800;
      ECIChangeParametersFlowMeteringById(245,timeSta,flow_rate,
      flow_rate,flow_rate);
      //sprintf_s(astring,"meter_rate is %f\n",flow_rate);
      //AKIPrintString(astring);
}
```

```
	return 0;
}


int AAPIPostManage(double time, double timeSta, double timTrans, double
acicle)
{
	static int j=1,k=1;
	static float cars=0,total_cars=0;

	if(j==80)
	{
	//input 1
	if(D_up_speed>=0)
	{Upspeed=Upspeed+D_up_speed;}

	//input 2
	if(D_up_flow>=0)
	{Upflow=Upflow+D_up_flow;}

	//input 3
	if(D_up_occ>=0)
	{Upocc=Upocc+D_up_occ;}

	//input 4
	if(D_v_s>=0)
	{V_C=V_C+D_v_s;}

	//input 5
	if(D_down_speed>=0)
	{Downspeed=Downspeed+D_down_speed;}

	//input 6
	if(D_checkin_occ>=0)
	{Checkinocc=Checkinocc+D_checkin_occ;}

	//input 7
	if(D_queue_occ>=0)
	{Queueocc=Queueocc+D_queue_occ;}

	if(up_count<0&&ramp_count<0&&down_count<0)
		cars=0;
	else
		{cars=cars+(up_count+ramp_count-down_count);
		 total_cars=cars+total_cars;}

	//calcualte the average inputs in 5 minutes
	if(k==5)
	{
	 local_speed[0]=Upspeed/5;
	 local_flow[0]=Upflow/5;
	 local_occ[0]=Upocc/5;
	 downstream_vc[0]=V_C/5;
	 downstream_speed[0]=Downspeed/5;
	 checkin_occ[0]=Checkinocc/5;
	 queue_occ[0]=Queueocc/5;
	 average_count=total_cars/5;
```

```
        Upspeed=0;
        Upflow=0;
        Upocc=0;
        V_C=0;
        Downspeed=0;
        Checkinocc=0;
        Queueocc=0;
        total_cars=0;

        // genetic tuning process to update the fuzzy parameters
        double flow;
        if(average_count>=15&& average_count<=30)
        {
             flow=60*(30-average_count);
             GA(local_occ,local_speed,local_flow,downstream_vc,
             downstream_speed,checkin_occ,queue_occ,flow,Ls,Ms,Hs,Lf,Mf,
             Hf,Lo,Mo,Ho,Vc,Ds,Checkin,Qocc);
             //sprintf_s(astring," the tuned flow rate is %f\n",flow);
             //AKIPrintString(astring);
        }
        if(average_count>30)
        {
              flow=240;

             GA(local_occ,local_speed,local_flow,downstream_vc,downstrea
             m_speed,checkin_occ,queue_occ,flow,Ls,Ms,Hs,Lf,Mf,Hf,Lo,Mo,
             Ho,Vc,Ds,Checkin,Qocc);
              //sprintf_s(astring,"flow rate is %f\n",flow);
              //AKIPrintString(astring);
        }

        k=0;
       }
       k=k+1;
       j=0;
       }
       j=j+1;

       // AKIPrintString("\tPostManage");
       return 0;
}

int AAPIFinish()
{
       // AKIPrintString("\tFinish");
       return 0;
}

int AAPIUnLoad()
{
       // AKIPrintString("UNLOAD");
       return 0;
}
```

- GA-FLC.cpp

```
// Genetic fuzzy control coding for ramp metering
// Yu Xue Feng - MASTER STUDENT OF ENGINEERING IN MECHATRONICS
// SEAT - MASSEY UNIVERSITY - AUCKLAND - NEW ZEALAND
// 2008-2009 - All rights reserved

#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <iostream>
#include "FLC.h"
#include "GA_fuzzy.h"

double f(float* local_occ, float* local_speed, float* local_flow,
float* downstream_vc, float* downstream_speed, float* checkin_occ,
float* queue_occ,double flow,double Lspeed,double Mspeed,double
Hspeed,double Lflow,double Mflow,double Hflow,double Locc,double
Mocc,double Hocc,double Vc,double Dspeed, double Checkin,double Qocc)
{
      //input 1
      *(local_speed+2)=float(Lspeed);
      *(local_speed+4)=float(Mspeed+Lspeed);
      *(local_speed+6)=float(Hspeed+Mspeed+Hspeed);
      //input 2
      *(local_flow+2)=float(Lflow);
      *(local_flow+4)=float(Mflow+Lflow);
      *(local_flow+6)=float(Hflow+Mflow+Lflow);
      //input 3
      *(local_occ+2)=float(Locc);
      *(local_occ+4)=float(Mocc+Locc);
      *(local_occ+6)=float(Hocc+Mocc+Locc);
      //input 4
      *(downstream_vc+2)=float(Vc);
      //input 5
      *(downstream_speed+2)=float(Dspeed);
      //input 6
      *(checkin_occ+2)=float(Checkin);
      //input 7
      *(queue_occ+2)=float(Qocc);
      float flow_rate=flcMeterRate(local_occ, local_speed, local_flow,
      downstream_vc, downstream_speed, checkin_occ, queue_occ);
      return (1/((flow-flow_rate)*(flow-flow_rate)));
}
int cLength(int precision, double rangeStart, double rangeEnd)
{
      int length=0;
      double total=(rangeEnd-rangeStart)*pow(10.0,precision);
      while(total>pow(2.0,length))
            {length++;}
      return(length);
}

double Lspeed(int* chromosome, int speedLength, double* domain)
{
```

```
	double m=0.0;
	for(int i=0; i<speedLength; i++)
	{m+=chromosome[speedLength-i-1]*pow(2.0,i);}
	double x=domain[0]+m*(domain[1]-domain[0])/(pow(2.0,speedLength)-
	1.0);
	return x;
}

double Mspeed(int* chromosome, int speedLength, double* domain)
{
	double m=0.0;
	for(int i=0; i<speedLength; i++)
	{m+=chromosome[2*speedLength-i-1]*pow(2.0,i);}
	double x=domain[0]+m*(domain[1]-domain[0])/(pow(2.0,speedLength)-
	1.0);
	return x;
}

double Hspeed(int* chromosome, int speedLength, double* domain)
{
	double m=0.0;
	for(int i=0; i<speedLength; i++)
	{m+=chromosome[3*speedLength-i-1]*pow(2.0,i);}
	double x=domain[0]+m*(domain[1]-domain[0])/(pow(2.0,speedLength)-
	1.0);
	return x;
}

double Lflow(int* chromosome, int speedLength, int flowLength, double*
domain)
{
	double m=0.0;
	int length=3*speedLength+flowLength;
	for(int i=0; i<flowLength; i++)
	{m+=chromosome[length-i-1]*pow(2.0,i);}
	double x=domain[2]+m*(domain[3]-domain[2])/(pow(2.0,flowLength)-
	1.0);
	return x;
}

double Mflow(int* chromosome, int speedLength, int flowLength, double*
domain)
{
	double m=0.0;
	int length=3*speedLength+2*flowLength;
	for(int i=0; i<flowLength; i++)
	{m+=chromosome[length-i-1]*pow(2.0,i);}
	double x=domain[2]+m*(domain[3]-domain[2])/(pow(2.0,flowLength)-
	1.0);
	return x;
}

double Hflow(int* chromosome, int speedLength, int flowLength, double*
domain)
{
	double m=0.0;
	int length=3*speedLength+3*flowLength;
```

```
	for(int i=0; i<flowLength; i++)
	{m+=chromosome[length-i-1]*pow(2.0,i);}
	double x=domain[2]+m*(domain[3]-domain[2])/(pow(2.0,flowLength)-
	1.0);
	return x;
}

double Locc(int* chromosome, int speedLength, int flowLength, int
occLength, double* domain)
{
	double m=0.0;
	int length=3*speedLength+3*flowLength+occLength;
	for(int i=0; i<occLength; i++)
	{m+=chromosome[length-i-1]*pow(2.0,i);}
	double x=domain[4]+m*(domain[5]-domain[4])/(pow(2.0,occLength)-
	1.0);
	return x;
}

double Mocc(int* chromosome, int speedLength, int flowLength, int
occLength, double* domain)
{
	double m=0.0;
	int length=3*speedLength+3*flowLength+2*occLength;
	for(int i=0; i<occLength; i++)
	{m+=chromosome[length-i-1]*pow(2.0,i);}
	double x=domain[4]+m*(domain[5]-domain[4])/(pow(2.0,occLength)-
	1.0);
	return x;
}

double Hocc(int* chromosome, int speedLength, int flowLength, int
occLength, double* domain)
{
	double m=0.0;
	int length=3*speedLength+3*flowLength+3*occLength;
	for(int i=0; i<occLength; i++)
	{m+=chromosome[length-i-1]*pow(2.0,i);}
	double x=domain[4]+m*(domain[5]-domain[4])/(pow(2.0,occLength)-
	1.0);
	return x;
}

double vc(int* chromosome, int speedLength, int flowLength, int
occLength, int vcLength, double* domain)
{
	double m=0.0;
	int length=3*speedLength+3*flowLength+3*occLength+vcLength;
	for(int i=0; i<vcLength; i++)
	{m+=chromosome[length-i-1]*pow(2.0,i);}
	double x=domain[6]+m*(domain[7]-domain[6])/(pow(2.0,vcLength)-
	1.0);
	return x;
}

double dspeed(int* chromosome, int speedLength, int flowLength, int
occLength, int vcLength, int dspeedLength, double* domain)
```

```
{
double m=0.0;
int  length=3*speedLength+3*flowLength+3*occLength+vcLength+dspeedLength;
for(int i=0; i<dspeedLength; i++)
{m+=chromosome[length-i-1]*pow(2.0,i);}
double x=domain[8]+m*(domain[9]-domain[8])/(pow(2.0,dspeedLength)-1.0);
      return x;
}

double checkin(int* chromosome, int speedLength, int flowLength, int
occLength, int vcLength, int dspeedLength, int checkinLength, double*
domain)
{
double m=0.0;
int
length=3*speedLength+3*flowLength+3*occLength+vcLength+dspeedLength+
checkinLength;
for(int i=0; i<checkinLength; i++)
{m+=chromosome[length-i-1]*pow(2.0,i);}
double x=domain[10]+m*(domain[11]-domain[10])/(pow(2.0,checkinLength)-
1.0);
      return x;
}

double qocc(int* chromosome, int speedLength, int flowLength, int
occLength, int vcLength, int dspeedLength, int checkinLength, int
qoccLength, double* domain)
{
      double m=0.0;
int
length=3*speedLength+3*flowLength+3*occLength+vcLength+dspeedLength+che
ckinLength+qoccLength;
for(int i=0; i<qoccLength; i++)
{m+=chromosome[length-i-1]*pow(2.0,i);}
double  x=domain[12]+m*(domain[13]-domain[12])/(pow(2.0,qoccLength)-1.0);
      return x;
}


void setup(int**farm, int size, int length,  int speedLength, int
flowLength, int occLength, int vcLength, int dspeedLength, int
checkinLength, int qoccLength, double* domain)
{
      time_t t;
      srand((unsigned) time(&t));
      for(int i=0; i<size; i++)
      {

            int ispop=0;

            while(!ispop)
            {

                  for(int j=0; j<3*speedLength; j++)
                  { farm[i][j]=rand()%2;}
                  double Ls=Lspeed(farm[i], speedLength, domain);
                  double Ms=Mspeed(farm[i], speedLength, domain);
```

```
                  double Hs=Hspeed(farm[i], speedLength, domain);
                  if((100-Ls-Ms-Hs)>=0)
                  {ispop=1;}
            }
            ispop=0;
            while(!ispop)
            {
      for(int j=(3*speedLength-1); j<(3*speedLength+3*flowLength); j++)
      { farm[i][j]=rand()%2;}
        double Lf=Lflow(farm[i], speedLength, flowLength, domain);
        double Mf=Mflow(farm[i], speedLength, flowLength, domain);
        double Hf=Hflow(farm[i], speedLength, flowLength, domain);
                  if((4000-Lf-Mf-Hf)>=0)
                  {ispop=1;}
            }
            ispop=0;
            while(!ispop)
            {
                  for(int j=(3*speedLength+3*flowLength-1);
                  j<(3*speedLength+3*flowLength+3*occLength); j++)
                  {farm[i][j]=rand()%2;}
                  double Lo=Locc(farm[i], speedLength, flowLength,
                  occLength, domain);
                  double Mo=Mocc(farm[i], speedLength, flowLength,
                  occLength, domain);
                  double Ho=Hocc(farm[i], speedLength, flowLength,
                  occLength, domain);
                  if((30-Lo-Mo-Ho)>=0)
                  {ispop=1;}
            }
            ispop=0;
            for(int j=(3*speedLength+3*flowLength+3*occLength-
            1);j<length;j++)
            {farm[i][j]=rand()%2;}
      }
}
void printFarm(int**farm, int length, int size)
{
      for(int i=0; i<size; i++)
      {
            std::cout<<"\n";
            for(int j=0; j<length; j++)
            {std::cout<<farm[i][j];}
      }
}

double fitnessValue(double (*f)(float*, float*, float*, float* ,
float* , float* , float* ,double,double, double,double, double,double,
double,double, double,double, double,double,double, double),float*
local_occ, float* local_speed, float* local_flow, float* downstream_vc,
float* downstream_speed, float* checkin_occ, float* queue_occ ,int*
chromosome, int length, double* domain, int speedLength, int flowLength,
int occLength, int vcLength, int dspeedLength, int checkinLength, int
qoccLength, double flow)
{
 double Ls=Lspeed(chromosome, speedLength, domain);
 double Ms=Mspeed(chromosome, speedLength, domain);
```

```
 double Hs=Hspeed(chromosome, speedLength, domain);
 double Lf=Lflow(chromosome, speedLength, flowLength, domain);
 double Mf=Mflow(chromosome, speedLength, flowLength, domain);
 double Hf=Hflow(chromosome, speedLength, flowLength, domain);
 double Lo=Locc(chromosome, speedLength, flowLength, occLength, domain);
 double Mo=Mocc(chromosome, speedLength, flowLength, occLength, domain);
 double Ho=Hocc(chromosome, speedLength, flowLength, occLength, domain);
 double Vc=vc(chromosome, speedLength, flowLength, occLength, vcLength,
domain);
 double Ds=dspeed(chromosome, speedLength, flowLength, occLength,
vcLength, dspeedLength, domain);
 double Check=checkin(chromosome, speedLength, flowLength, occLength,
vcLength, dspeedLength, checkinLength, domain);
 double Qo=qocc(chromosome, speedLength, flowLength, occLength,
vcLength, dspeedLength, checkinLength, qoccLength, domain);


      double result;
      result=f(local_occ,local_speed,local_flow,downstream_vc,
      downstream_speed, checkin_occ,
      queue_occ,flow,Ls,Ms,Hs,Lf,Mf,Hf,Lo,Mo,Ho,Vc,Ds,Check,Qo);
      return(result);
}
void roulette(float* local_occ, float* local_speed, float* local_flow,
float* downstream_vc, float* downstream_speed, float* checkin_occ,
float* queue_occ ,int** farm, int length, int size, double* domain, int
speedLength, int flowLength, int occLength, int vcLength, int
dspeedLength, int checkinLength, int qoccLength,double flow)
{
      int i, j;
      double* fitnessVector=NULL;
      fitnessVector = new double[size];
      for(i=0; i<size; i++)
      {fitnessVector[i]=fitnessValue(f,local_occ, local_speed,
      local_flow,downstream_vc, downstream_speed, checkin_occ,
      queue_occ ,farm[i],length,domain, speedLength, flowLength,
      occLength, vcLength, dspeedLength, checkinLength,
      qoccLength,flow);}

      double totalFitness=0.0;
      for(i=0; i<size; i++)
      {totalFitness += fitnessVector[i];}

      double* probabilityVector=NULL;
      probabilityVector = new double[size];
      for(i=0; i<size; i++)
      {probabilityVector[i]=fitnessVector[i]/totalFitness;}

      double cumulativeProb = 0.0;
      double* cum_prob_Vector = NULL;
      cum_prob_Vector = new double [size];
      for(i=0; i<size; i++)
      {cumulativeProb += probabilityVector[i];
       cum_prob_Vector[i] = cumulativeProb;}

      double* randomVector = NULL;
      randomVector = new double[size];
```

```
	time_t t;
	srand((unsigned) time(&t));
	for(i=0; i<size; i++)
	randomVector[i] = rand()/double(RAND_MAX);

	int count;
	int** newFarm = NULL;
	newFarm = new int*[size];
	for(i=0; i <size; i++)
	newFarm[i]= new int[length];
	for( i=0; i<size; i++)
	{
		count=0;
		while(randomVector[i]>cum_prob_Vector[count]) count++;
		for(j=0; j<length; j++)
		{newFarm[i][j] =farm[count][j];}
	}

	for(i=0; i<size; i++)
	for(j=0; j<length; j++)
	farm[i][j] = newFarm[i][j];

	delete[] fitnessVector;
	delete[] probabilityVector;
	delete[] cum_prob_Vector;
	delete[] randomVector;
	for(i=0; i<size; i++)
	delete[] newFarm[i];
	delete[] newFarm;
}

void crossing(int** farm, int size, int length,double* domain, int
speedLength, int flowLength, int occLength)
{
	int i, j, k, m;
	int count = 0;
	int* chosen = NULL;
	int ispop=0;
	chosen = new int[size];

	double* randomVector = NULL;
	randomVector= new double [size];
	time_t t;
	srand((unsigned) time(&t));
	for(i=0; i<size; i++)
	randomVector[i] = rand()/double(RAND_MAX);

	for(i=0; i<size; i++)
	{
		if(randomVector[i]<0.4)
		{chosen[count]=i;
		 count++;}
	}

	if((count%2 != 0) || (count==1))
	{
		int index=0;
```

```
            while(randomVector[index]<0.4) index++;
            count++;
            chosen[count-1] = index;
      }

      int** temp = NULL;
      temp = new int*[2];
      for(i=0; i<2; i++)
      {temp[i] = new int[length];}

      for(i=0;i<count;i=i+2)
      {
            while(!ispop)
            {
                  for(j=0;j<length;j++)
                  {temp[0][j]=farm[chosen[i]][j];
                   temp[1][j]=farm[chosen[i+1]][j];}
                   srand((unsigned) time(&t));
                   int position=rand()%length;
                   for(k=position; k<length; k++)
                   {temp[0][k]=farm[chosen[i+1]][k];
                   temp[1][k]=farm[chosen[i]][k];}

double Ls0=Lspeed(temp[0], speedLength, domain);
double Ms0=Mspeed(temp[0], speedLength, domain);
double Hs0=Hspeed(temp[0], speedLength, domain);
double Lf0=Lflow(temp[0], speedLength, flowLength, domain);
double Mf0=Mflow(temp[0], speedLength, flowLength, domain);
double Hf0=Hflow(temp[0], speedLength, flowLength, domain);
double Lo0=Locc(temp[0], speedLength, flowLength, occLength, domain);
double Mo0=Mocc(temp[0], speedLength, flowLength, occLength, domain);
double Ho0=Hocc(temp[0], speedLength, flowLength, occLength, domain);
double Ls1=Lspeed(temp[1], speedLength, domain);
double Ms1=Mspeed(temp[1], speedLength, domain);
double Hs1=Hspeed(temp[1], speedLength, domain);
double Lf1=Lflow(temp[1], speedLength, flowLength, domain);
double Mf1=Mflow(temp[1], speedLength, flowLength, domain);
double Hf1=Hflow(temp[1], speedLength, flowLength, domain);
double Lo1=Locc(temp[1], speedLength, flowLength, occLength, domain);
double Mo1=Mocc(temp[1], speedLength, flowLength, occLength, domain);
double Ho1=Hocc(temp[1], speedLength, flowLength, occLength, domain);

      if ((100-Ls0-Ms0-Hs0)>=0&&(4000-Lf0-Mf0-Hf0)>=0&&(30-Lo0-Mo0-
      Ho0)>=0&&(100-Ls1-Ms1-Hs1)>=0&&(4000-Lf1-Mf1-Hf1)>=0&&(30-Lo1-
      Mo1-Ho1)>=0)
                  {ispop=1;}
            }
            ispop=0;
            for(m=0; m<length; m++)
            {farm[chosen[i]][m]=temp[0][m];
             farm[chosen[i+1]][m]=temp[1][m];}
      }

      delete[] chosen;
      delete[] randomVector;
      for(i=0; i<2; i++)
      delete[] temp[i];
```

```
	delete[] temp;
}



void mutate(int** farm, int size, int length,double* domain, int
speedLength, int flowLength, int occLength)
{
	int i,j;
	int totalbits=size*length;
	double* randomVector= NULL;
	int** temp = NULL;
	temp = new int*[1];
	for(i=0; i<1; i++)
	{temp[i] = new int[length];}
	randomVector= new double[totalbits];
	time_t t;
	srand((unsigned) time(&t));
	for(i=0; i<totalbits; i++)
	randomVector[i]=rand()/double(RAND_MAX);

	int a,b;
	for(i=0; i<totalbits; i++)
	{
			if(randomVector[i]<0.01)
			{
				if(i>=length)
				{a=i/length; b=i%length;}
				else
				{a=0; b=i;}
				for(j=0;j<length;j++)
				{temp[0][j]=farm[a][j];}
				if(temp[0][b]==0)
				temp[0][b]=1;
				else
				temp[0][b]=0;

	double Ls0=Lspeed(temp[0], speedLength, domain);
	double Ms0=Mspeed(temp[0], speedLength, domain);
	double Hs0=Hspeed(temp[0], speedLength, domain);
	double Lf0=Lflow(temp[0], speedLength, flowLength, domain);
	double Mf0=Mflow(temp[0], speedLength, flowLength, domain);
	double Hf0=Hflow(temp[0], speedLength, flowLength, domain);
	double Lo0=Locc(temp[0], speedLength, flowLength, occLength,
	domain);
	double Mo0=Mocc(temp[0], speedLength, flowLength, occLength,
	domain);
	double Ho0=Hocc(temp[0], speedLength, flowLength, occLength,
	domain);
if ((100-Ls0-Ms0-Hs0)>=0&&(4000-Lf0-Mf0-Hf0)>=0&&(30-Lo0-Mo0-Ho0)>=0)
				{
				  for(j=0; j<length; j++)
				  {farm[a][j]=temp[0][j];}
				}
			}
		}
```

```
	delete[] randomVector;
	delete[] temp[0];
	delete[] temp;
}

void printResult(float* local_occ, float* local_speed, float*
local_flow, float* downstream_vc, float* downstream_speed, float*
checkin_occ, float* queue_occ ,int** farm, int length, int size,
double* domain, int speedLength, int flowLength, int occLength, int
vcLength, int dspeedLength, int checkinLength, int qoccLength, int
iterations, double flow)
{
	int i;
	double* fitnessVector=new double[size];

	for(i=0; i<size; i++)
fitnessVector[i]= fitnessValue(f,local_occ,local_speed, local_flow,
downstream_vc, downstream_speed, checkin_occ, queue_occ ,farm[i],length,
domain, speedLength, flowLength, occLength, vcLength, dspeedLength,
checkinLength, qoccLength,flow);

	int pos=0;
	double max=fitnessVector[0];
	for(i=1; i<size; i++)
	{
		if(fitnessVector[i]>max)
		{max= fitnessVector[i];
		 pos=i;}
	}
	double Ls=Lspeed(farm[pos], speedLength, domain);
	double Ms=Mspeed(farm[pos], speedLength, domain);
	double Hs=Hspeed(farm[pos], speedLength, domain);
	double Lf=Lflow(farm[pos], speedLength, flowLength, domain);
	double Mf=Mflow(farm[pos], speedLength, flowLength, domain);
	double Hf=Hflow(farm[pos], speedLength, flowLength, domain);
	double Lo=Locc(farm[pos], speedLength, flowLength, occLength,
	domain);
	double Mo=Mocc(farm[pos], speedLength, flowLength, occLength,
	domain);
	double Ho=Hocc(farm[pos], speedLength, flowLength, occLength,
	domain);
	double Vc=vc(farm[pos], speedLength, flowLength, occLength,
	vcLength, domain);
	double Ds=dspeed(farm[pos], speedLength, flowLength, occLength,
	vcLength, dspeedLength, domain);
	double Check=checkin(farm[pos], speedLength, flowLength,
	occLength, vcLength, dspeedLength, checkinLength, domain);
	double Qo=qocc(farm[pos], speedLength, flowLength, occLength,
	vcLength, dspeedLength, checkinLength, qoccLength, domain);

std::cout<<"\n\n After"<<iterations<<"iterations the fitness are: \n";
	for(i=0; i<size; i++)
	{
	std::cout<<"\n fitness of chromosome"<<i<<":"<<fitnessVector[i];
	}
	std::cout<<"\n\n The maximum fitness: "<< max;
	std::cout<<"\n Ls:"<<Ls;
```

```
        std::cout<<"\n Ms:"<<Ms+Ls;
        std::cout<<"\n Hs:"<<Hs+Ms+Ls;
        std::cout<<"\n Lf:"<<Lf;
        std::cout<<"\n Mf:"<<Mf+Lf;
        std::cout<<"\n Hf:"<<Hf+Mf+Lf;
        std::cout<<"\n Lo:"<<Lo;
        std::cout<<"\n Mo:"<<Mo+Lo;
        std::cout<<"\n Ho:"<<Ho+Mo+Lo;
        std::cout<<"\n Vc:"<<Vc;
        std::cout<<"\n Ds:"<<Ds;
        std::cout<<"\n Check:"<<Check;
        std::cout<<"\n Qo:"<<Qo;

        float local_occ[7], local_speed[7], local_flow[7],
        downstream_vc[3], downstream_speed[3], checkin_occ[3],
        queue_occ[3];
        //input 1
        local_speed[0] = 0;
        local_speed[1]=21.5f; local_speed[2]=float(Ls);
        local_speed[3]=21.5f; local_speed[4]=float(Ms+Ls);
        local_speed[5]=21.5f; local_speed[6]=float(Hs+Ms+Hs);
        //input 2
        local_flow[0] = 0;
        local_flow[1]=850; local_flow[2]=float(Lf);
        local_flow[3]=850; local_flow[4]=float(Mf+Lf);
        local_flow[5]=850; local_flow[6]=float(Hf+Mf+Lf);
        //input 3
        local_occ[0] = 0;
        local_occ[1]=6.4f; local_occ[2]=float(Lo);
        local_occ[3]=6.4f; local_occ[4]=float(Mo+Lo);
        local_occ[5]=6.4f; local_occ[6]=float(Ho+Mo+Lo);
        //input 4
        downstream_vc[0] = 0;
        downstream_vc[1]=6.5f; downstream_vc[2]=float(Vc);
        //input 5
        downstream_speed[0] = 0;
        downstream_speed[1]=-0.25; downstream_speed[2]=float(Ds);
        //input 6
        checkin_occ[0] = 0;
        checkin_occ[1]=0.4f; checkin_occ[2]=float(Check);
        //input 7
        queue_occ[0] = 0;
        queue_occ[1]=0.4f; queue_occ[2]=float(Qo);
        float flow_rate=flcMeterRate(local_occ, local_speed, local_flow,
        downstream_vc, downstream_speed, checkin_occ, queue_occ);

        std::cout<<"\n flow_rate:"<<flow_rate;
        std::cout<<"\n";

        delete[] fitnessVector;
}


void GA(float* local_occ, float* local_speed, float* local_flow, float*
downstream_vc, float* downstream_speed, float* checkin_occ, float*
queue_occ, double flow, double* L_speed,double* M_speed,double*
H_speed,double* L_flow,double* M_flow,double* H_flow,double*
```

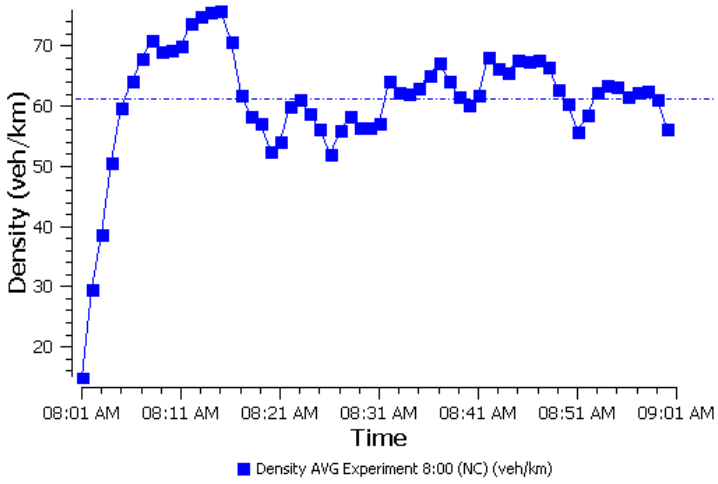
```
L_occ,double* M_occ,double* H_occ,double* V_c,double* D_speed, double*
Check_in,double* Q_occ)
{
      int size=50;
      int precision=1;
      int iter=400;
      double domain[14];
      double
      speed1,speed2,flow1,flow2,occ1,occ2,vc1,vc2,dspeed1,dspeed2,check
      in1,checkin2,qocc1,qocc2;
      speed1=0;speed2=100;
      flow1=0;flow2=4000;
      occ1=0;occ2=30;
      vc1=0;vc2=1;
      dspeed1=0;dspeed2=100;
      checkin1=0;checkin2=50;
      qocc1=0;qocc2=50;

      domain[0]=speed1; domain[1]=speed2;
      domain[2]=flow1; domain[3]=flow2;
      domain[4]=occ1; domain[5]=occ2;
      domain[6]=vc1; domain[7]=vc2;
      domain[8]=dspeed1; domain[9]=dspeed2;
      domain[10]=checkin1; domain[11]=checkin2;
      domain[12]=qocc1; domain[13]=qocc2;

      int speedLength=cLength(precision, domain[0], domain[1]);
      int flowLength=cLength(precision, domain[2], domain[3]);
      int occLength=cLength(precision, domain[4], domain[5]);
      int vcLength=cLength(precision, domain[6], domain[7]);
      int dspeedLength=cLength(precision, domain[8], domain[9]);
      int checkinLength=cLength(precision, domain[10], domain[11]);
      int qoccLength=cLength(precision, domain[12], domain[13]);
      int length=3*speedLength+3*flowLength+3*occLength+vcLength+
      dspeedLength+checkinLength+qoccLength;
      //std::cout<<"\n the chromosone length is: "<<length;

      int** farm=NULL;
      farm=new int*[size];
      for(int i=0; i<size; i++)
      {farm[i]= new int[length];}
      setup(farm, size, length, speedLength, flowLength, occLength,
      vcLength, dspeedLength,checkinLength,qoccLength,domain);
      //std::cout<<"\n\n The initial farm: \n";
      //printFarm(farm,length,size);
      //printResult(local_occ, local_speed, local_flow,
      downstream_vc,downstream_speed, checkin_occ, queue_occ ,farm,
      length, size, domain, speedLength, flowLength, occLength,
      vcLength, dspeedLength, checkinLength, qoccLength, iter, flow);
      std::cout<<std::endl;
            for(int t=0; t<iter; t++)
      {
            roulette(local_occ, local_speed, local_flow,
            downstream_vc,downstream_speed, checkin_occ,
            queue_occ ,farm,length,size,domain,speedLength, flowLength,
            occLength, vcLength, dspeedLength, checkinLength,
            qoccLength, flow);
```

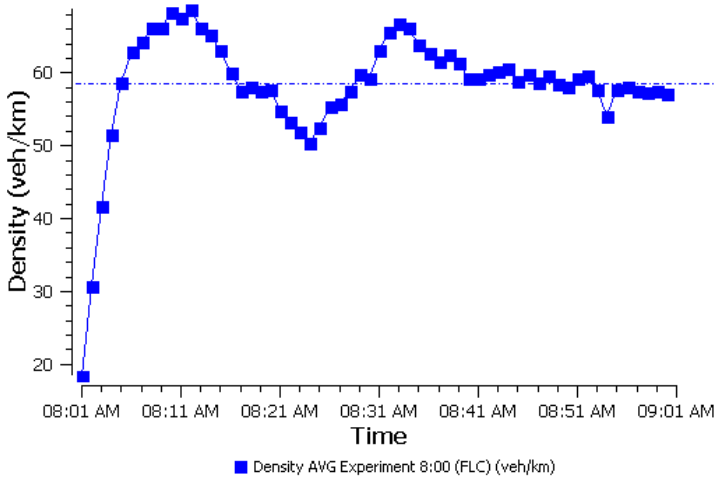
```
      crossing(farm,size,length, domain,speedLength,flowLength,
      occLength);
      roulette(local_occ, local_speed, local_flow,
      downstream_vc,downstream_speed, checkin_occ,
      queue_occ ,farm,length,size,domain,speedLength, flowLength,
      occLength, vcLength, dspeedLength, checkinLength,
      qoccLength, flow);
      mutate(farm,size,length,domain,speedLength,flowLength,
      occLength);
}
//printResult(local_occ, local_speed, local_flow,
downstream_vc,downstream_speed, checkin_occ, queue_occ ,farm,
length, size, domain, speedLength, flowLength, occLength,
vcLength, dspeedLength, checkinLength, qoccLength, iter, flow);

int i;
double* fitnessVector=new double[size];

for(i=0; i<size; i++)
fitnessVector[i]= fitnessValue(f,local_occ,local_speed,
local_flow, downstream_vc, downstream_speed, checkin_occ,
queue_occ ,farm[i],length, domain, speedLength, flowLength,
occLength, vcLength, dspeedLength, checkinLength,
qoccLength,flow);

int pos=0;
double max=fitnessVector[0];
for(i=1; i<size; i++)
{
      if(fitnessVector[i]>max)
      {max= fitnessVector[i];
       pos=i;}
}
double Ls=Lspeed(farm[pos], speedLength, domain);
double Ms=Mspeed(farm[pos], speedLength, domain);
double Hs=Hspeed(farm[pos], speedLength, domain);
double Lf=Lflow(farm[pos], speedLength, flowLength, domain);
double Mf=Mflow(farm[pos], speedLength, flowLength, domain);
double Hf=Hflow(farm[pos], speedLength, flowLength, domain);
double Lo=Locc(farm[pos], speedLength, flowLength, occLength,
domain);
double Mo=Mocc(farm[pos], speedLength, flowLength, occLength,
domain);
double Ho=Hocc(farm[pos], speedLength, flowLength, occLength,
domain);
double Vc=vc(farm[pos], speedLength, flowLength, occLength,
vcLength, domain);
double Ds=dspeed(farm[pos], speedLength, flowLength, occLength,
vcLength, dspeedLength, domain);
double Check=checkin(farm[pos], speedLength, flowLength,
occLength, vcLength, dspeedLength, checkinLength, domain);
double Qo=qocc(farm[pos], speedLength, flowLength, occLength,
vcLength, dspeedLength, checkinLength, qoccLength, domain);

*L_speed=Ls;
*M_speed=Ms+Ls;
*H_speed=Hs+Ms+Ls;
```

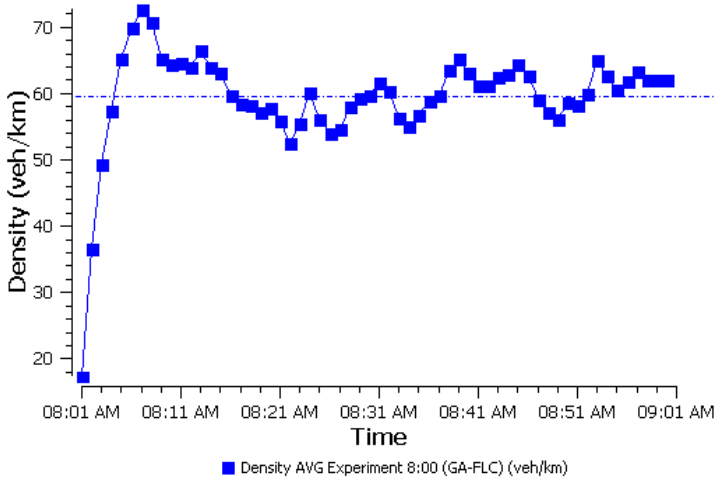
```
        *L_flow=Lf;
        *M_flow=Lf+Mf;
        *H_flow=Lf+Mf+Hf;
        *L_occ=Lo;
        *M_occ=Lo+Mo;
        *H_occ=Lo+Mo+Ho;
        *V_c=Vc;
        *D_speed=Ds;
        *Check_in=Check;
        *Q_occ=Qo;



        delete[] fitnessVector;
        for(int k=0; k<size; k++)
        delete[] farm[k];
        delete[] farm;
}
```

Appendix C: Simulation Results-The change of system flow density

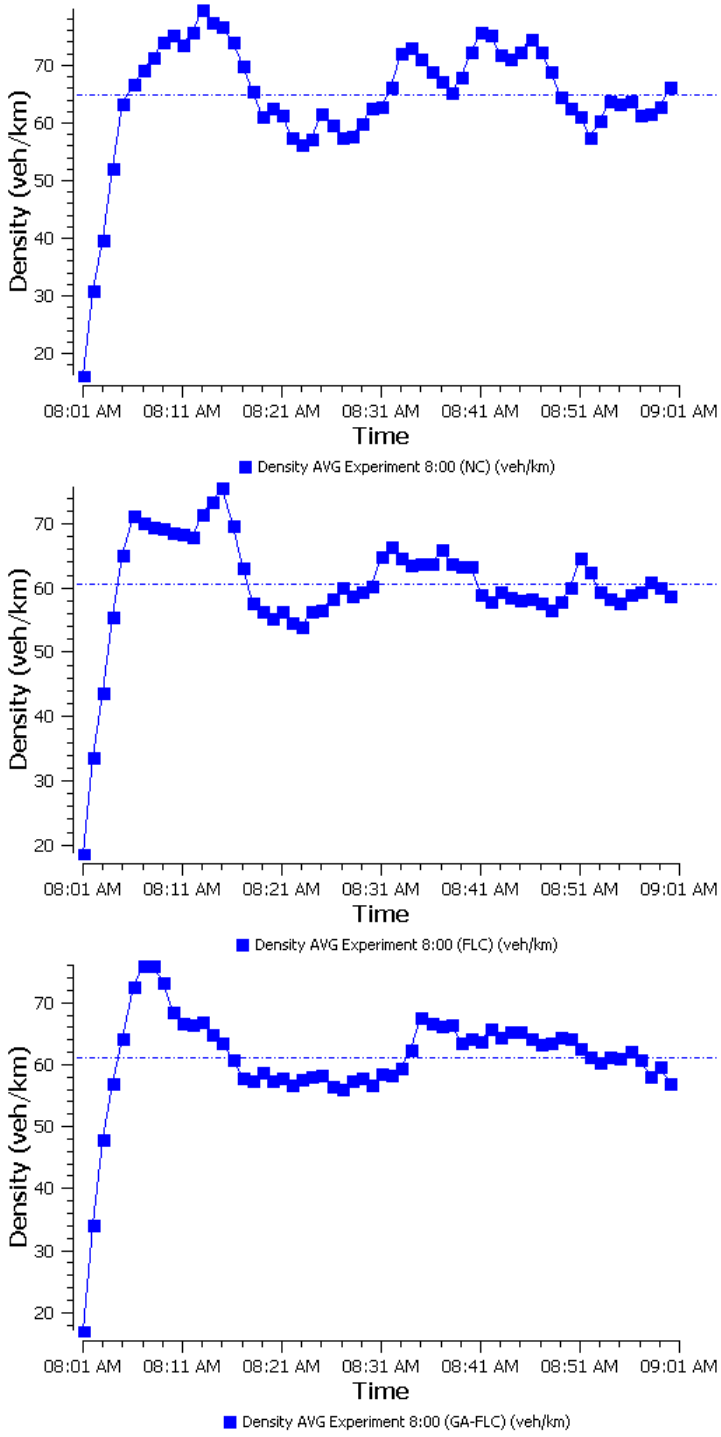
Traffic demand: 3000-1600 vehs/h



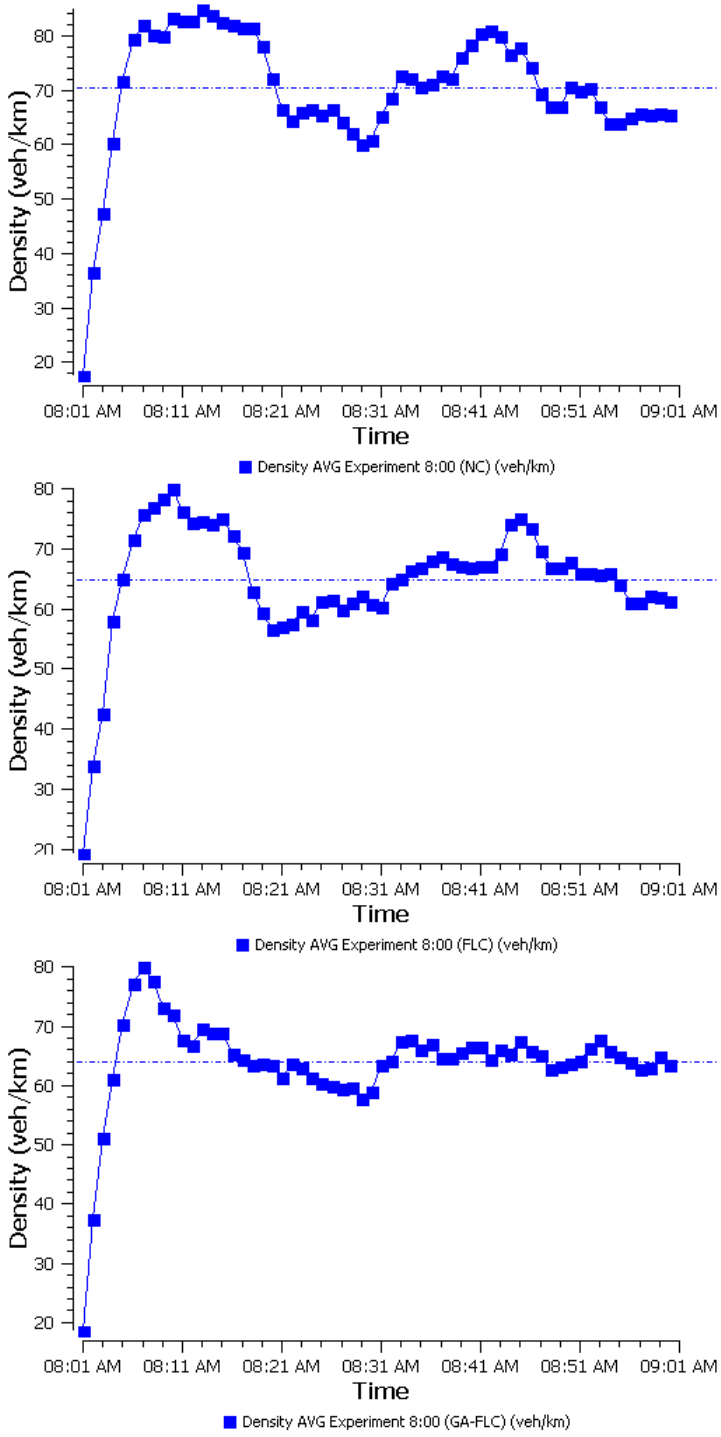




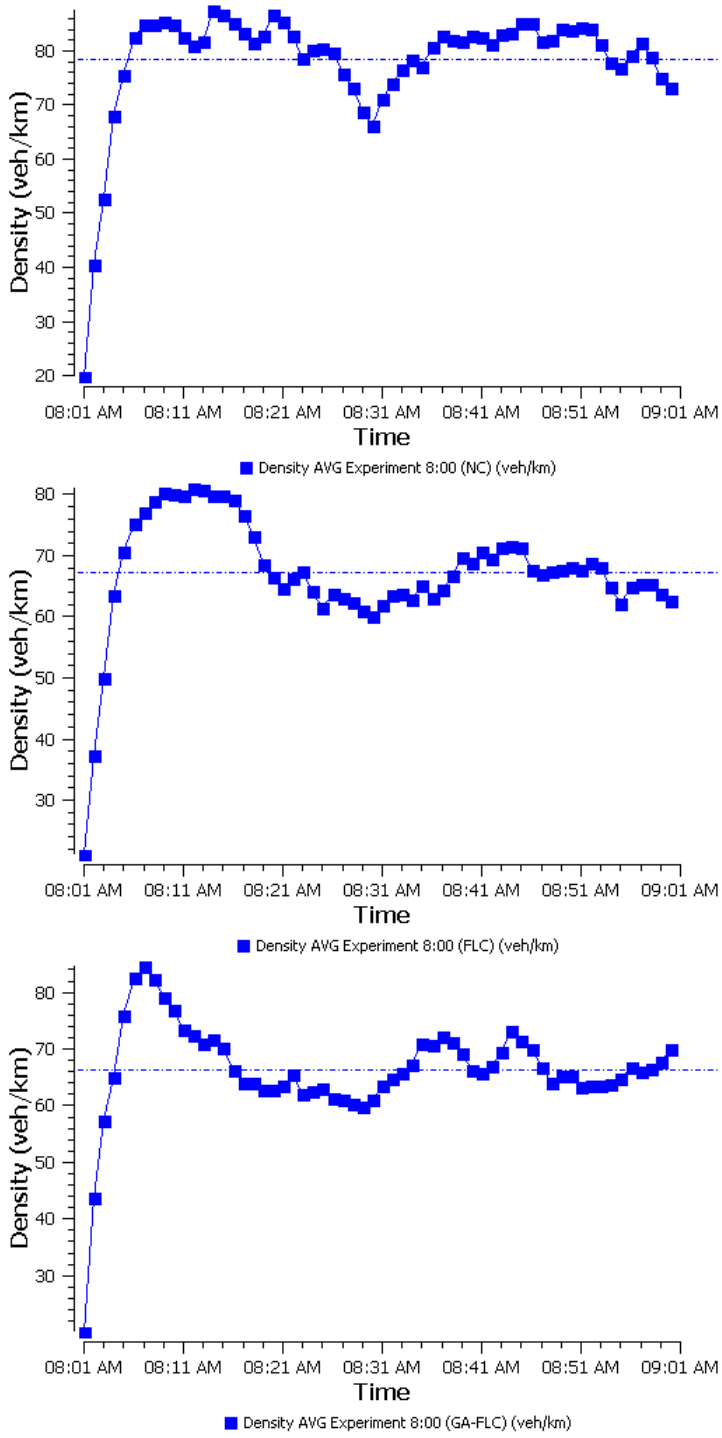
Total demand: 3200-1600 vehs/h



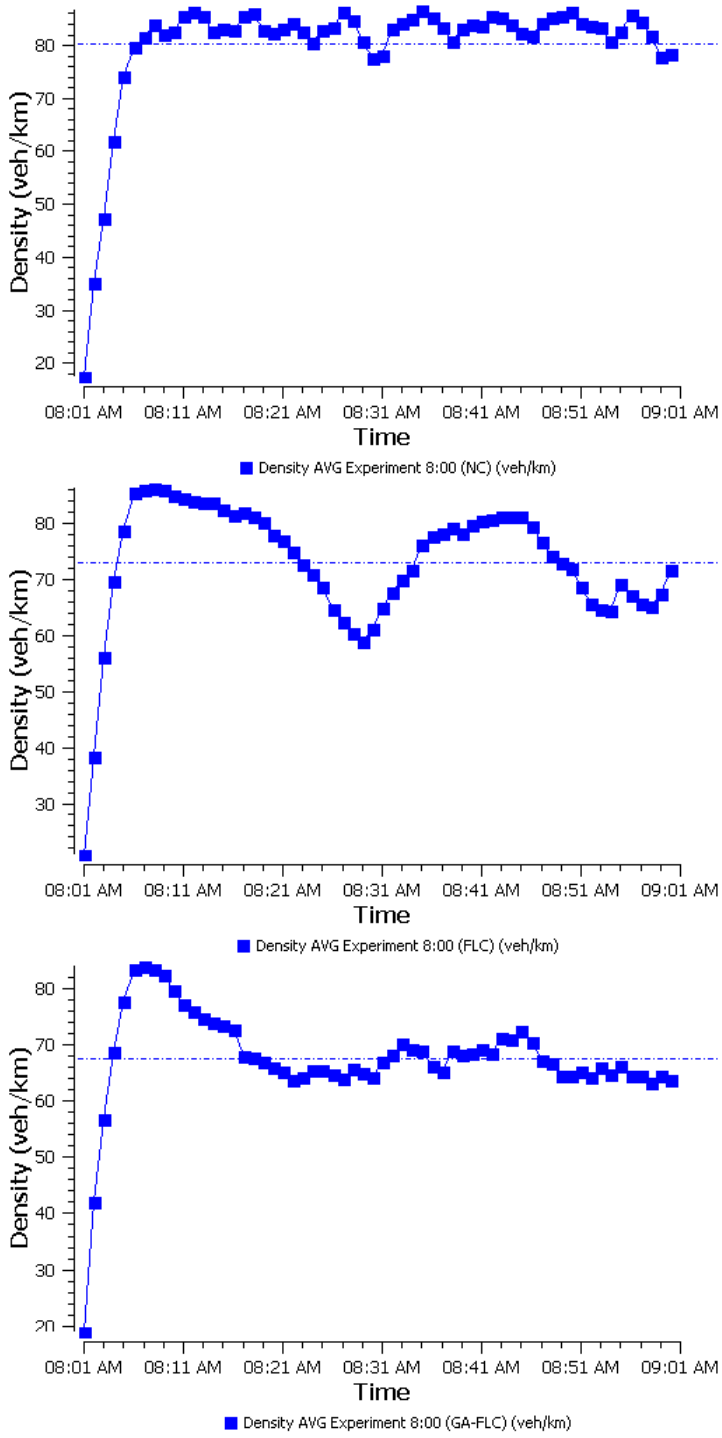
Total demand: 3400-1600 vehs/h



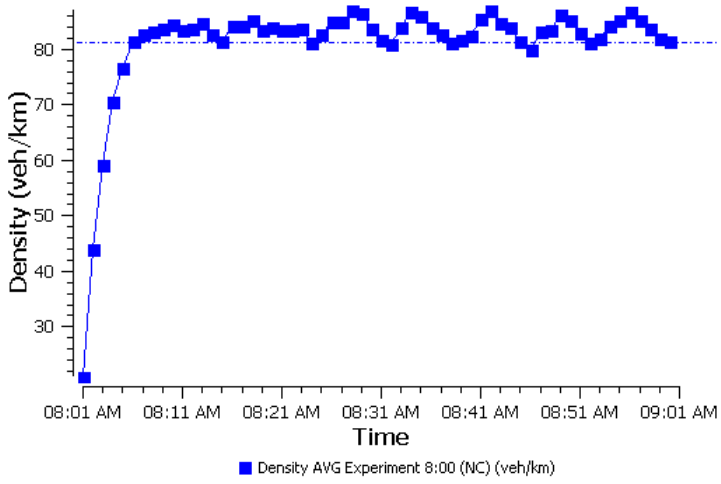
Total demand: 3600-1600 vehs/h

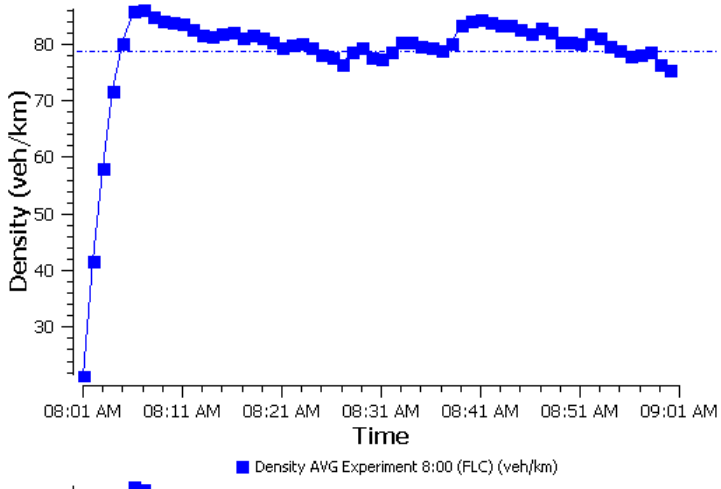


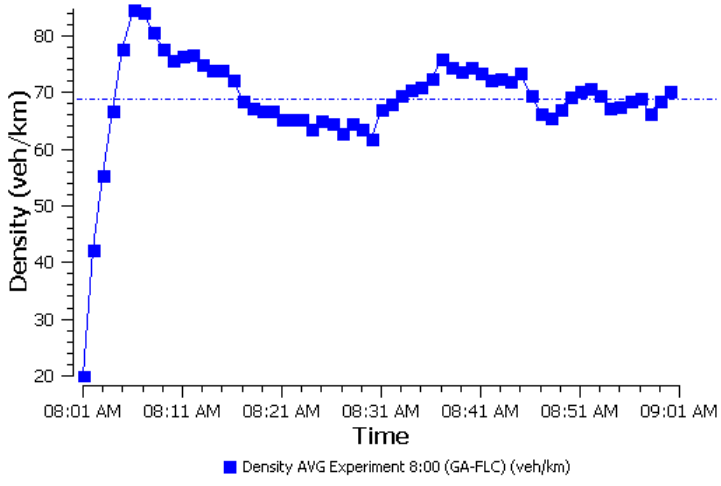
Total demand: 3800-1600 vehs/h



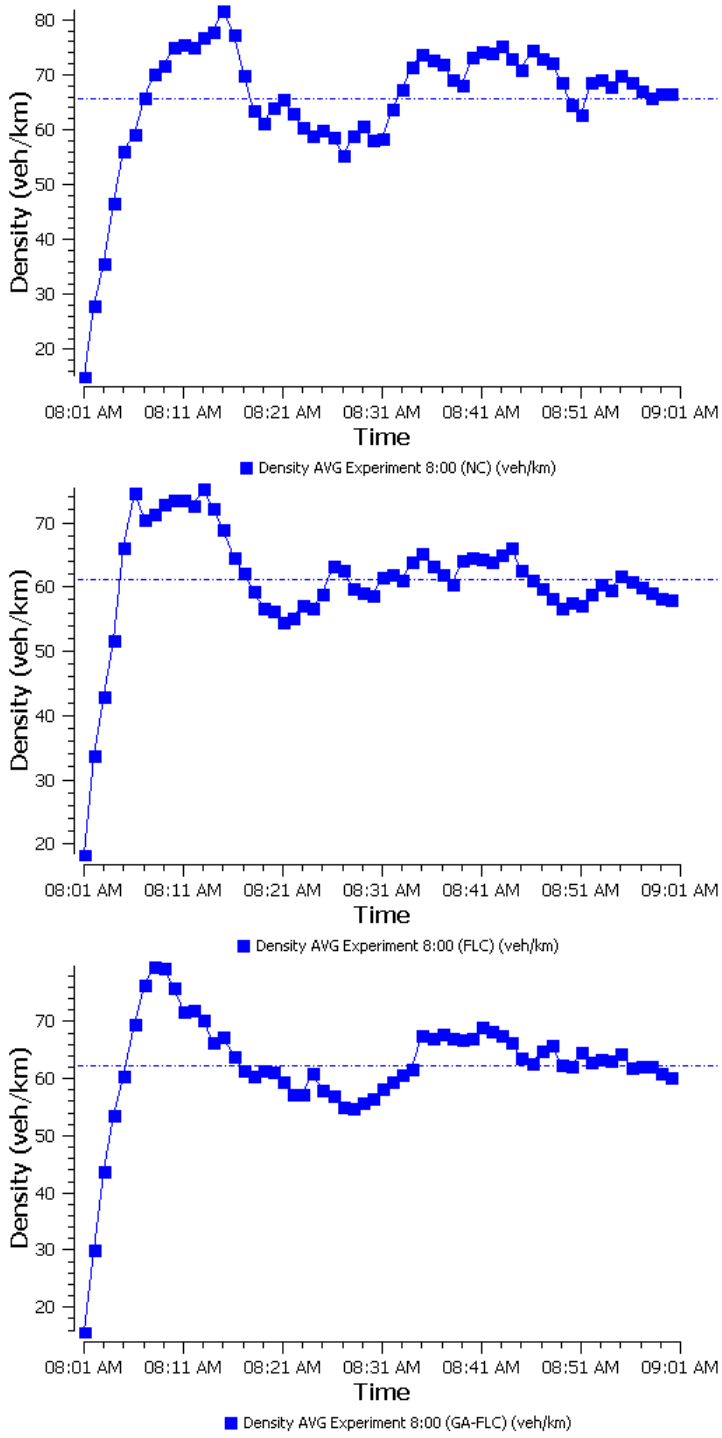
Total demand: 4000-1600 vehs/h



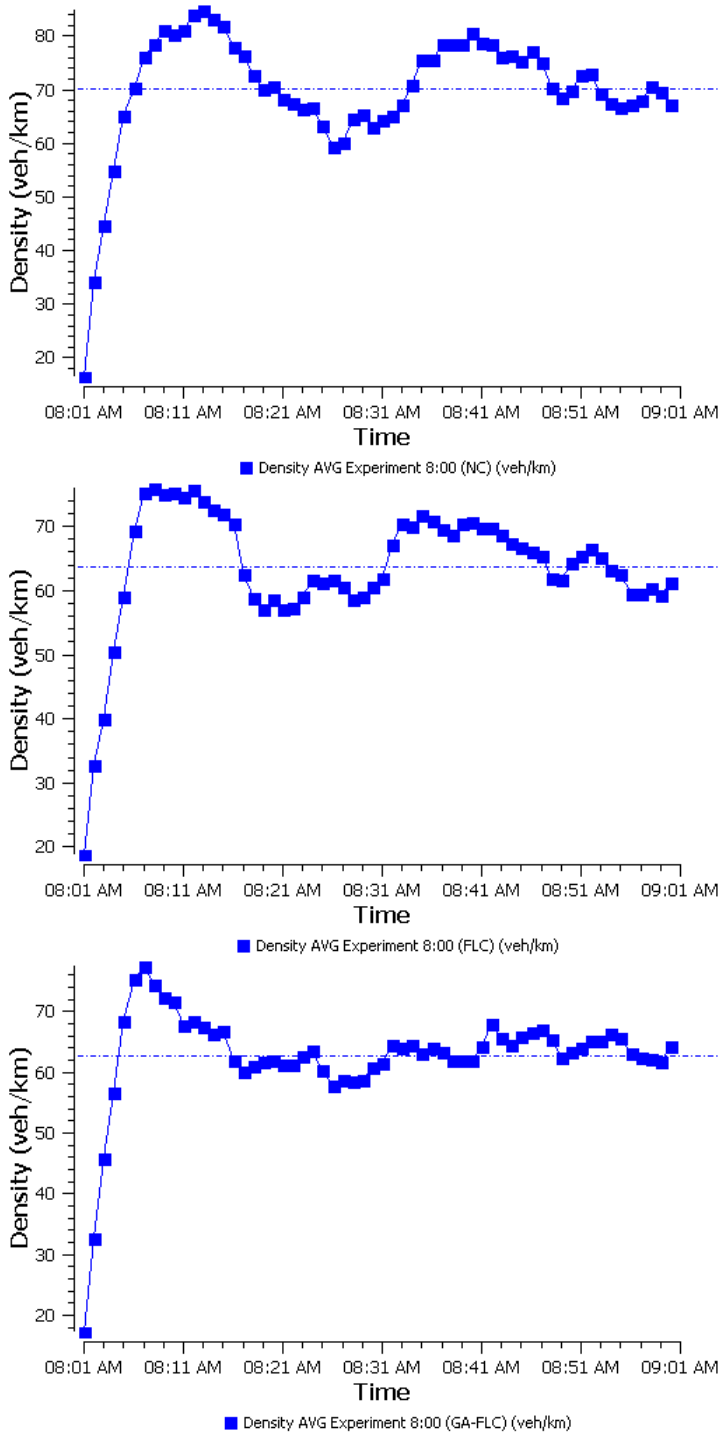




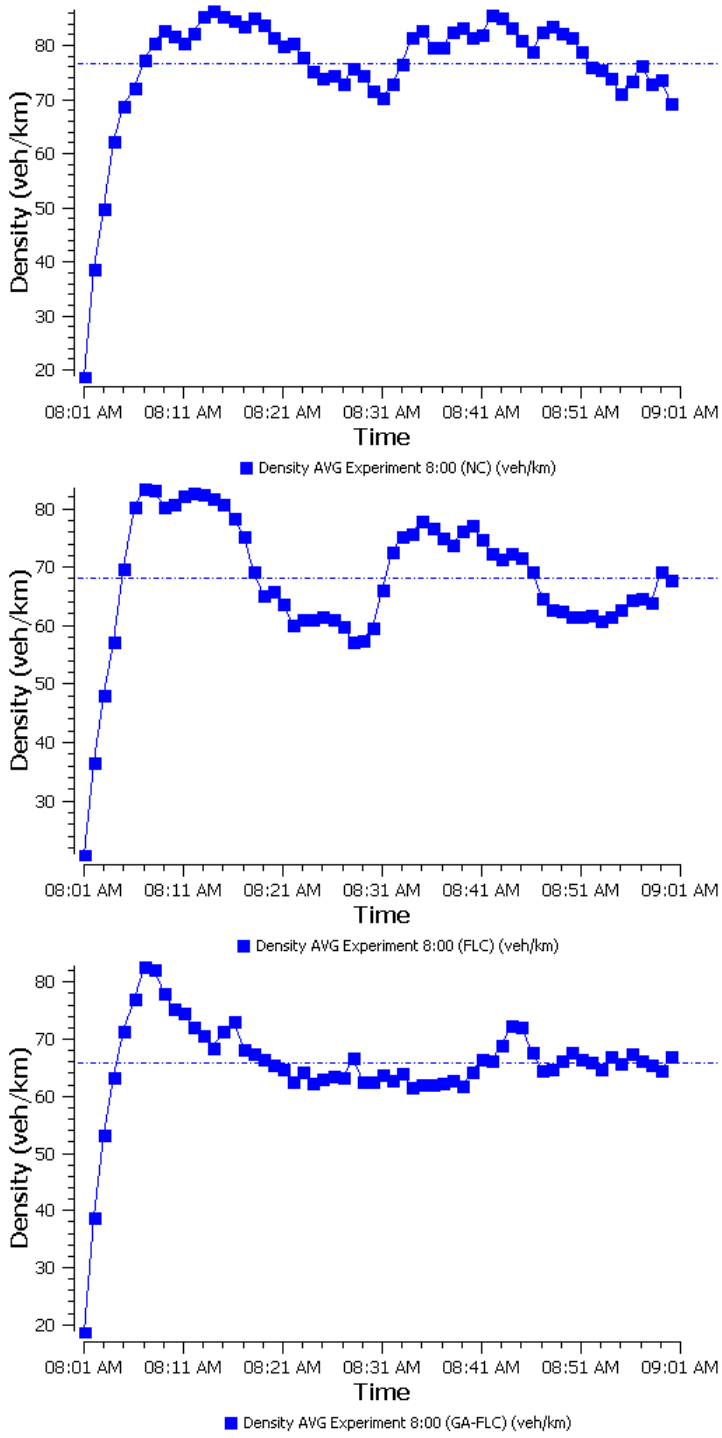
Total demand: 3200-1400 vehs/h



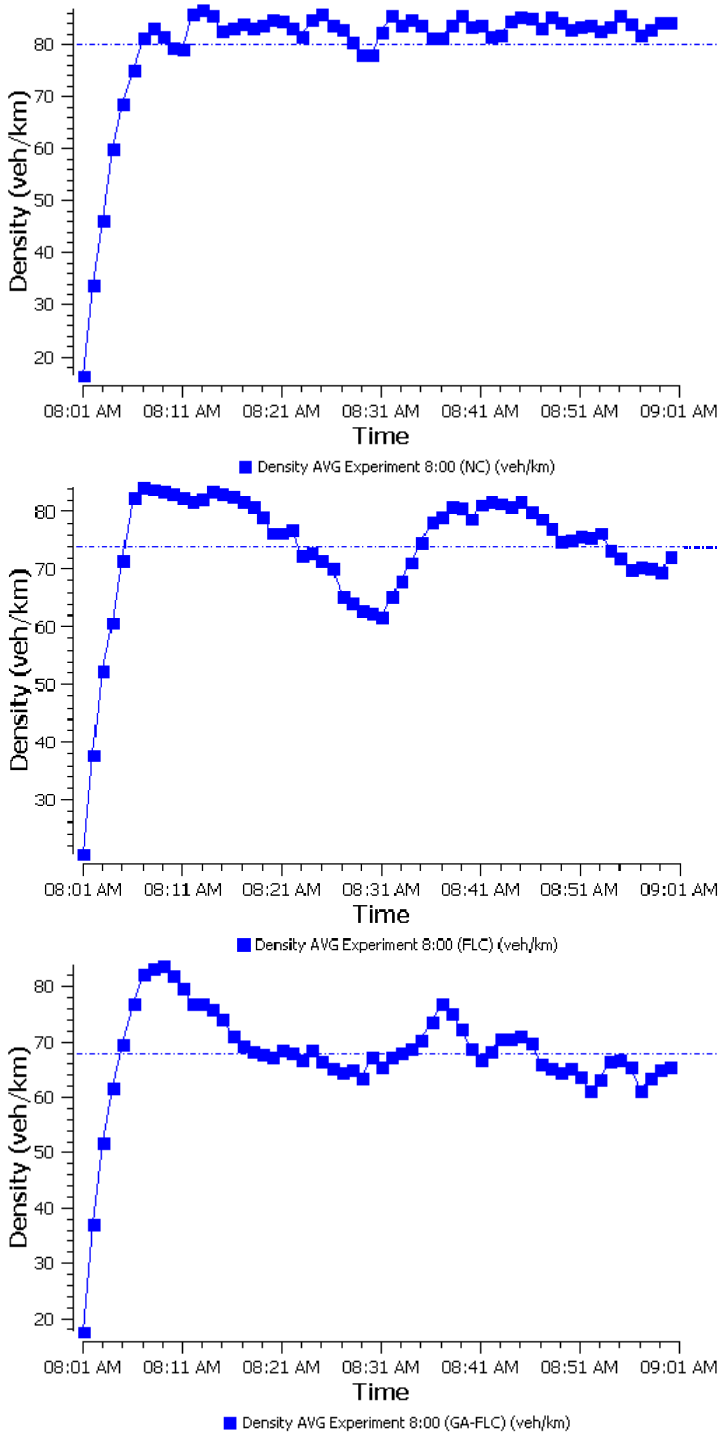
Total demand: 3400-1400 vehs/h



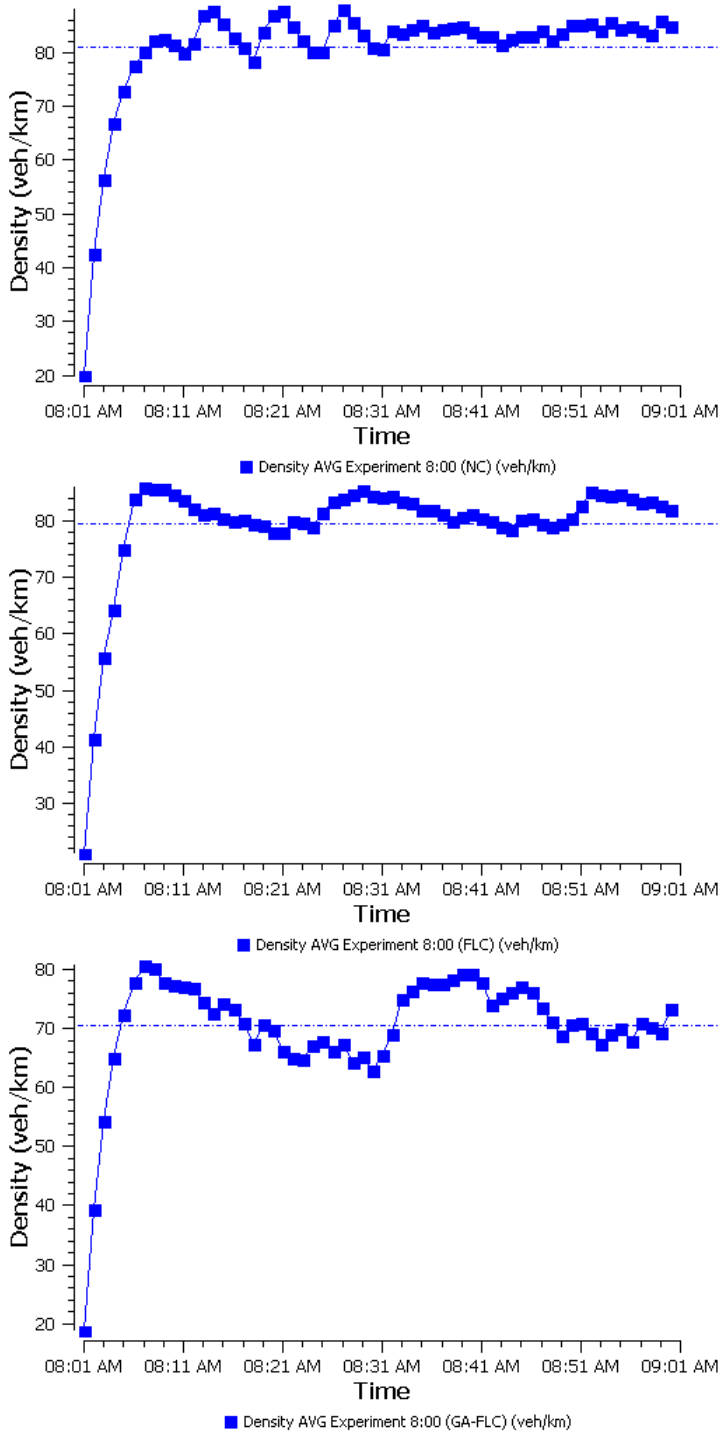
Total demand: 3600-1400 vehs/h



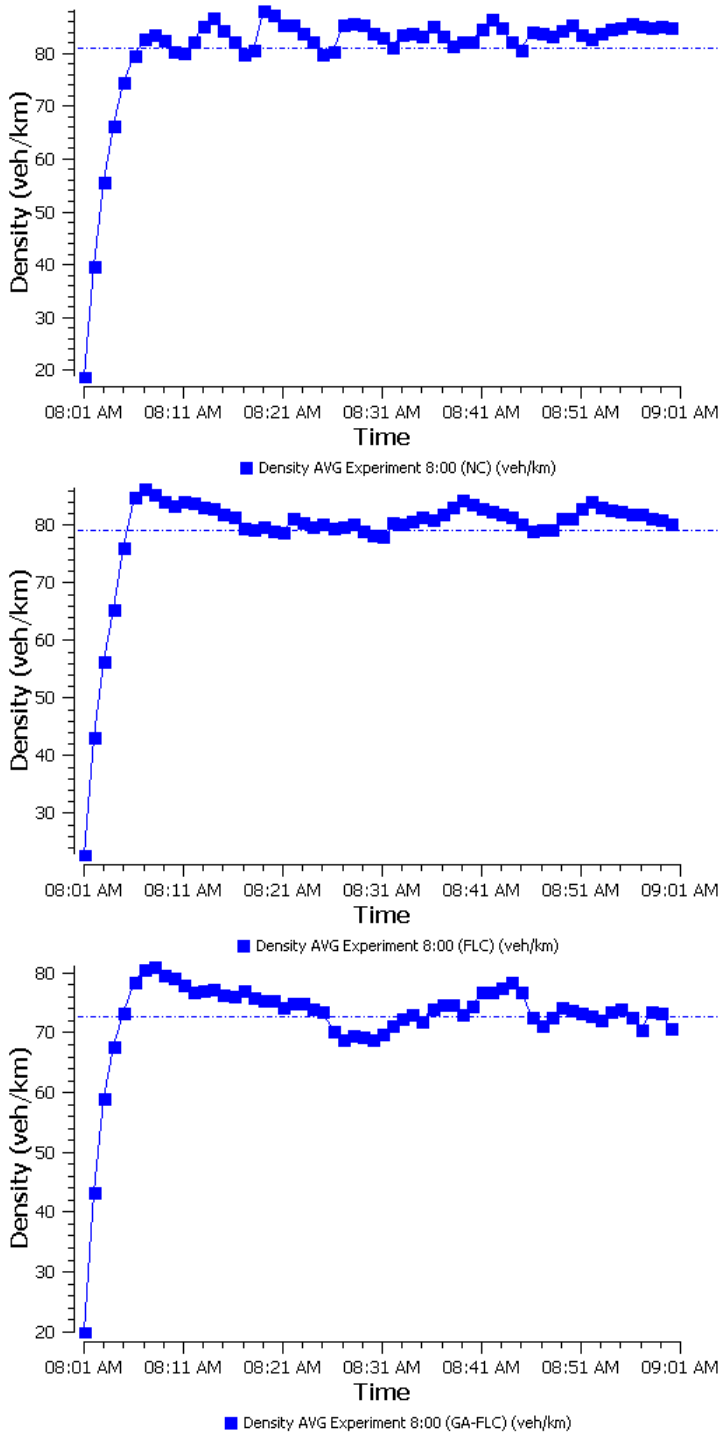
Total demand: 3800-1400 vehs/h



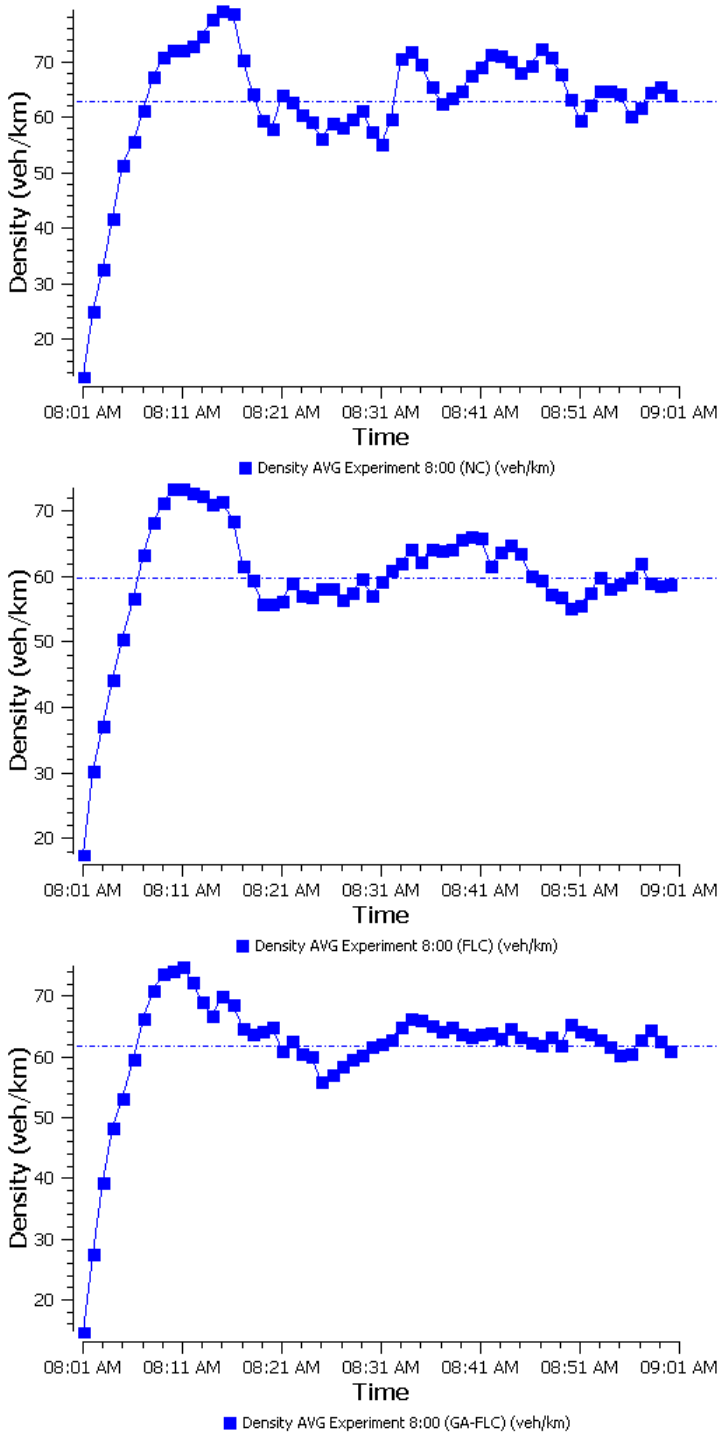
Total demand: 4000-1400 vehs/h



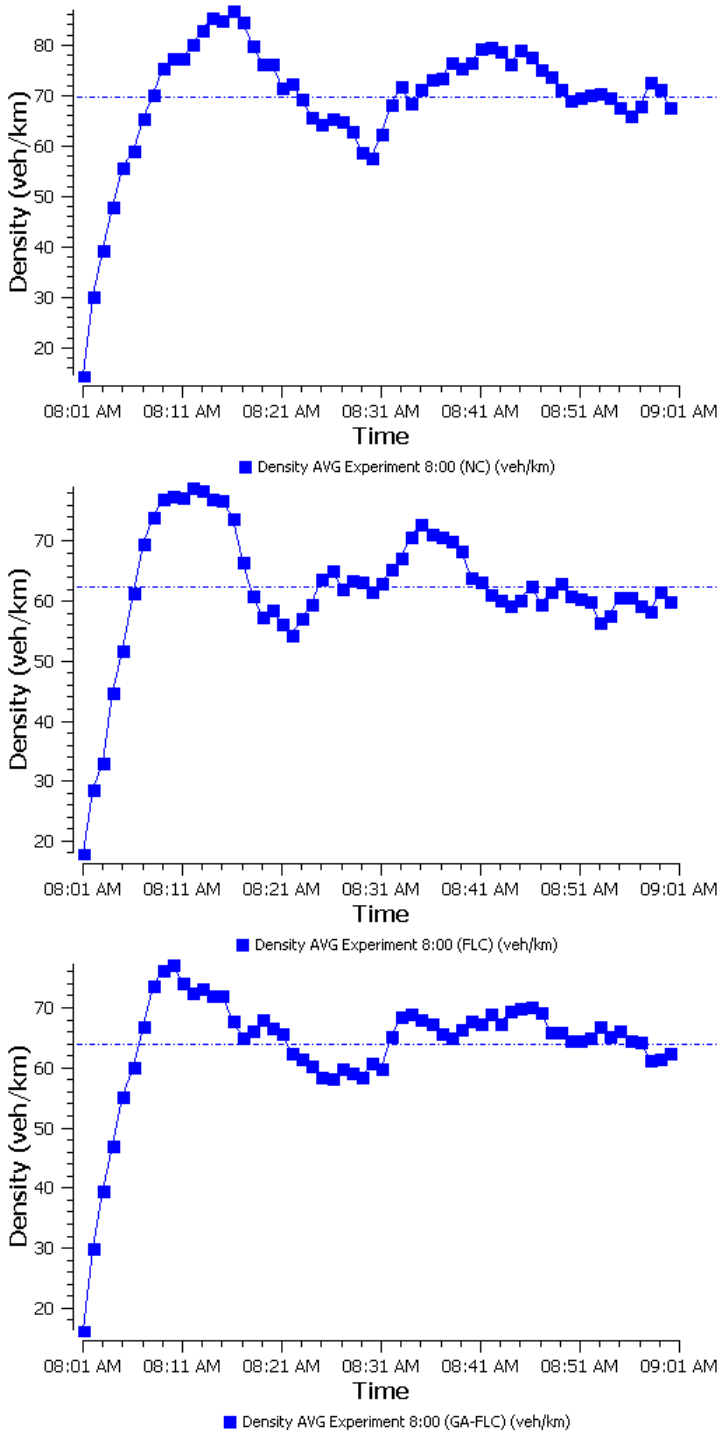
Total demand: 4200-1400 vehs/h



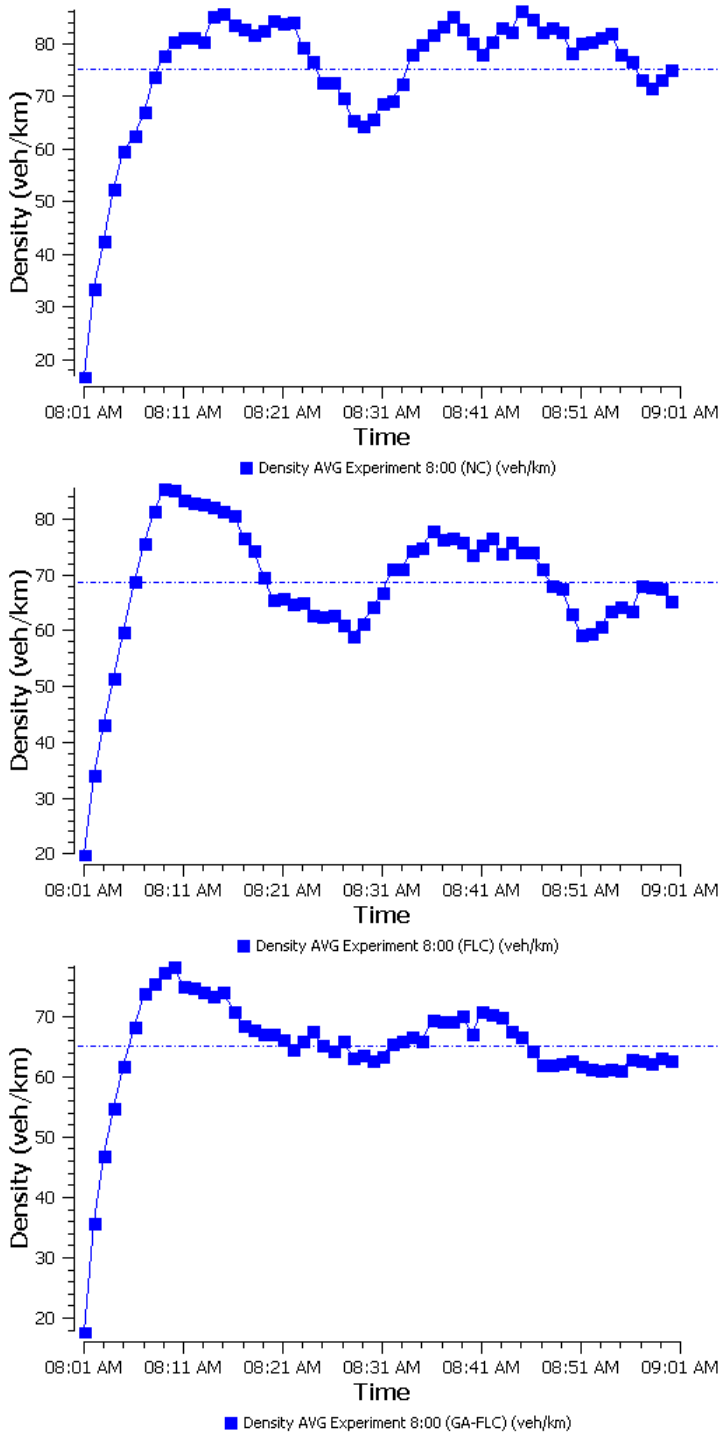
Total demand: 3200-1200 vehs/h



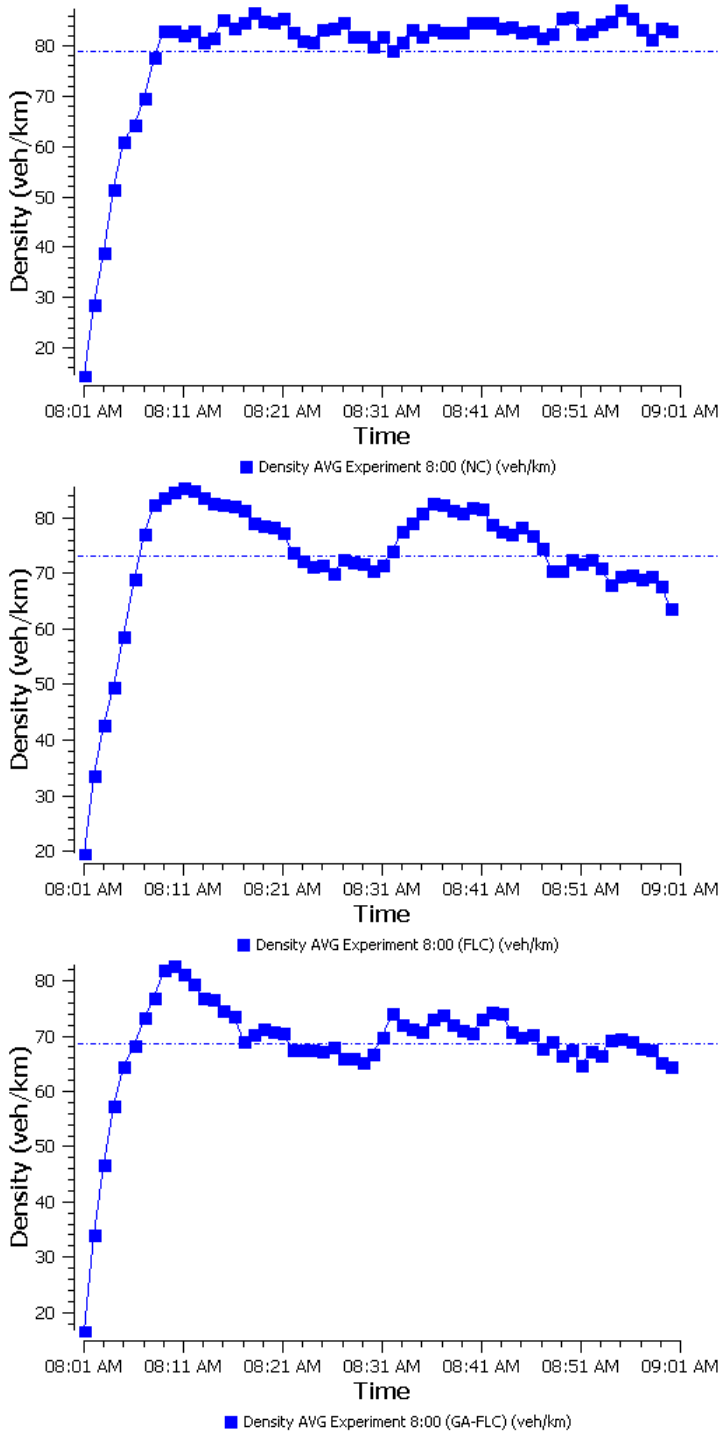
Total demand: 3400-1200 vehs/h



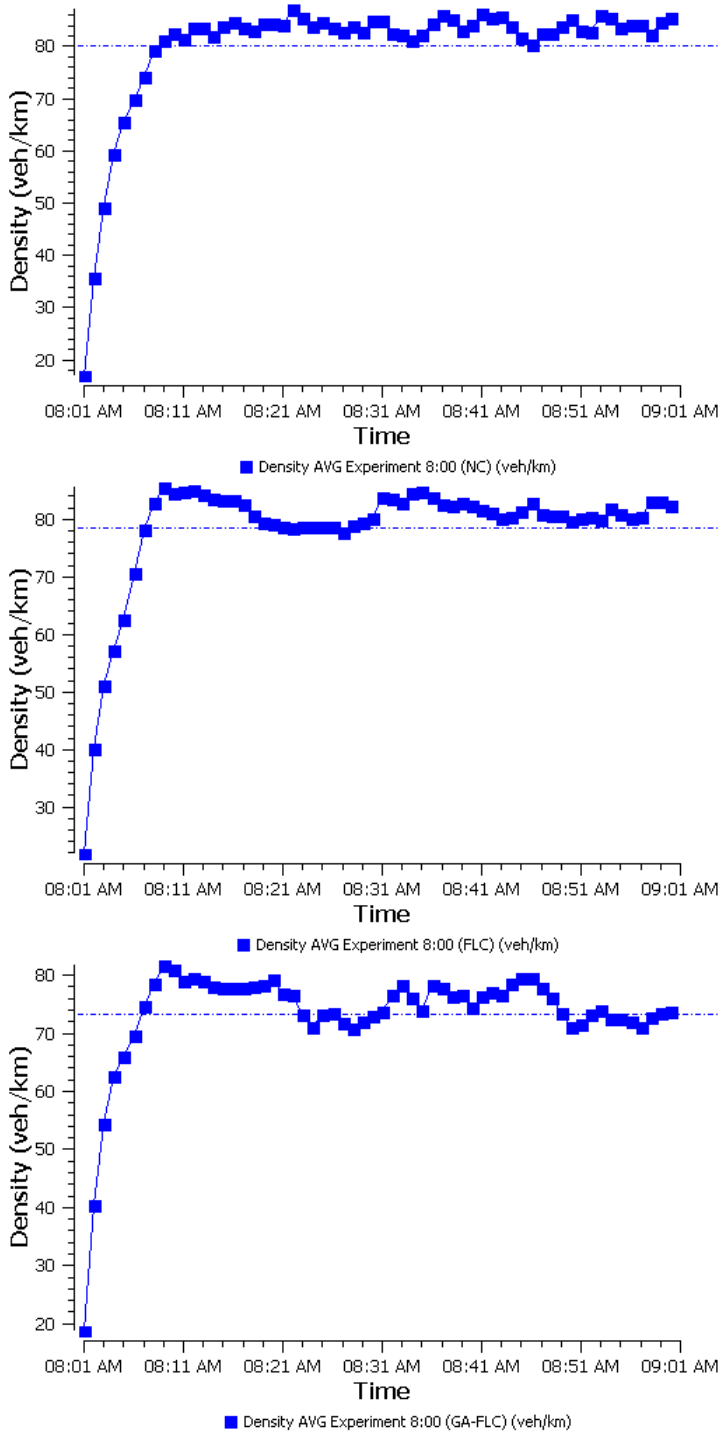
Total demand: 3600-1200 vehs/h



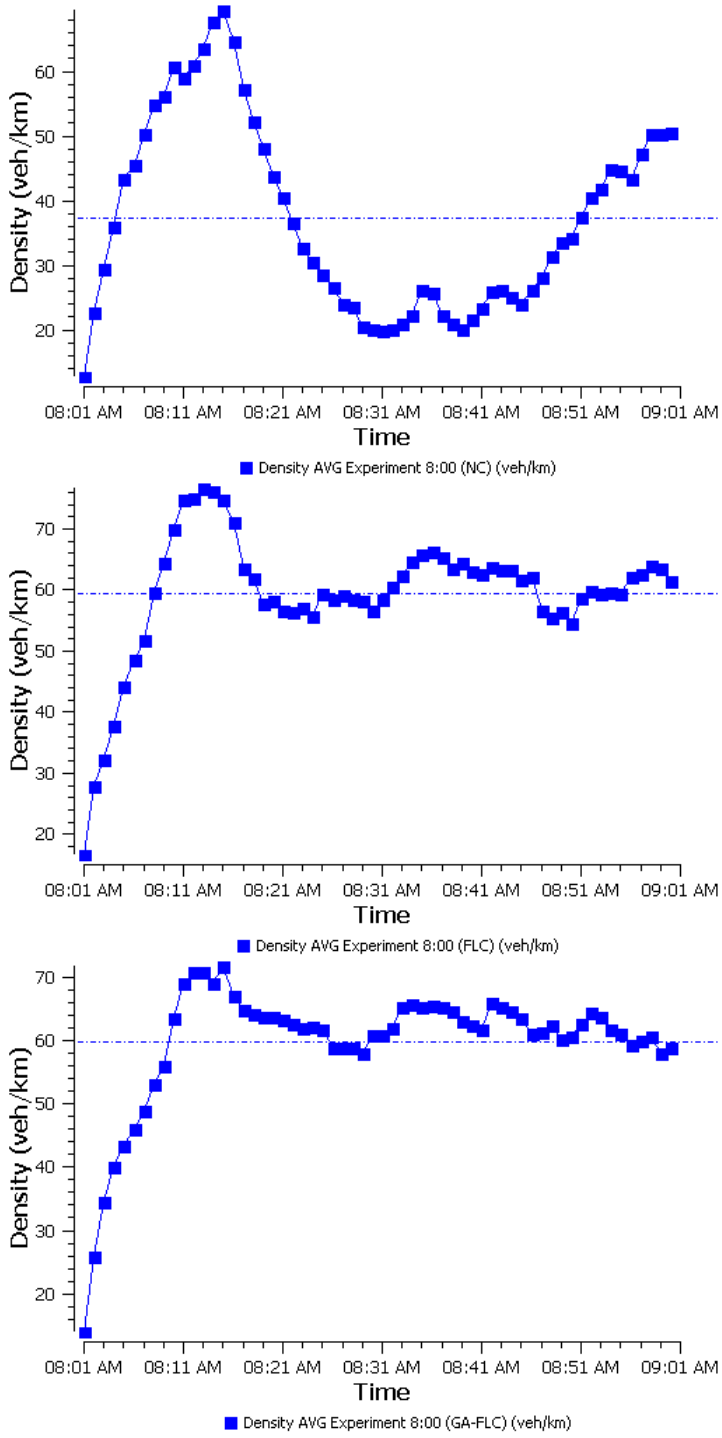
Total demand: 3800-1200 vehs/h



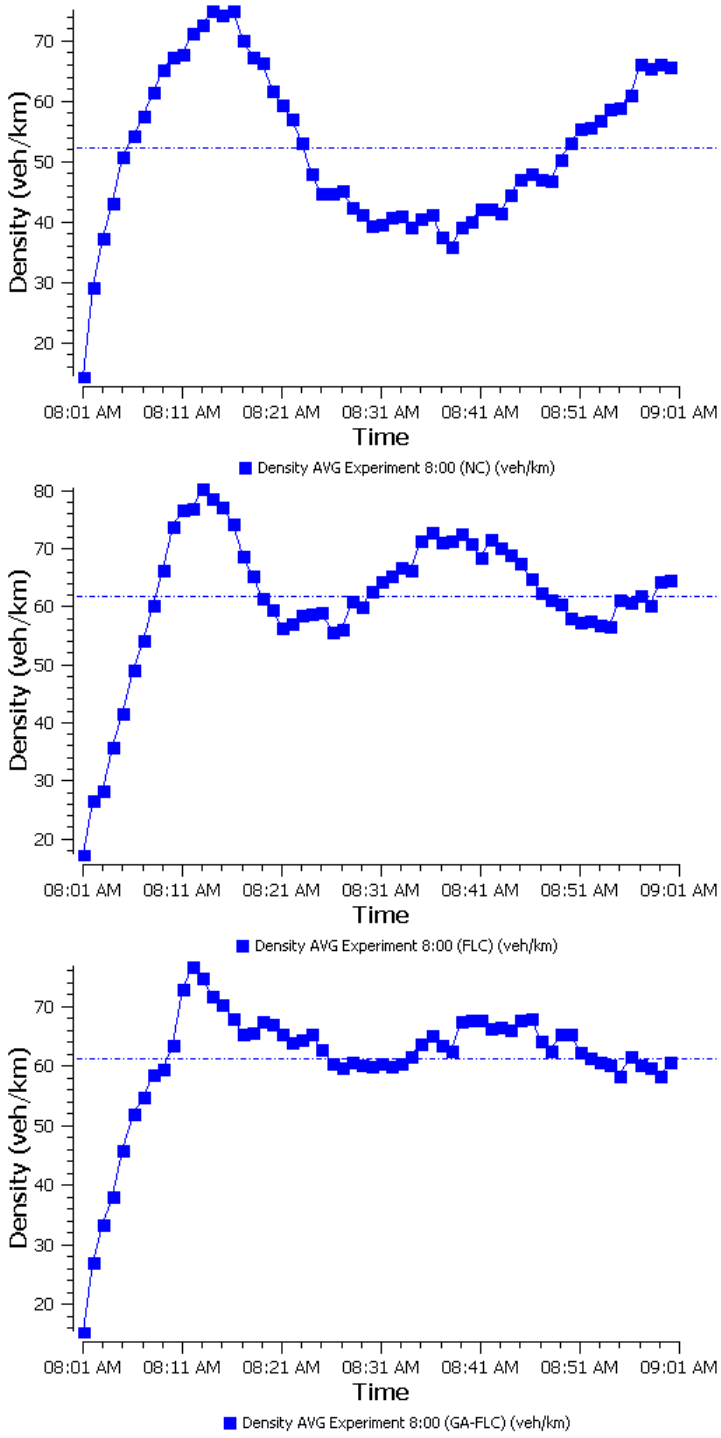
Total demand: 4000-1200 vehs/h



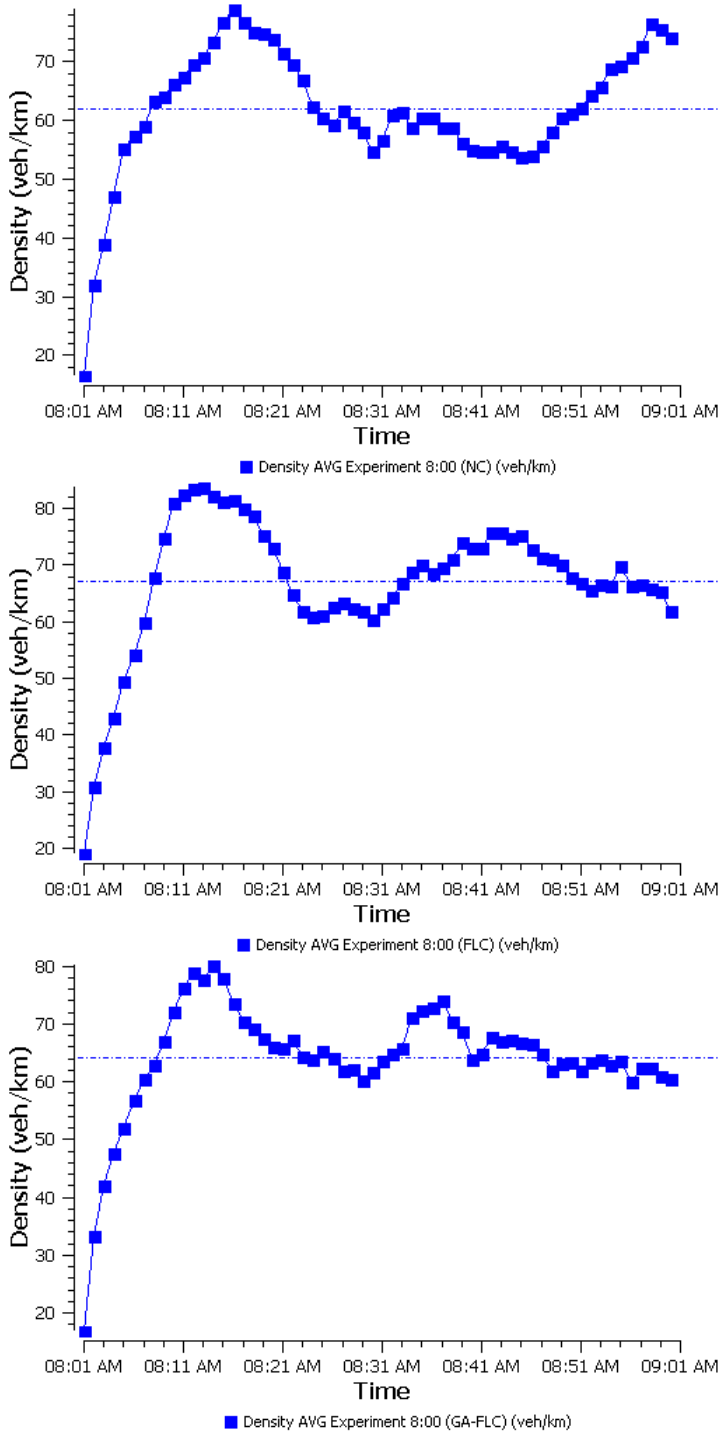
Total demand: 3200-1000 vehs/h



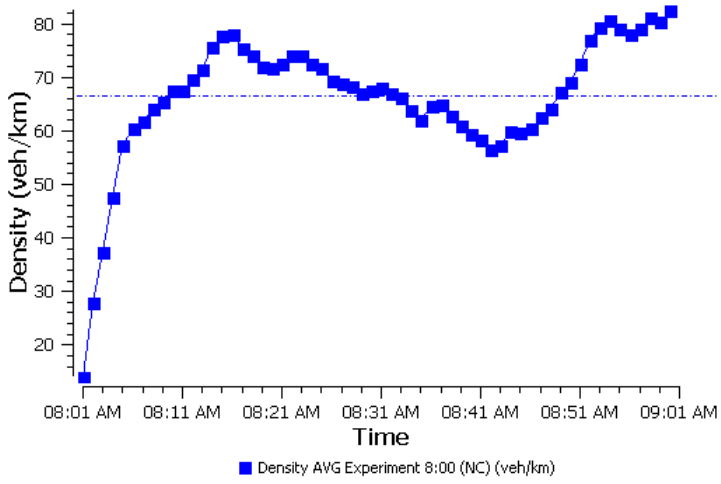
Total demand: 3400-1000 vehs/h

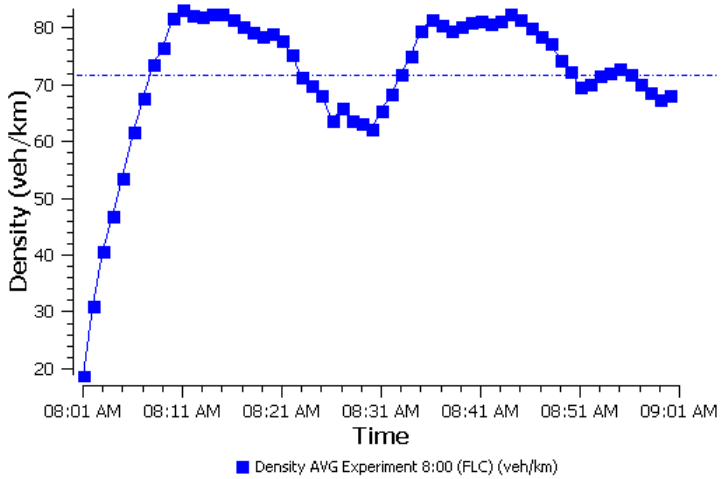


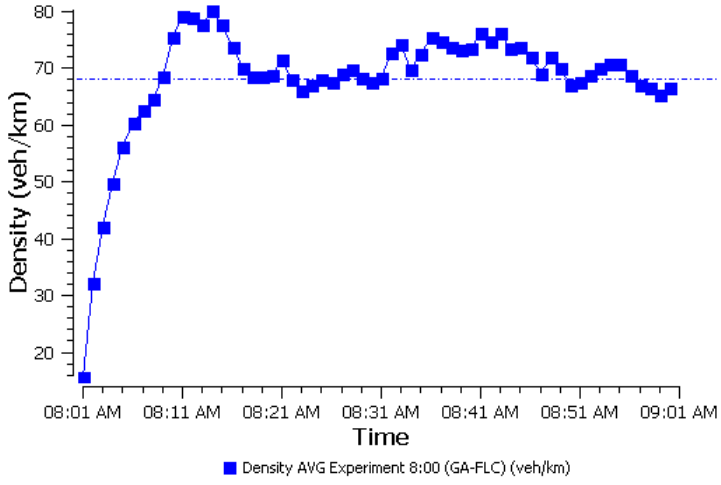
Total demand: 3600-1000 vehs/h



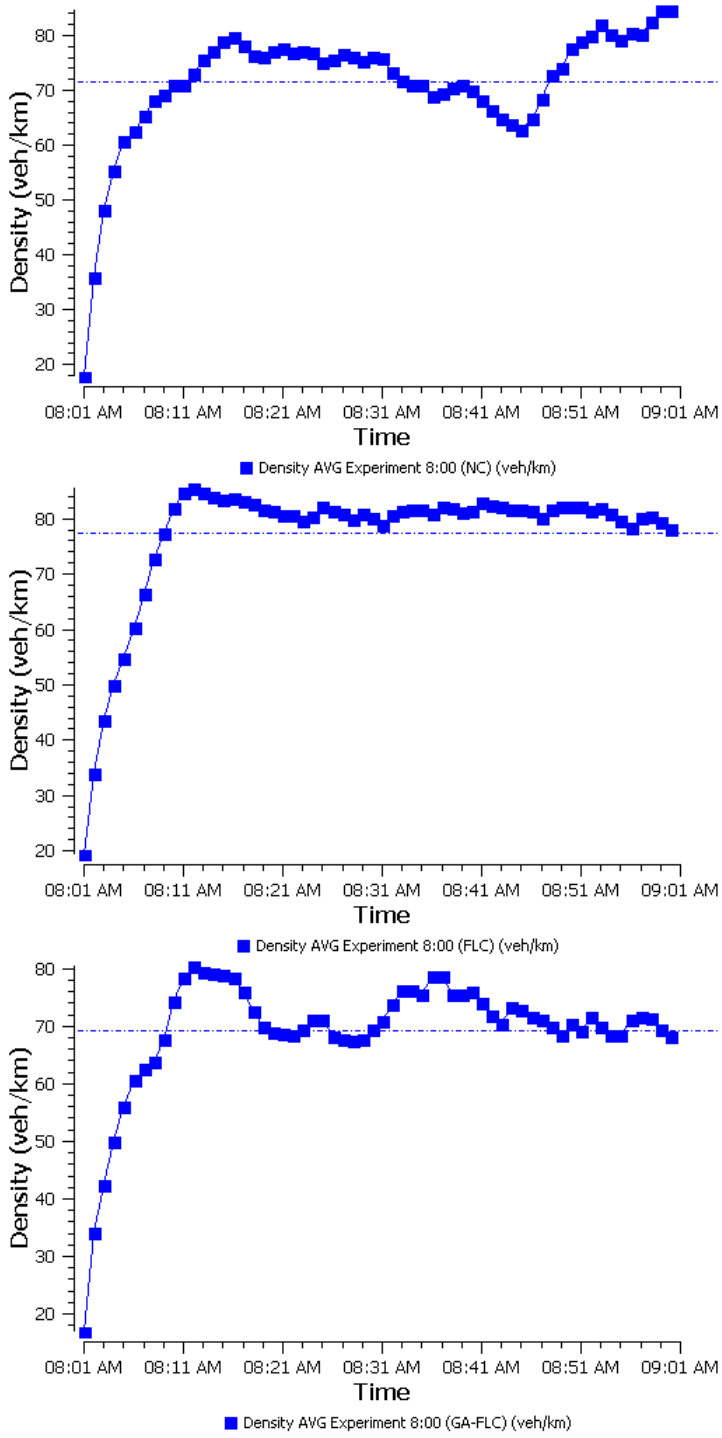
Total demand: 3800-1000 vehs/h







Total demand: 4000-1000 vehs/h



Total demand: 4200-1000 vehs/h