Research Report
Research Project T9903, Task 48
Fuzzy Neural Seattle

SIMULATION TESTING OF A FUZZY NEURAL RAMP METERING ALGORITHM

by

Cynthia E. Taylor
Research Engineer

Deirdre R. Meldrum
Assistant Professor

Department of Electrical Engineering
University of Washington, Box 352500
Seattle, Washington 98195

Washington State Transportation Center (TRAC)
University of Washington, Box 354802
University District Building, Suite 535
1107 N.E. 45th Street
Seattle, Washington 98105-4631

Washington State Department of Transportation
Technical Monitor
Art Lemke
Research Office

Sponsored by

Washington State Transportation Commission
Department of Transportation
Olympia, Washington 98504-7370

Transportation Northwest (TransNow)
University of Washington
135 More Hall, Box 354802
Seattle, Washington 98195

and in cooperation with
U.S. Department of Transportation
Federal Highway Administration
October 1995

TECHNICAL REPORT STANDARD TITLE PAGE

1. REPORT NO.	2. GOVERNMENT ACCESSION NO.	3. RECIPIENT'S CATALOG NO.
WA-RD 395.1		

4. TITLE AND SUBTITLE	5. REPORT DATE
Simulation Testing of a Fuzzy Neural Ramp Metering Algorithm	October 1995
	6. PERFORMING ORGANIZATION CODE

7. AUTHOR(S)	8. PERFORMING ORGANIZATION REPORT NO.
Cynthia E. Taylor and Deirdre R. Meldrum	

9. PERFORMING ORGANIZATION NAME AND ADDRESS	10. WORK UNIT NO.
Washington State Transportation Center (TRAC) University of Washington, Bx 354802 University District Building; 1107 NE 45th Street, Suite 535 Seattle, Washington 98105-4631	11. CONTRACT OR GRANT NO. WSDOT Agreement T9903, Task 48 USDOT Grant DTRS92-G 0010 Mod #2

12. SPONSORING AGENCY NAME AND ADDRESS	13. TYPE OF REPORT AND PERIOD COVERED
Washington State Department of Transportation Transportation Building, MS 7370 Olympia, Washington 98504-7370	Final Research Report
	14. SPONSORING AGENCY CODE

15. SUPPLEMENTARY NOTES

This study was conducted in cooperation with the U.S. Department of Transportation, Federal Highway Administration.

16. ABSTRACT

A fuzzy logic ramp metering algorithm will address the needs of Seattle's freeway system and overcome limitations of the existing ramp metering algorithm. The design of the fuzzy logic controller (FLC) reduced the sensitivity to sensor data which frequently contains errors or noise. The rule base effectively balanced two opposing needs: to alleviate mainline congestion by restricting the metering rate, and to disperse the ramp queue by increasing the metering rate. To avoid oscillation between these two conflicting demand, the controller used inputs that were more descriptive of congestion levels, providing smooth transitions rather than threshold activations.

Testing was performed with the freeway simulation software FRESIM. A multiple-ramp study site from Seattle's I-5 corridor was modeled using data such as freeway geometry entry volumes, desired speeds, and driver behavior. To evaluate the FLC under a variety of conditions, entry volumes and incidents (such as blocked lane or reduced capacity) were varied to create six test data sets. The performance of the FLC was compared to that of other available controllers, including clock, demand/capacity, and speed metering. The objective was to maximize total vehicle miles, maximize mainline speeds, and minimize delay/vehicle-mile while maintaining an acceptable ramp queue. For five of the six data sets, the FLC outperformed the other three controllers. In the FLC, sensors from the on-ramp were helpful in maintaining an acceptable ramp queue. Future work will involve on-line testing of the FLC.

17. KEY WORDS	18. DISTRIBUTION STATEMENT
Artificial neural networks (ANN), fuzzy logic controller (FLC), traffic data prediction, ramp metering	No restrictions. This document is available to the public through the National Technical Information Service, Springfield, VA 22616

19. SECURITY CLASSIF. (of this report)	20. SECURITY CLASSIF. (of this page)	21. NO. OF PAGES	22. PRICE
None	None	64	

DISCLAIMER

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SUMMARY

This report discusses simulation results for the fuzzy logic controller tested under Phase I of the project "Fuzzy Neural Ramp Metering Algorithm Implementation."[1] The overall objective of this project is on-line implementation of a fuzzy neural ramp metering algorithm. Phase I involves testing the controller in simulation, implementing the neural networks on-line, and incorporating the fuzzy controller into Seattle's Traffic Systems Management Center (TSMC). Phase II will involve testing the fuzzy neural controller on-line.

The fuzzy logic controller (FLC) was tested using FRESIM, a microscopic freeway simulation model. A multiple-ramp section of I-5 in Seattle was modeled. Through simulation testing, the fuzzy control base was minimized and default control parameters were determined. The FLC was compared to no metering, clock metering, speed metering, and demand/capacity metering. To evaluate FLC under a variety of conditions, entry volumes and incidents were varied to create six testing sets. The performance objectives were, in order of priority, to maximize total vehicle-miles, maximize mainline speeds, and minimize delay per vehicle while maintaining an acceptable ramp queue.

In five out of six data sets, the FLC outperformed the other ramp metering types. In general, the FLC performed well in a variety of situations, particularly when demand exceeded capacity. Because speed and demand/capacity metering do not use ramp inputs, they are liable to form unreasonable ramp queues under heavy congestion. In the FLC,

---

[1]TransNow grant #DTRS92-G-0010 MOD#2;
WSDOT Agreement Number T9903, Task 48.

the sensors from the on-ramps were helpful in maintaining reasonable ramp queues. The FLC effectively balanced between the conflicting needs of the ramp queue and mainline congestion because it considered these factors simultaneously. Each metered ramp had a parameter input file, which allowed the controller to be modified without recompiling the software. Consequently, maintenance costs should be minimal.

Simulation results suggested that the FLC algorithm will perform better on-line than it did with the FRESIM model. The benefits of ramp metering for conventional controllers are typically higher on-line than in simulation. Whereas numerous on-line studies have reported that metering has improved performance measures 30 to 60 percent in comparison to no metering (Robinson and Doctor, 1989), metering in simulation testing improved performance measures only up to 15 percent. Differences between on-line studies and testing results exist for reasons such as the model's difficulty in representing rapid oscillations in traffic flow. Given these limitations in FRESIM, the ability of ramp metering to smooth traffic flow and delay critical flow breakdown is not as evident in simulation. Hence, the benefits of ramp metering are expected to be more pronounced with on-line testing.

CONCLUSIONS AND RECOMMENDATIONS

Simulation testing was worthwhile for preliminary controller minimization, testing, and tuning. Because simulation results were encouraging, the researchers recommend that FLC implementation and testing on-line be continued as planned. Proposals to TransNow and WSDOT by Meldrum and Taylor describe the implementation tasks that will be performed.

Ramp metering benefits freeway efficiency only under certain circumstances, such as when demand is near capacity. By making ramp metering more versatile, the researchers plan to increase the range of conditions for which ramp metering is useful. For on-line implementation, two features are recommended to improve ramp metering versatility:

1. automatic on/off to allow congestion levels to determine when ramp metering begins and ends
2. adaptive control so that the rule weights will adjust in real time to various traffic conditions such as incidents

Feature 1 will operate ramp metering only when it is helpful, whereas feature 2 will broaden the conditions for which ramp metering is useful.

A system-wide evaluation is necessary to determine the effectiveness of any ramp metering algorithm. It is unreasonable to assume that the effects of a single ramp can be isolated and then generalized to the entire system. When real-time inputs are used for control, simulation testing has shown that the specified headways depend on each other and the effect they have on the system. Although a single ramp is useful for debugging

purposes, multiple metered ramps should be used for on-line testing. If time permits, a hierarchical fuzzy controller will be developed to adjust the local metering rates for global performance.

With on-line implementation of fuzzy neural control, the researchers expect to achieve a moderate improvement over Seattle's current ramp metering algorithm in terms of total vehicle-miles traveled, travel times, and delay per vehicle. Maximizing freeway efficiency is of great importance given the formidable costs of freeway accidents, delay, and wasted fuel. This improved performance will result in substantial savings in freeway operating costs. With no additional construction or hardware requirements, the reasonable implementation cost of the fuzzy neural controller is well worth the savings.

INTRODUCTION

Objective

The objectives of this research project are to test and implement a fuzzy neural ramp metering algorithm for the Seattle freeway system. This project strives to overcome the limitations of Seattle's current bottleneck ramp metering algorithm. The overall goal of this research project is to maximize freeway efficiency.

Problem Statement

Traffic congestion is a growing problem worldwide that cannot be solved by roadway construction alone. Freeway congestion in the U.S. results in approximately 2 billion hours of delay per year and contributes to an estimated $70 billion in annual accident costs (Fenton, 1994). Roadway travel is expected to double by the year 2010. However, geographical boundaries, construction costs, and environmental issues limit roadway construction. Practical solutions require maximizing freeway capabilities by methods such as public transit, high occupancy vehicles, variable direction lanes, and ramp metering.

This paper focuses on ramp metering, which uses traffic signals to regulate the rate at which vehicles are admitted onto the freeway. Ramp metering improves freeway flow by delaying bottleneck formation and smoothing vehicle entry. This research tests a fuzzy neural ramp metering algorithm that uses artifical neural networks (ANNs) for traffic prediction.

REVIEW OF CURRENT PRACTICE

On-line Studies of Ramp Metering

In 20 metropolitan areas across the United States, on-line studies have shown that ramp metering dramatically improves travel times, decreases accident rates, and decreases fuel consumption. A six-year study on the effects of ramp metering in Seattle indicated that the travel time for a specific 6.9-mile I-5 course decreased from 22 to 11.5 minutes, and the accident rate decreased by 39 percent. Mainline volumes increased by 86 percent northbound and 62 percent southbound (Henry and Mehyar, 1989). These benefits are comparable to those experienced in other cities where ramp metering has been implemented. In Portland, the travel time for a given 6-mile section of I-5 decreased from 23 to 9 minutes. Traffic accidents decreased by 43 percent during peak periods, and fuel consumption was reduced by 540 gallons/weekday (Robinson and Doctor, 1989). In Minneapolis/St. Paul, studies of a 5-mile section of I-35E indicated that ramp metering increased overall speeds by 16 percent, increased volumes by 25 percent, and decreased the accident rate by 38 percent. In Denver, speeds increased by 57 percent and average travel times decreased by 37 percent on a 2.9-mile section of I-25 (Corcoran and Hickman, 1989). In Detroit, evaluation showed that metering decreased the accident rate by 50 percent on I-94, while increasing speeds by 8 percent and increasing volumes by 14 percent (Kostyniuk, 1988). Metering in Austin increased speeds by 60 percent and increased volumes by 8 percent for a 2.6-mile segment of I-35N (Marsden, 1981). Metering in Long Island decreased travel time by 20 percent and decreased carbon monoxide emissions by 17 percent (James Kell and Associates, 1989).

Given the highly beneficial effects of ramp metering, optimizing metering rates is of great importance. Even slight improvements in the ramp metering algorithm may have significant returns in terms of operating costs, increased safety, reduced delay, and decreased fuel consumption.

Operation of Current Seattle Algorithm

The factors that make Seattle's current ramp metering algorithm more sophisticated than others in this country include a volume reduction based on downstream bottlenecks and further local adjustments, such as advanced queue override (Havinoviski, 1991). The Seattle ramp metering system responds to real-time loop detector data through a centralized computer and field-located microprocessors. The controller calculates both a local metering rate and a bottleneck metering rate and uses the more restrictive of these two rates (see Figure 1). The local metering rate is based upon adjacent upstream mainline occupancies. Linear interpolation between the actual occupancy and predetermined metering rates for given occupancies determines the local metering rate. The bottleneck algorithm activates when congestion on a downstream bottleneck-prone section surpasses a predetermined occupancy threshold. In this event, the algorithm reduces the number of vehicles entering the freeway by the bottleneck section's *storage rate* (the number of vehicles entering the section minus the number of vehicles exiting the section). This volume reduction is distributed over the upstream ramps that can influence that bottleneck. The number of ramps that can affect a bottleneck varies for each site. A weighting factor for each ramp determines the fraction of volume reduction targeted for that ramp.

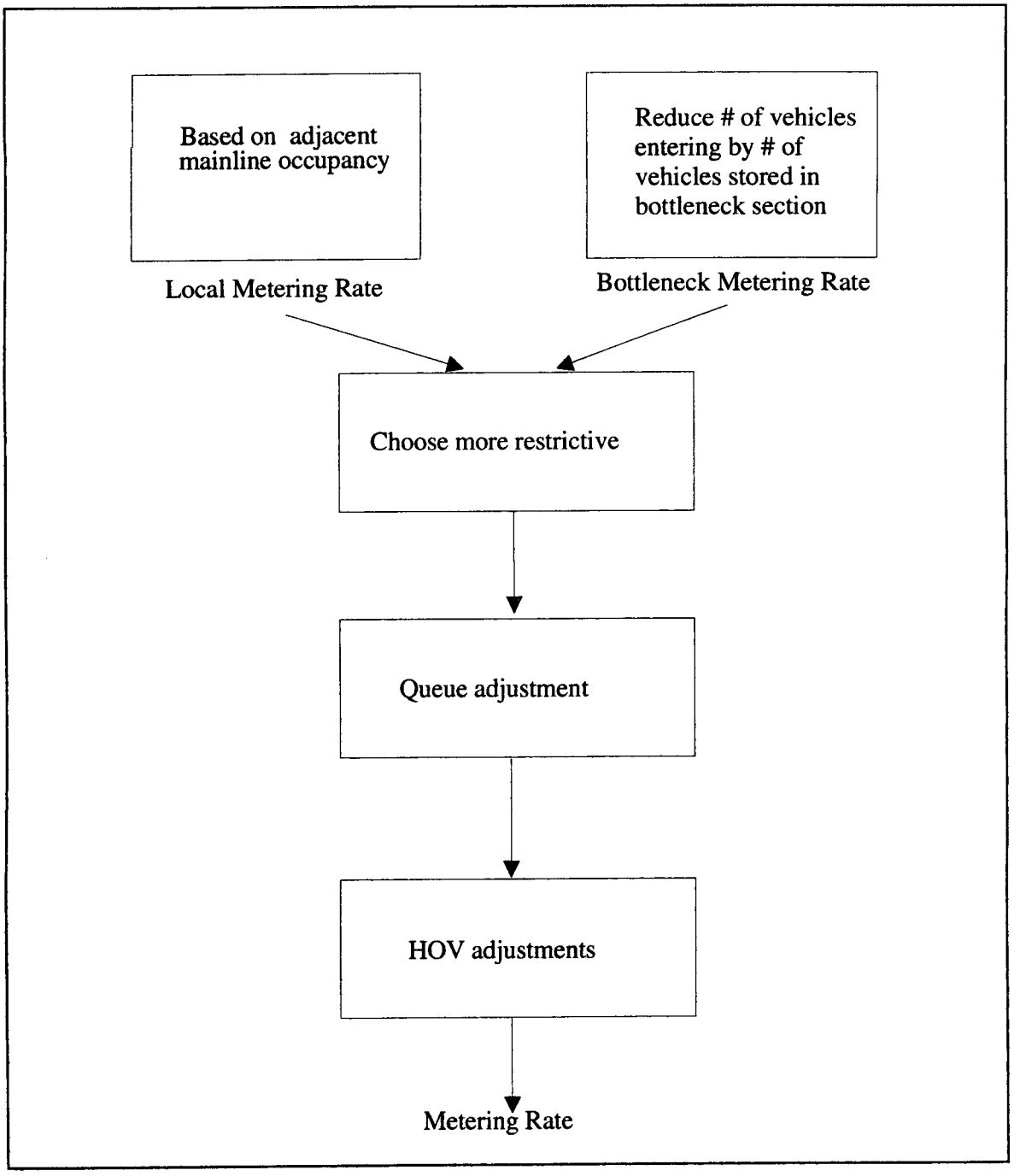

Figure 1. Flowchart of Seattle's ramp metering algorithm

After selecting the more restrictive of the local and bottleneck metering rates, the controller further adjusts the metering rate on the basis of local conditions. Queue adjustments prevent the ramp queue from blocking arterials. A queue adjustment occurs when the occupancy on a ramp exceeds a predetermined threshold for at least a specified duration. In this event, the metering rate increases by a certain number of vehicles per minute. The increase is dependent on which of the two occupancy and duration threshold sets has been exceeded. An advance queue adjustment occurs when a loop detector located near the arterial activates over a particular occupancy threshold for at least a specified duration. The advance queue adjustment also increases the metering rate by a specific number of vehicles per minute. High occupancy vehicle (HOV) adjustment accounts for the difference between the number of cars targeted for freeway entry and the actual number of cars that enter. Basically, this adjustment subtracts the number of HOV and illegal entries per period from the metering rate.

Advantages of Fuzzy Logic Algorithm

One research group has demonstrated that the use of an FLC (fuzzy logic controller) is an improvement over traditional ramp metering strategies (Chen and May, 1990). They tested an FLC to control entry to the San Francisco-Oakland Bay Bridge, particularly under incident conditions. This ramp metering algorithm performed well in simulation but was not tested on-line. This study's fuzzy neural algorithm differs from Chen and May's in that it is proactive, covers multiple ramps, and uses more comprehensive inputs such as ramp data and upstream data.

This fuzzy neural algorithm offers several advantages over traditional controllers for the ramp metering application:

1. The FLC does not require a mathematical system model. The freeway system is difficult to model, being nonlinear and nonstationary. Unlike a conventional controller, the FLC is not limited by the accuracy of the system model.

2. The FLC can utilize partial or imprecise information, thereby reducing the sensitivity to missing input data. Seattle's current algorithm typifies the problems of missing data. The loop detectors that sense traffic data frequently contain errors due to noise, mechanical failure, construction, and transmission errors. The bottleneck ramp metering algorithm is susceptible to error because it calculates the metering rates directly from raw loop detector data. The FLC, on the other hand, preprocesses the inputs through fuzzification, stressing qualitative information over quantitative information (see Appendix A for details). The parallel nature of the rule base further lowers sensitivity to imprecise data. If a detector fails, inputs from the other detectors still fire appropriate rules. Several rules can produce each fuzzy class of metering rate, preventing reliance on any single detector.

3. The fuzzy neural algorithm uses ANN predictors to prevent or delay heavy congestion. The predictive capabilities address the time lag between problem detection and corrective action present in Seattle's existing ramp metering algorithm (Jacobson, Henry, and Mehyar, 1988). Once flow breakdown has occurred, congestion levels typically remain high until peak demand subsides. For this reason, the fuzzy neural algorithm focuses on delaying critical flow breakdown instead of merely reacting to it. When the ANN predicts a lull in

traffic flow, the fuzzy controller can allow more vehicles to enter, increasing the overall flow. When the ANN predicts a peak in traffic flow, the fuzzy controller can reduce the number of vehicles entering. By smoothing traffic flow, the predictive control algorithm increases freeway volumes and reduces travel times.

4. The FLC can intelligently balance conflicting needs. In particular, it can relieve mainline congestion while maintaining an acceptable ramp queue. Seattle's existing bottleneck algorithm illustrates the inherent struggle between these opposing ramp metering needs. A downstream occupancy threshold indicates when to alleviate freeway bottlenecks with a restrictive metering rate. The metering reduction is calculated directly from the downstream storage rate. However, plots of storage rate show that it is not an accurate indicator of congestion (Nihan, 1995). Even in stop and go traffic flow, the storage rate rapidly fluctuates between positive and negative values. In effect, the bottleneck metering rate fluctuates up and down along with the storage rate. The restrictive action prompted by mainline congestion is countered with a nonrestrictive metering rate to flush out the ramp queue when it exceeds occupancy thresholds. The controller can get caught cycling between these opposing actions.

   In constrast, the FLC combats an oscillatory metering rate through qualitative inputs rather than threshold activations. The ramp inputs to the FLC describe the queue formation over time, while downstream speed and occupancy inputs more readily identify mainline congestion (Iwasaki, 1991; Masher et al., 1975). With these more descriptive inputs, excessive queue formation can be prevented,

avoiding the cycling between restrictive and nonrestrictive metering rates. Smooth transitions occur because the FLC considers all input factors simultaneously instead of making a series of adjustments.

5. The fuzzy ramp metering algorithm emphasizes easy tuning without recompiling the code. With an input file for each metered ramp, rules can be stressed or minimized by changing their weighting factor, and sensitivity to particular inputs can be adjusted through fuzzification parameters. Consequently, development and maintenance costs are minimal in comparison to algorithms that require hardware or software modification.

PROCEDURES

Description of Fuzzy Ramp Metering Algorithm

The FLC interprets rules of thumb that describe a control strategy. This process involves three main steps: (1) *fuzzification* to convert the quantitative inputs into natural language variables, (2) *rule evaluation* to implement the control logic, and (3) *defuzzification* to map the rule outcomes to a numerical output. The resulting control action is an appropriate metering rate to permit vehicles onto the freeway. The introduction to FLC in Appendix A explains these steps, and Appendix B contains the fuzzy controller code, written in C.

Fuzzification

Fuzzification preprocesses the inputs to the FLC. This controller uses eight inputs sensed by five loop detectors. Table 1 describes each input, and Figure 2 shows its sensor location. Here the occupancy is defined as the time percentage that vehicles occupy that detector location. Both occupancy and speed are useful indicators of congestion levels. Variables are based on the previous 20-second time period, except for ramp occupancy and predicted occupancy, which use additional past samples.

Fuzzification translates each of these crisp inputs into the five fuzzy classes shown in Table 2. Each variable has several parameters that add flexibility to the class definitions (Table 3). The first two scaling parameters set the low limit (*LL*) and high limit (*HL*) for the dynamic control range of each variable. The following scaling equation normalizes the crisp variables from the (*LL, HL*) range to the (0,1) range:

$$scaled\ crisp\ variable = \frac{crisp\ variable}{HL - LL} - \frac{LL}{HL - LL}$$

Table 1. Description of Algorithm Variables

Variable	Description
OC	Mainline occupancy just before ramp outlet
DO	Downstream occupancy of nearest bottleneck-prone section
UO	Occupancy from adjacent upstream station
PO	1-minute prediction of mainline occupancy just before ramp outlet
SP	Speed for mainline just before ramp outlet
DS	Downstream speed of nearest bottleneck-prone section
QO	Ramp queue occupancy based on past six samples
AQO	Advanced queue detector occupancy over past sample
MR	Metering rate--the control action

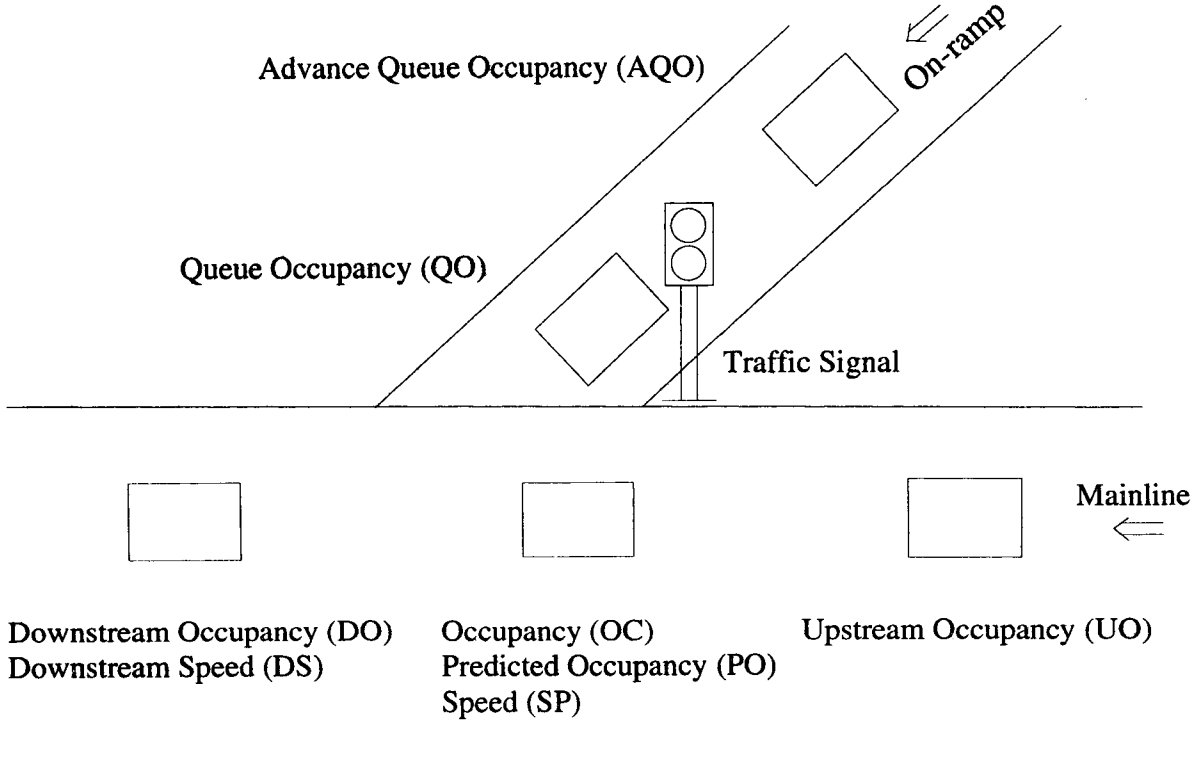

Figure 2. Detector location for algorithm variables

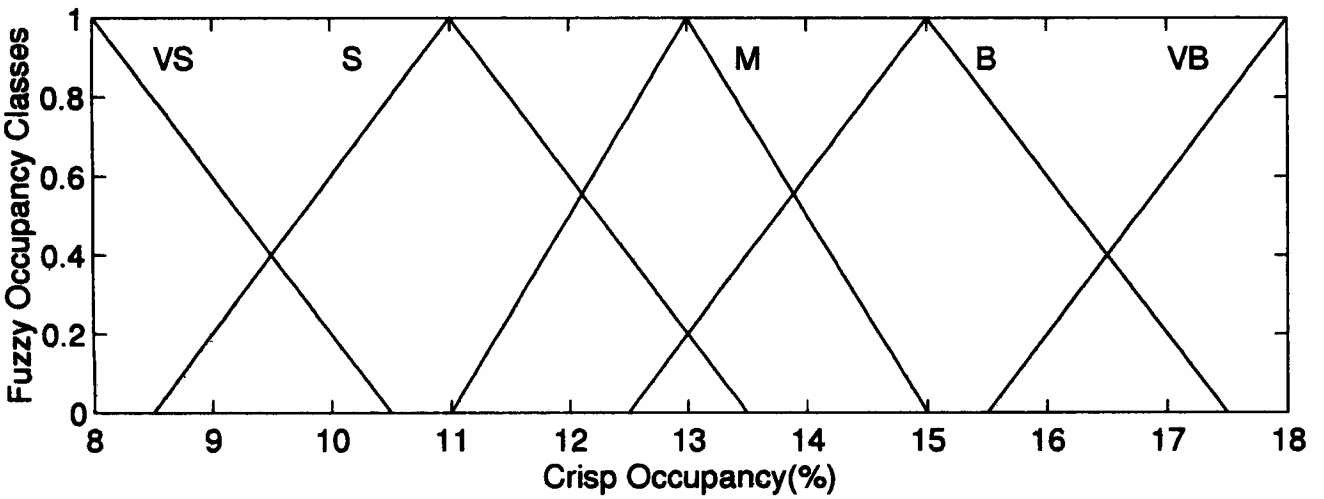

Figure 3. Fuzzy sets for occupancy

Table 2. Fuzzy Classes

i	Class	Description
1	VS	Very small
2	S	Small
3	M	Medium
4	B	Big
5	VB	Very big

Table 3. Parameter Input File

Fuzzification Parameters

		LL	HL	C_S	C_M	C_B	B_VS	B_S	B_M	B_B	B_VB
V	OC	8.0	18.0	0.3	0.5	0.7	0.25	0.25	0.2	0.25	0.25
a	DO	8.0	18.0	0.3	0.5	0.7	0.25	0.25	0.2	0.25	0.25
r	UO	8.0	18.0	0.3	0.5	0.7	0.25	0.25	0.2	0.25	0.25
i	PO	8.0	18.0	0.3	0.5	0.7	0.25	0.25	0.2	0.25	0.25
a	SP	30.0	60.0	0.3	0.5	0.7	0.25	0.25	0.2	0.25	0.25
b	DS	30.0	60.0	0.3	0.5	0.7	0.25	0.25	0.2	0.25	0.25
l	QO	12.0	30.0	0.3	0.5	0.7	0.25	0.25	0.2	0.25	1.0
e	AQO	12.0	30.0	0.3	0.5	0.7	0.25	0.25	0.2	0.25	1.0
s	MR	2.0	5.0	0.3	0.5	0.7	0.25	0.25	0.2	0.25	0.25

Rule Weights

Rule #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Weight	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	2.0	2.0	2.5	2.5	1.0	2.0	3.0

Variable scaling simplifies the code by allowing all variables to use the same fuzzification equations, as well as allowing easy class modification. By adjusting the dynamic range, the sensitivity to a particular input can be increased, causing the subsequent rules to fire with greater frequency and magnitude.

In addition to high/low limit parameters, each variable class has a centroid (C_i) and base width (β_i) parameter to define the ith class. The S, M, and B classes are defined by an isosceles triangle with a base of $2\beta_i$ and a height of 1 (Figure 3 shows an example of fuzzy classes for occupancy). The triangle is centered at C_i and has slopes of $\pm\frac{1}{\beta_i}$.

The resulting fuzzy classes are calculated from the scaled crisp input, x. For S, M, and B,

$$f_i(x) = \begin{cases} \frac{1}{\beta_i}(x - C_i + \beta_i) & \text{for } C_i - \beta_i < x < C_i \\ -\frac{1}{\beta_i}(x - C_i - \beta_i) & \text{for } C_i < x < C_i + \beta_i \end{cases}.$$

A right triangle defines VS and VB. For VS, the peak is at 0, so C_i is $\frac{\beta_i}{3}$. The class is 1 if x is less than 0. For VS,

$$f_i(x) = \begin{cases} 1 & \text{for } x < 0 \\ -\frac{1}{\beta_i}(x - \beta_i) & \text{for } 0 < x < \beta_i \end{cases}.$$

For VB, the peak is at 1, so C_i is $1 - \frac{\beta_i}{3}$. The class is 1 if x is greater than 1. For VB,

$$f_i(x) = \begin{cases} \frac{1}{\beta_i}(x - 1 + \beta_i) & \text{for } 1 - \beta_i < x < 1 \\ 1 & \text{for } x > 1 \end{cases}.$$

An example of parameter values to define the set of fuzzy classes for occupancy are LL=8.0, HL=18.0, C=[0.083, 0.3, 0.5, 0.7,0.916], and β =[0.25, 0.25, 0.2, 0.25, 0.25]. Table 3 shows a sample input file containing the parameters that are user specified. Default values are shown. Notice that C _VS and C _VB are not user specified, as they can be calculated from β _VS and β _VB.

Rules

At the heart of the controller, the rule base describes the control strategy. The preliminary rule base was developed from heuristics and human expertise (Taylor, 1994). The original set of rules was minimized from 27 to 19 rules found necessary in simulation testing. Table 4 states these 19 essential rules, in which a comma between two premises

denotes a logical AND. To evaluate a rule's outcome, the FLC takes the minimum activation degree of the premise (see Appendix A for example). The default rule weighting reflects the relative rule importance.

Table 4. Rule Base for Fuzzy Ramp Metering Algorithm

Rule	Premise	MR outcome
1	OC_VB	VS
2	OC_B	S
3	OC_M	M
4	OC_S	B
5	OC_VS	VB
6	SP_VS, OC_VB	VS
7	SP_S	S
8	SP_B	B
9	SP_VB, OC_VS	VB
10	PO_VB	VS
11	PO_VS	VB
12	UO_M	M
13	UO_S	B
14	UO_VS	VB
15	DS_VS, DO_VB	VS
16	DS__S, DO_B	S
17	DS_M, DO_M	M
18	QD_VB	VB
19	AQO_VB	VB

The purpose of rules 1 through 5 is to form a complete rule base. In other words, at least one of these rules will always fire because the entire occupancy range is considered. Traditionally, an FLC individually considers all possible input combinations. However, this FLC has 5^8 possible input combinations, given that it has five fuzzy classes for each of its eight inputs. Although it is impractical to consider each of these possibilities individually, it is vital that at least one rule fire for every input combination. Rules 1 through 5 overcome this problem by covering the entire input space.

Rules 6 through 9 are closely related to rules 1 through 5. They couple speed with occupancy for a more specific congestion index and adjust the metering rate accordingly.

Rules 10 and 11 prevent congestion rather than simply react to it. With vehicles naturally traveling in platoons, a short-term prediction enables the ramp metering algorithm to fit more vehicles between the traffic gaps, resulting in higher freeway flow (rule 11). When a peak in traffic is predicted, the metering rate is reduced to prevent congestion formation (rule 10). With predictive capabilities, the ramp metering algorithm has a better opportunity to smooth traffic flow and maximize freeway efficiency.

Rules 12 through 17 use the knowledge that traffic waves travel forward in uncongested conditions, and shock waves travel backward during heavy congestion. Hence, occupancy from the upstream detector is used during lighter traffic conditions (rules 12 to 14), and occupancy/speed inputs from the downstream detector are used during heavier traffic conditions (rules 15 to 17). In medium congestion, forward and backward traffic waves coexist, in which case inputs from both upstream and downstream detectors are used. Rules 15 through 17 prevent or delay heavy congestion. The higher weighting of these rules emphasizes that these are the chief means by which ramp

metering is effective. These rules use the premise that high downstream occupancy along with low downstream speed indicate bottleneck formation, calling for a more restrictive metering rate.

Rules 18 and 19 prevent excessive queue formation on the basis of information from two ramp detectors (AO and AQO in Figure 2). A high queue occupancy indicates that the queue is building up over time. In this event, rule 18 increases the metering rate. The advance queue occupancy input, detected near the ramp entrance, prevents the queue from reaching the arterial (rule 19). Its higher weighting stresses its priority. The advance queue override is based on only the previous sample, so it responds immediately to ramp backup. This rule weighting scheme encourages the queue length to remain between the two ramp detectors.

Defuzzification

Defuzzification produces a control action, given a fuzzy set of rule outcomes. Before a numerical metering rate (MR) is calcualted, an *implication* step scales each rule's output class by its activation degree, known as correlation-product encoding. Correlation-product encoding allows use of the discrete centroid equation,

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

where w_i weights the importance of the ith rule, c_i is the centroid of the output class, and I_i is the implicated area of the output class. For classes of S, M, and B, the class area I_i equals β_i. For MR classes of VS and VB, the class area I_i equals $\beta_i/2$.

Once the centroid of the crisp MR has been found, this control action is rescaled back to its dynamic range using the specified *LL/HL* range. As with Seattle's bottleneck algorithm, the MR is then adjusted to account for the number of carpools and illegal entries that bypassed metering during the previous sampling interval. The maximum MR possible is 900 vehicles/lane/hour to allow a 4-second cycle for each car, when one car is released at a time. The minimum MR is 240 vehicles/lane/hour, for a maximum vehicle delay of 15 seconds. Drivers are apt to run a ramp metering light if it is red for more than 15 seconds. Thus, reasonable limits for headway (defined as the cycle length of the light sequence) are 4 and 15 seconds. The metering rate is equal to the sampling period divided by the headway.

Procedure for Simulation Testing

The following tasks were necessary to test the FLC:

- incorporate the FLC into the freeway model
- provide controller inputs
- describe the study site to the model
- calibrate the model
- create testing scenarios
- define performance criteria
- optimize control parameters
- compare the FLC against other controllers

The fuzzy controller was tested using the microscopic freeway simulation model FRESIM. For background on FRESIM, see other literature (FHWA, 1994). The FLC

code was incorporated into FRESIM as one of the available ramp metering algorithms. Programmers at Viggen/IDI Inc. modified FRESIM to communicate with the fuzzy controller. The fuzzy controller code was written in C, whereas FRESIM was written in FORTRAN. FRESIM was compiled using Lahey for a DOS operating system. Every control period (20 seconds) was used to match that of the TSMC) the FLC provides control rates for all metered ramps based on current traffic conditions.

FRESIM was unable to provide the 1-minute occupancy prediction. Although the FRESIM programmers believe it is possible to run a background simulation 1 minute in advance, they were not able to implement it. Instead, the occupancy from the previous 20-second sample was used as the predicted occupancy. Although this alternative is not ideal, it is acceptable for simulation testing for the following reasons. FRESIM traffic data do not fluctuate sharply like actual traffic does (see Limitations section for details). Thus, the previous value is a reasonable prediction for simulation purposes. In addition, a prediction is not as valuable within simulated control as it is for on-line control. Without oscillatory extremes in the simulated traffic conditions, the advantages of predictive control are less pronounced. It is also important that the ramp metering algorithm has the ability to function successfully without the predictive inputs, and these simulations tested that autonomy.

Another option would have been to obtain the prediction using ANNs. The ANNs could have been trained on the simulated traffic data. Although this prediction should have worked fine, it was not necessary to retest the ANNs. Consequently, this option was not pursued. The ANNs have already been shown to successfully predict actual traffic data (Taylor, 1994), so their potential to predict simulated traffic data is of little

consequence. For on-line testing, the ANN predictors will be valuable and will be utilized, allowing the ramp metering algorithm to capitalize on the extremes in actual traffic data.

An appropriate study site to evaluate ramp metering contains multiple ramps with recurrent congestion. With these criteria, the Seattle study site chosen was the northbound section of I-5 between NE Northgate Way and NE 175 Street (Figure 4). As required by FRESIM, an input file describes the freeway geometry in terms of links and nodes. Link lengths and detector locations were determined using the State Highway Log. A subset of this study site, with only one metered ramp, was also used for debugging and to isolate the effects of parameter adjustment.

During this study FRESIM did not have HOV modeling capabilities, so adjustments were made to account for HOVs. The HOV lane for this study site typically has 80 percent of the volume of the other mainline lanes during peak operation. Because FRESIM distributes the flow evenly across all lanes, the mainline volumes would drop when averaged with the HOV lane. A future version of FRESIM can model HOV lanes, but it would not have been suitable for this application. In the forthcoming FRESIM, the user will specify a percentage of vehicles as HOV. Those designated vehicles will move into the HOV lane at the earliest access point and exit the HOV lane near their destination. However, the problem with this scenario is that it requires specified access/exit points for the HOV lane. Seattle drivers can access/exit the HOV lane at any time, and they take advantage of that fact. Modeling Seattle's HOV lanes is a more

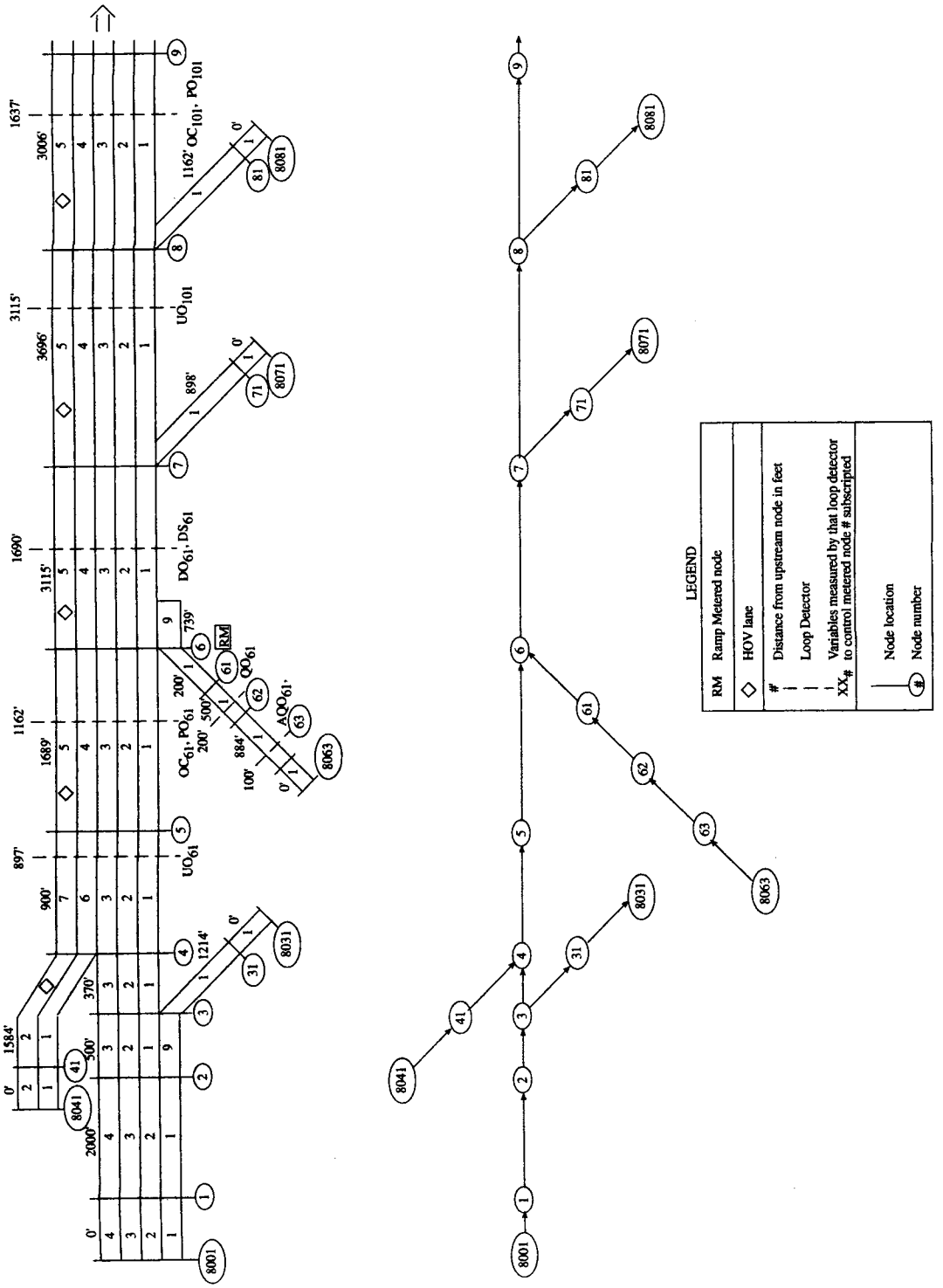

Figure 4. FRESIM study site--Part A

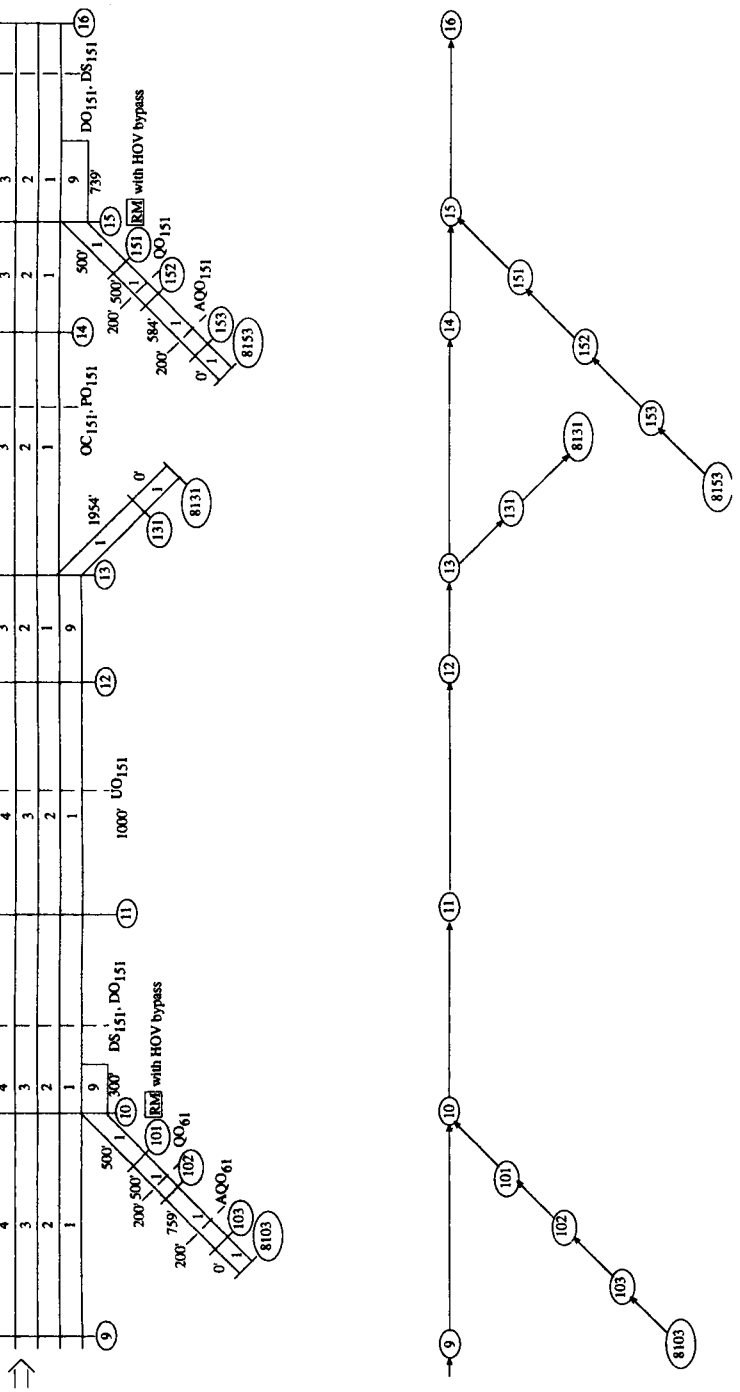

Figure 4. FRESIM study site--Part B

difficult problem than FRESIM's HOV modeling because it depends on driver behavior. A driver's use of the HOV lane depends on relative lane speeds, trip length, and ease in accessing the HOV lane. Upon examining historical loop detector data from Seattle's I-5, we found that drivers may take a while to enter the HOV lane, and usage varies from section to section.

With FRESIM unable to directly model the HOV lanes, the volume discrepancy was accounted for with an alternative technique. All entry volumes were padded by 5 percent to account for the disproportionate HOV volume. This padding was just enough to keep the mainline lane volumes from dropping when they were averaged with the HOV lane. Another technique tested was eliminating the HOV lane altogether and subtracting the HOV vehicles from the entry vehicles. These results produced higher congestion and did not appear to model the section as well.

In additional to the HOV lane, FRESIM was unable to model HOV bypasses. The study site contains two on-ramps where HOV vehicles bypass metering. To overcome this difficulty, the HOV vehicles were added into the entry volumes to be metered. The extent to which the lack of HOV capabilities jeopardized the model's accuracy with respect to ramp metering is not known, but no doubt, on-line testing will be necessary to verify any simulation results.

Care was taken to ensure that simulated behavior emulated actual freeway behavior as closely as possible. To calibrate the freeway model, the researchers determined the desired driver speeds and driver sensitivity (distribution of car following headway in seconds) by driving the study site and examining detector data. Other parameters that were adjusted were queue discharge delay (reaction time to move in

start/stop traffic), minimum separation for vehicle generation, and the courtesy factor (percentage of drivers that yield to lane changers).

Entry volumes and turn movements were determined from historical loop detector data. In original attempts to calibrate the model, 15-minute data were averaged over several weekdays to determine appropriate entry volumes. However, graphs of the averaged data placed beside the individual days demonstrated that the averaged 15-minute data did not accurately represent the fluctuations in typical traffic data. Because averaging the data and using a large time period reduced flow variations, the benefit of ramp metering in smoothing traffic flow was underrepresented (see Limitations section for explanation). To maintain greater variability in entry volumes, FRESIM was recalibrated with data of 5-minute periods from one typical day rather than several days of averaged data. An even smaller period, such as 1 minute, would better represent the fluctuations in traffic flow. However, if a 1-minute period were used, the simulation length would be reduced to 19 minutes, because only 19 time periods were available with FRESIM. The smallest period that captured traffic fluctuations yet still allowed for a simulation of reasonable duration was 5 minutes; consequently, it was used for all testing sets. The simulated duration was up to 95 minutes. During this time, the demand typically started under capacity and increased to over capacity.

To evaluate the FLC under a variety of conditions, entry volumes and incidents (such as a blocked lane or reduced capacity) were varied to create several testing sets. Results are reported from six representative scenarios. Data set #1 contained only one metered ramp at NE Northgate Way and was a subset of the large study site (Figure 4) used by the other five data sets. Data sets #2 to 6 also included metered ramps at NE

145th Street and NE 175th Street. Data set #2 used entry volumes that were slightly less than freeway capacity, which was typical of actual measurements. Data set #3 used slightly higher entry volumes in which demand was equal to freeway capacity. In data set #4, the demand was slightly greater than capacity. In data set #5, an incident occurred that reduced the capacity of a downstream mainline link by 20 percent. In data set #6, an incident blocked one lane of a downstream mainline link.

The FLC performance was compared to common controllers within FRESIM. These built-in controllers included *clock, demand/capacity, and speed metering.* Clock metering emits vehicles at a constant specified headway, without using any sensor feedback. Demand/capacity metering adjusts the number of vehicles permitted on the basis of how much the demand exceeds capacity. Speed metering determines the metering rate on the basis of interpolation between measured mainline speeds and appropriate metering rates for those speeds. Of all the controllers within FRESIM, speed metering was most similar to Seattle's currently used *Local Metering Rate* (Jacobson, Henry, and Mehyar), which uses occupancy rather than speed as the measure of congestion.

The performance objectives were to maximize total vehicle miles, maximize mainline speeds, and minimize delay per vehicle while maintaining an acceptable ramp queue. An acceptable ramp queue was a constraint because Seattle has limited alternative routes.

For each data set, the FLC was adjusted for best performance through rule weights and fuzzification parameters. Likewise, the parameters of the other controllers, such as capacity or metering rates for given speeds, were adjusted for best performance. For on-

line implementation, however, the researchers will experiment with an *adaptive* controller in which the rule weights will adjust in real time to various traffic conditions such as accidents. This versatility is expected to increase the range of conditions for which ramp metering is beneficial.

DISCUSSION

Simulation Results

Once the FLC was minimized and tuned, it performed better than the other controllers while maintaining an acceptable ramp queue. Through simulation testing, the researchers found that several rules and inputs could be eliminated. Within the original FLC, the number of inputs was reduced from 12 to 7, and the number of rules was reduced from 27 to 19. The mainline volume input and downstream storage rate were eliminated because occupancy and speed were found to be better indicators of congestion. Predicted occupancy eliminated the need for predicted volume. Only two of the four ramp inputs were necessary, one from each of the sensors. The fuzzy classes of the remaining queue occupancy inputs were reshaped for a quicker and smoother transition.

For each of the six data sets described previously, a graph summarizes the relative performance between the controllers (Figures 5 through 10). The x-axis indicates four types of metering. The y-axis shows three performance measures, in order of priority: total miles traveled by all vehicles in the system, average mainline speed, and delay per vehicle. The z-axis graphs the percentage of change between that type of metering and no metering.

In five out of six data sets, the FLC outperformed the other ramp metering types. As Figure 5 shows, the FLC demonstrated the highest speed and least delay, with vehicle-miles similar to those of the other controllers. In the second data set, in which demand was just under capacity, results were nearly identical between controllers except that the

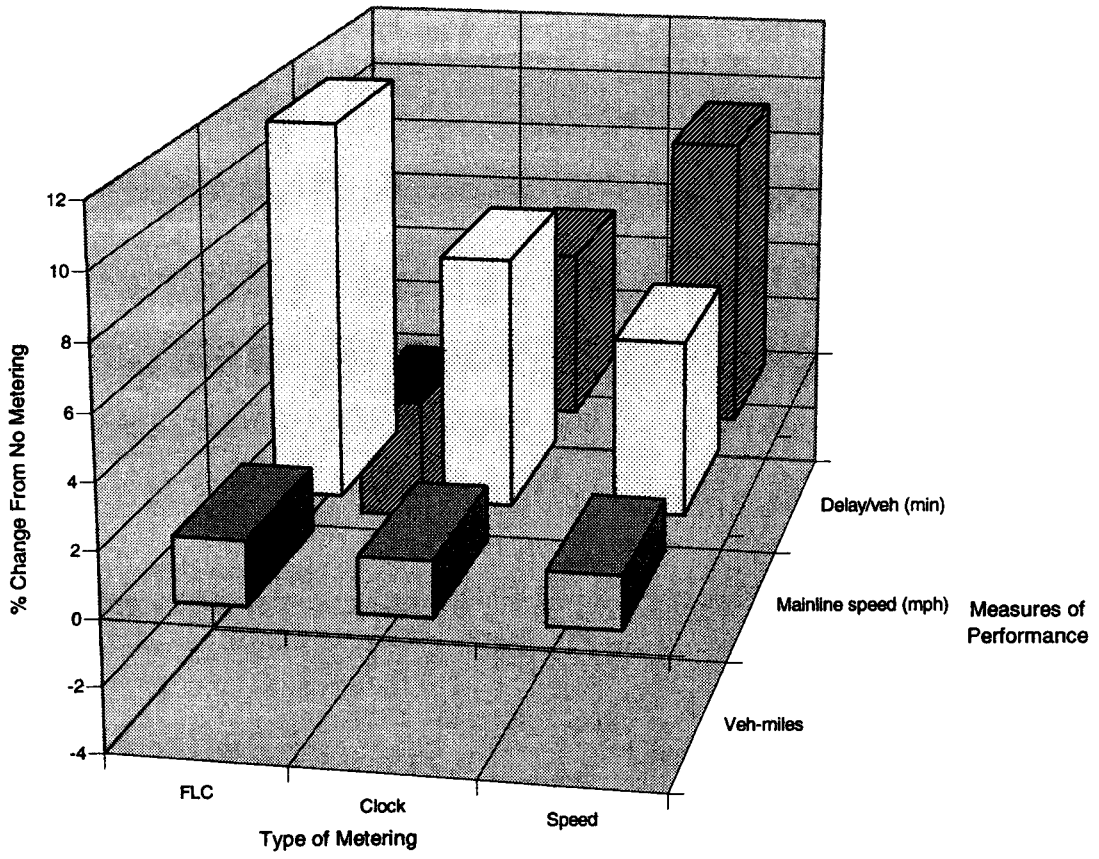

Figure 5. Subset of large data set, contains only one metered ramp

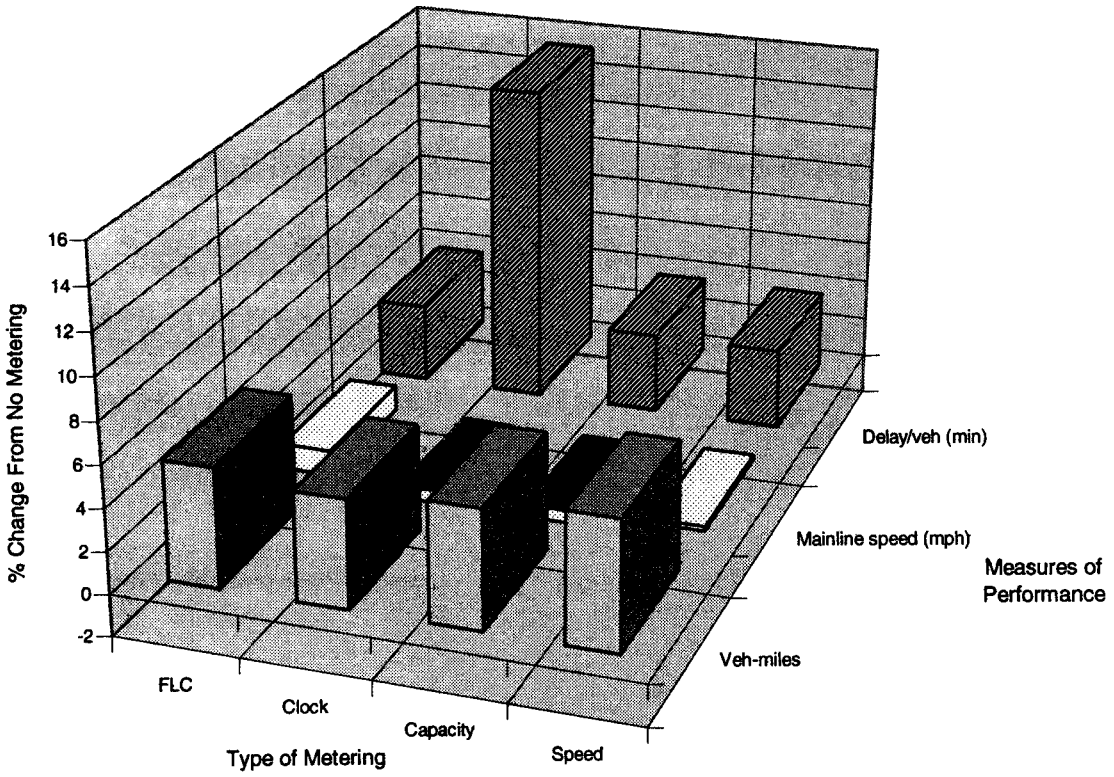

Figure 6. Actual entry volumes where demand ≅ .9 capacity

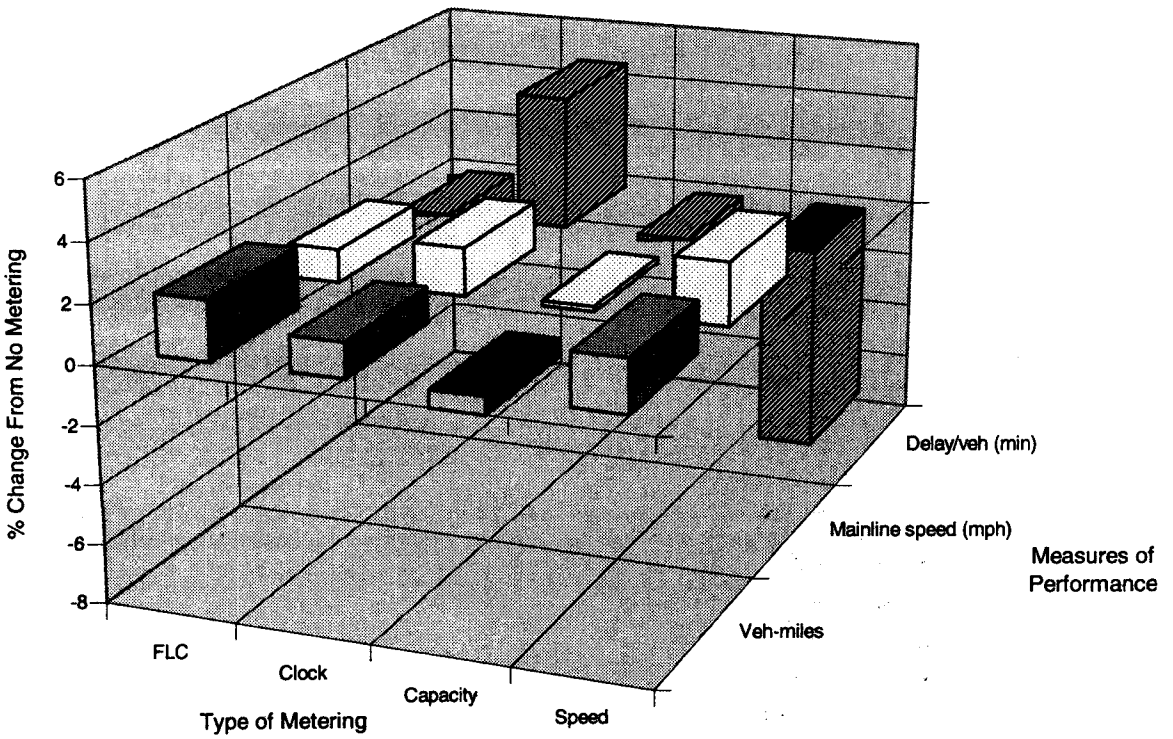

Figure 7. Demand ≅ capacity

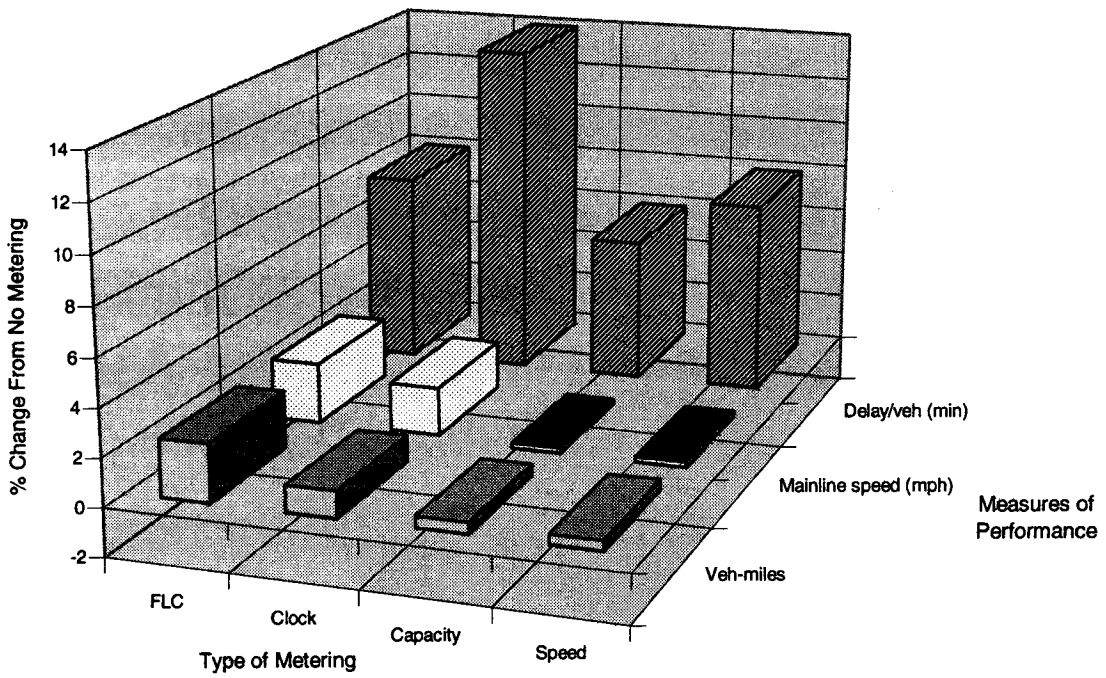

Figure 8. Demand ≅ 1.1 capacity

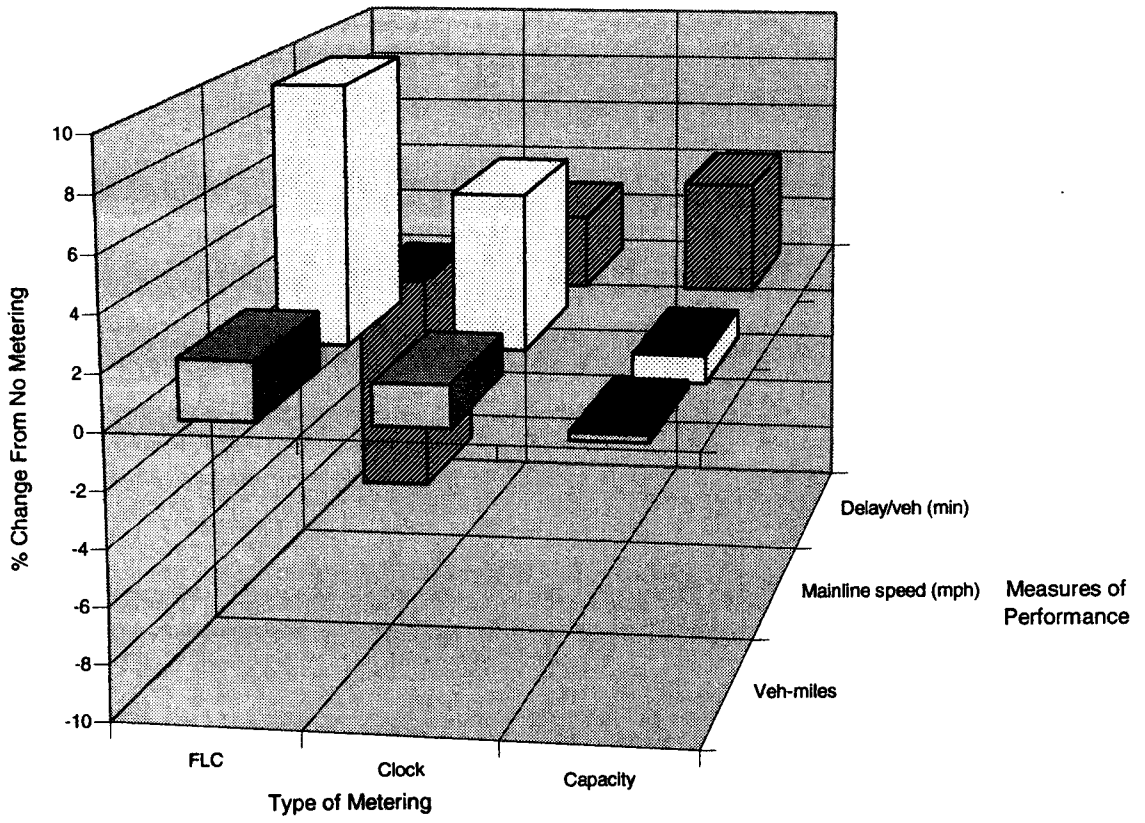

Figure 9. Incident with capacity reduction of 20 percent on downstream mainline link

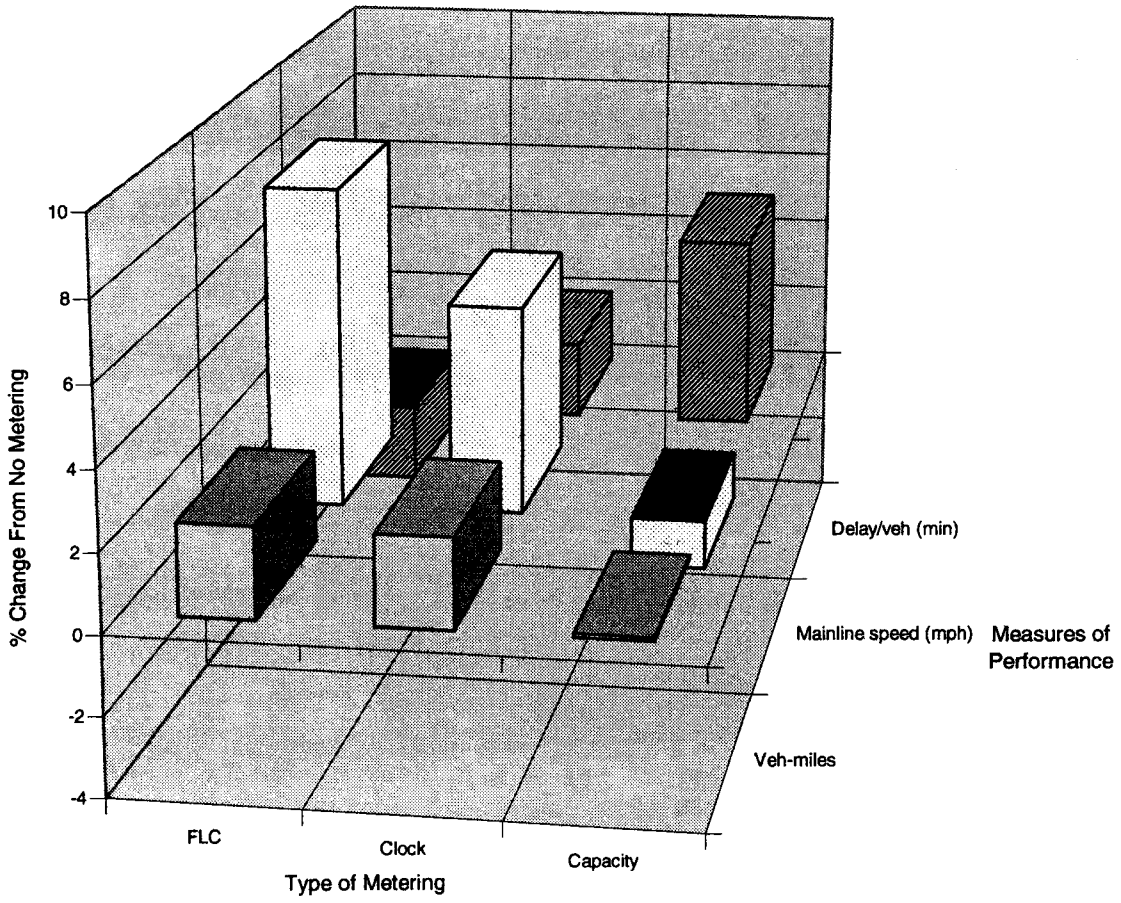

Figure 10. Incident with one-lane blockage on downstream mainline link

FLC did slightly better in all three categories (Figure 6). When demand equaled capacity, speed metering outperformed the fuzzy controller in terms of delay and speeds (Figure 7). When demand just exceeded capacity, the fuzzy controller did best in vehicle-miles and speed (Figure 8). With a 20 percent downstream capacity reduction, the fuzzy controller reacted best (Figure 9). It significantly improved delay and vehicle-miles in comparison to the other controllers, as it did when a lane was blocked downstream (Figure 10).

On-ramp sensors helped the FLC maintain an acceptable ramp queue while preserving mainline efficiency. Because speed and capacity metering did not use ramp inputs, they were liable to demand unreasonable ramp queues under heavy congestion. Speed metering sometimes produced superior performance measures at the expense of an excessive ramp queue. In two of the data sets, speed metering did not meet the ramp queue constraint, in which case the results were disqualified. Likewise, capacity metering did not meet the ramp queue constraint in the first data set. The more that demand exceeded capacity, the more evident was the FLC's advantage in balancing between mainline efficiency and ramp queues. Ramp inputs became increasingly important to meet the demand while stressing vehicle-miles.

Simulation testing uncovered some interesting points. When the FLC was tested on a subset of the large data set (one metered ramp of the three), the specified metering rates were different for a given data set and constant control parameters. In other words, we cannot expect to isolate the effects of a single ramp and generalize those results to the entire system. When real-time inputs are used for control, simulation testing has shown that the specified metering rates depend on each other and the effect they have on the system.

Another relevant observation is that the best controllers restricted the farthest upstream ramp the most. The improved upstream flow propagated down the entire section, while the benefits of restricting downstream ramps were not as noticeable upstream, at least within simulation. This control is the opposite of the heuristic used in Seattle's bottleneck algorithm, in which ramps closest to the downstream bottleneck are targeted for the most vehicle entry reduction. Although this result could be an anomaly of simulation testing because a limited section was modeled and only performance criteria from that section were examined, at the very least it reinforces the belief that system-wide evaluation is necessary.

Simulation Limitations

This section assesses the reliability of the FRESIM model. Software bugs found in FRESIM are discussed, as well as how the limitations of the FRESIM model explain the discrepancy between ramp metering simulation results and on-line studies.

Through simulation testing, several bugs were found in FRESIM. Some of these problems were related to how FRESIM executed the metering rates specified by the controller. The FRESIM programmers improved this logic so that the executed metering rates more closely matched the controller specified metering rates. Other problems included indexing errors, inability to meter multiple ramps for speed and demand/capacity type controllers, and errors in incident creation. The FRESIM programmers corrected these bugs. However, these FRESIM bugs affected the ramp metering simulations, so the simulations had to be recalibrated and rerun with each of the four new versions of FRESIM received. This situation delayed progress by several months, but the significant FRESIM bugs that affected ramp metering are now fixed.

Aside from FRESIM bugs, simulation results suggested that the FLC algorithm will perform better on-line than it did with the FRESIM model. The benefits of ramp metering for conventional controllers are typically higher on-line than in simulation. While numerous on-line studies have reported that metering has improved performance measures 30 to 60 percent in comparison to no metering (see Review of Current Practice), metering in simulation testing has improved performance measures only up to 15 percent, and typically much less than that. Causes for this discrepancy between simulation testing and on-line studies were analyzed because ramp metering algorithms cannot be compared unless ramp metering is an improvement over no metering.

To understand the discrepancy between simulation testing and on-line studies, one must know how ramp metering functions. The principal means by which metering provides benefits is to delay critical flow breakdown. The three ways by which metering can achieve this objective are to (1) divert traffic to alternative routes, (2) spread out demand over a longer time frame, and (3) break up platooning. With few alternative routes in Seattle, the second and third methods are of most interest. The condition under which ramp metering typically benefits on-line is during the transient period from the uncongested to the congested state (for proof, see Zhang, Ritchie, and Recker, 1995). This result agrees with our intuition that metering must begin before bottleneck formation in order to prevent or delay it. In simulation, however, metering is only somewhat helpful under transient conditions and is more helpful under extreme congestion. This contradiction regarding when metering provides benefits, along with the fact that metering provides benefits substantially more on-line than it does in simulation, makes the freeway model suspect.

Several limitations of FRESIM reduce the effectiveness of ramp metering in simulation. One limitation is that vehicles are generated uniformly in simulation. In actual traffic flow, platoon formation is pronounced, and the traffic flow fluctuates rapidly up and down. Without these irregular traffic characteristics, simulated ramp metering does not provide benefits noticeably by the third method, smoothing traffic flow. In actual traffic flow, vehicles arrive at on-ramps in platoons, created in part by arterial signals. In the simulation model, the vehicles are already uniformly spaced, so metering does not provide a benefit by breaking up the platoons to allow more effective merging.

A second limitation of FRESIM is that congestion does not form as easily as it does in actual traffic flow. For a given density level, we expect lower speeds than those produced in simulation. Although several parameters have been adjusted to calibrate the model (such as entry volumes, desired speed, driver sensitivity, driver courtesy, queue discharge delay), the problem lies in the transition from the uncongested to the congested state. Plots of actual occupancy levels on I-5 in Seattle versus 20-second time periods (Nihan, 1995) show a distinct transition from the uncongested to the congested state when the demand exceeds the capacity. While the simulation models the free-flow traffic adequately, a decided transition from uncongested to congested states is not as pronounced, and the link speeds remain overly high as the density level increases. Given that the principal means by which ramp metering improves freeway efficiency is to delay or prevent heavy congestion, we cannot easily demonstrate the effectiveness of ramp metering when this transition is not discernible. Method two of spreading out the demand over time cannot produce the desired improvement without this impending congestion.

Another problem with FRESIM is that it has too much buffer to absorb entering vehicles. In actual traffic flow, vehicles take longer to reach desired speeds under congested conditions. This is one of the reasons that FRESIM produces higher link speeds for given density levels. Because congestion does not form as easily in simulation, there is not as great a need for ramp metering.

The developer of FRESIM believes that the simulation model overestimates the ramp delay caused by metering. In actual traffic flow, vehicles spend time waiting at arterial signals, so the time spent waiting on the ramp is less than that in simulation. If this is the case, ramp metering does not produce as much delay per vehicle as the simulation indicates.

These limitations of FRESIM explain why ramp metering does not improve performance measures to the extent and under the circumstances that it does on-line. The benefits of ramp metering are expected to be more pronounced with on-line testing, although studies are necessary to verify that this is the case for the Seattle freeway system.

Simulation testing was worthwhile despite FRESIM's limitations. Through simulation testing, the rule base was minimized. Default control parameters were determined. The fuzzy controller's versatility and effectiveness were demonstrated. Two prevailing factors made simulation testing worth the effort: uniformity of traffic conditions and availability of performance measures. Unlike on-line testing, the weather conditions and traffic demand do not vary day to day. If one parameter is altered at a time, the simulation will show the effect of that change. Thus, different controllers can be compared, with knowledge that the differences are not due to outside influences. With

on-line testing, obtaining performance criteria such as travel times, link speeds, and delay is not easy. Simulation testing provided all of the statistics necessary to streamline the controller.

APPLICATION AND IMPLEMENTATION

Because results from simulation testing were encouraging, the researchers recommend that implementation and testing on-line be continued as planned. The 1995/96 TransNow and WSDOT proposals for Phase II, by Meldrum and Taylor, describe the implementation tasks that will be performed. The ANN predictors will first be incorporated into the TSMC framework and trained for individual prediction sites. Following ANN implementation, the fuzzy controller will be programmed into the TSMC software. The testing methodology calls for a step-by-step expansion to the entire system.

For on-line testing, performance measures such as total vehicle-miles are not available as in simulation. Instead, to compare ramp metering algorithms, the volumes, occupancies, and estimated speeds will be examined. If possible, initial on-line testing will take place in a section where speed sensors are available. Travel time is highly desirable for comparing algorithms, but it is not available in real time. To estimate it, we will videotape the traffic flow at two observation points and see how long a platoon takes to travel that distance. The TSMC cameras will be helpful in judging ramp delay and overall congestion.

Unlike the uniform traffic conditions in simulation testing, nonuniform weather and traffic demand complicate simulation testing. If FLC algorithm testing happens to fall on less congested days, the FLC algorithm performance may be deceptively better. Hence, on-line algorithm testing must take place over a several-day span to compare the average performance to other algorithms. This approach will minimize the effect of variable traffic conditions.

Few controllers are adaptive in the sense that the control parameters vary with time (i.e., a given input does not always produce the same control response). The researchers will experiment with an adaptive controller in which the rule weights adjust on-line to various traffic conditions such as incidents. This versatility is expected to increase the range of conditions for which ramp metering is beneficial.

An auto on/off is recommended for the ramp metering algorithm, given that ramp metering is beneficial under only particular conditions (see Simulation Limitations for discussion). The ramp metering will not oscillate on and off, but rather, when the ramp metering begins and ends for each peak commute will depend on congestion levels. Operators will have the ability to override ramp metering operation.

ACKNOWLEDGMENTS

The authors would like to thank the system managers at the University of Washington, James Hewitt and Steve Venema, who demonstrated considerable patience in solving both PC and UNIX complications. Jolie Tahara helped us navigate through budget turmoil.

Les Jacobson at WSDOT has been instrumental in obtaining funding for this project. Without his enthusiastic support, this project would not have been possible. We greatly appreciate his assistance. The staff at TRAC and WSDOT have been considerate and helpful throughout the project. Thanks go to Pete Briglia and Larry Senn for their suggestions. Amy O'Brien and John Anzinger assisted in report editing and production. Dina Palas and Mike Swires gave me a jump start tutorial on FRESIM. Mark Morse and Amity Trowbridge provided me with TSMC information. Charlie Higuera supported VAX operations.

At Viggen/IDI, programmers provided essential support for FRESIM modifications. Hassan Halati was quick to devise new possibilities and guide me in the right direction. Jifeng Wu, Steven Chien, and Yanlin Lee made FRESIM source code changes to incorporate the fuzzy controller. These changes to FRESIM would not have been possible without support from the FHWA.

REFERENCES

Chen, L., and May, A. "Freeway Ramp Control Using Fuzzy Set Theory for Inexact Reasoning," *Transportation Research-A*, Vol. 24A, No. 1, 1990, pp. 15-25.

Corcoran, L., and Hickman, G. "Freeway Ramp Metering Effects in Denver," Compendium of Technical Papers, ITE, 1989.

Federal Highway Administration. *FRESIM User Guide*, Version 4.5, Turner-Fairbank Highway Research Center, McLean, Virginia, April 1994.

Fenton, R. "IVHS/AHS: Driving into the Future, "*IEEE Control Systems*, Dec. 1994, p. 13.

Gupta, A. "Controller Design Using Fuzzy Logic (RT/Fuzzy)," Integrated Systems, Inc., 1991.

Havinoviski, G. "Ramp Queues? Not in My Back Yard! A Survey of Queue Detector Design and Operation Criteria for Metered Freeway Entrances, " *Compendium of Technical Papers*, Institute of Transportation Engineers, 1991.

Henry, K., and Mehyar, O. "Six-Year FLOW Evaluation," Washington State Department of Transportation, District 1, Jan. 1989.

Iwasaki, M. "Empirical Analysis of Congested Traffic Flow Characteristics and Free Speed Affected by Geometric Factors on an Intercity Expressway," *Transportation Research Board* 1320, National Research Council, Washington, D.C., 1991, p. 244.

Jacobson, L., Henry, K., and Mehyar, O. "Real-Time Metering Algorithm for Centralized Control," *Transportation Research Board* 1232, National Research Council, Washington, D.C., 1988.

James Kell and Associates. "Before and After Study, Ramp Metering, Eastbound Long Island Expressway, Suffolk County," Technical Memorandum, 1989.

Kostyniuk, L., et al. "An Evaluation of the Detroit Freeway Operations (SCANDI) Project," Michigan State University, 1988.

Kosko, B. *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*, Prentice-Hall, New Englewood Cliffs, New Jersey, pp. 318 and 386, 1992.

Lin, Y., and Lee, T. "Modeling for Fuzzy Logic Control of Deformable Manipulators," Proceedings of the American Controls Conference, San Francisco, June 1993.

Marsden, B. "Ramp Meters and Travel Quality in Austin, Texas," Texas DH&PT, 1981.

D. Masher, etc. *Guidelines for Design and Operation of Ramp Control Systems*, Stanford Research Institute, Menlo Park, California, 1975, p. II-18-II-24.

Meldrum, D., and Taylor, C. "Freeway Traffic Data Prediction Using Artificial Neural Networks and Development of a Fuzzy Logic Ramp Metering Algorithm," Final Technical Report, Washington State Department of Transportation, Olympia, Washington, April 1995.

Nihan, N. "Freeway Congestion Prediction," Joint TransNow/WSDOT Draft Technical Report, Washington State Department of Transportation, Olympia, Washington, March 1995.

Robinson, J., and Doctor, M. "Ramp Metering Status in North America: Final Report," Office of Traffic Operations, Federal Highway Administration, U. S. Department of Transportation, Washington, D.C., Sept. 1989.

Taylor, C. "Freeway Traffic Data Prediction Using Artificial Neural Networks and the Development of a Fuzzy Logic Ramp Metering Algorithm," Master's Thesis, Department of Electrical Engineering, University of Washington, 1994.

Yasunobu, S., and Miyamoto, S. "Automatic Train Operation by Predictive Fuzzy Control," *Industrial Applications of Fuzzy Control*, editor M. Sugeno, pp. 1-18, North-Holland, Amsterdam, 1985.

Zadeh, L. "Fuzzy Sets," *Information and Control* 8, pp. 338-353, 1965.

Zhang, H., Ritchie, S., and Recker, W. "On the Optimal Ramp Control Problem: When Does Ramp Metering Work?" Transportation Research Board Annual Meeting, National Research Council, Washington, D.C., 1995.

APPENDIX A: INTRODUCTION TO FUZZY LOGIC CONTROL

Fuzzy logic allows the use of qualitative knowledge. Rather than forcing a yes or no, on or off response, fuzzy logic utilizes imprecise information such as maybe, occasionally, and probably. Fuzzy set theory, developed by Zadeh in 1965 (Zadeh, 1965), excels in a variety of control applications.

Fuzzy logic control (FLC) is now common in Japan. Since 1987, the FLC Sendai subway designed by Hitachi, LTD (Yasunobu and Miyamoto, 1985) stops three times more accurately in position than a manually controlled train and has an exceptionally smooth ride. An FLC air conditioner reduced Mitsubishi Heavy Industries' power consumption by 20 percent. Of the washing machines produced by Matsushita Electrical Industrial, over 70 percent are fuzzy controlled. Canon's H800 camera uses FLC auto focusing. With fuzzy logic's widespread applicability, development ease, and cost effectiveness, its popularity will continue to grow in the United States.

FLC involves four main steps: fuzzification, rule evaluation, implication, and defuzzification. There are many variations on how to implement an FLC. The techniques that are most appropriate depend on user preference and the specific application. This section discusses the most common FLC techniques.

Fuzzification

The fuzzification process translates each precise input into a set of fuzzy variables defined by membership classes, also called membership functions. A membership function describes, on a scale of 0 to 1, the degree to which an input belongs to that set. Membership functions can be discrete or continuous, and triangles or bell-shaped curves commonly define them. Figure A-1 shows membership classes appropriate for the

storage rate, that is, the number of vehicles entering a section minus the number of vehicles exiting a section during the past minute. Each fuzzy variable represents a class:

NB	negative big
NS	negative small
ZE	zero
PS	positive small
VB	positive big

If the numerical, or crisp, storage rate input is 12 vehicles/minute, then the fuzzy PS degree is 0.6, and the PB degree is 0.2, with the remaining classes zero. If the crisp input is less than -20 vehicles/minute, the NB membership function is 1, and if the crisp input is greater than 20 vehicles/minute, the PB membership function is 1. If more membership classes are added, such as negative medium and positive medium, the

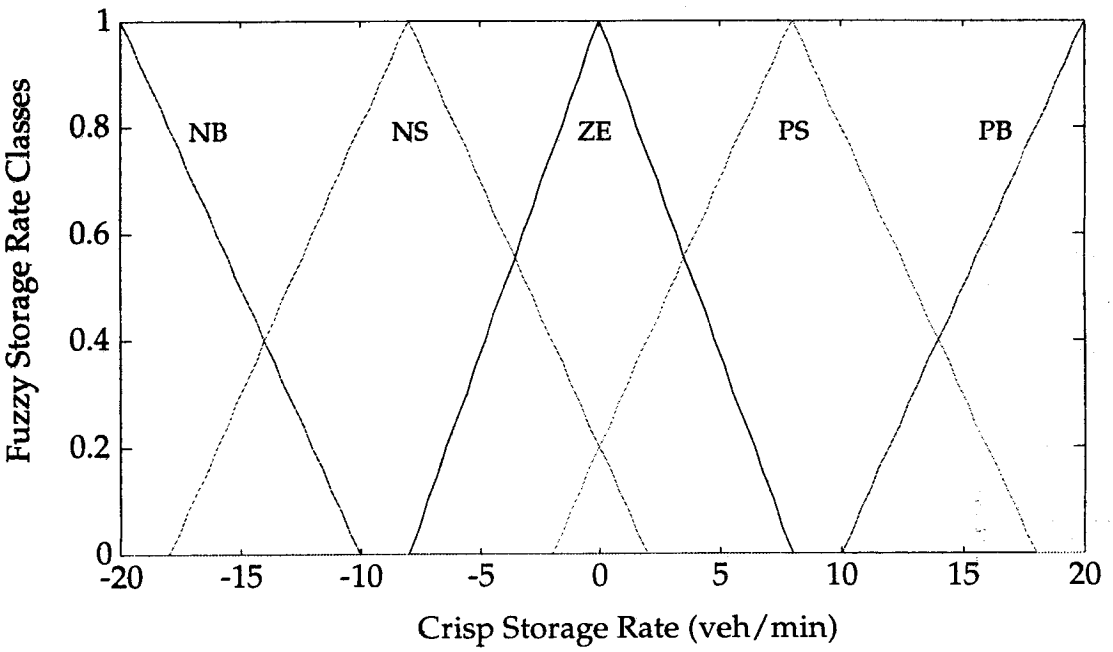

Figure A-1. Storage rate membership classes rules

triangular bases can be narrowed. With fewer membership classes, more overlap is needed. The best percentage of overlap between classes depends on the specific application. Researchers recommend between 25 percent overlap (Kosko, 1992) and 75 percent overlap (Lin and Lee, 1993; Gupta, 1991).

Rules

The rules, sometimes called the knowledge base, are the heart of a fuzzy logic controller. Rules are based on expert opinions, operator experience, and system knowledge. For the fuzzy logic ramp metering algorithm, the existing ramp metering algorithm provides a starting point for rule development. Rule evaluation, based on fuzzy set theory, uses fuzzy operators to perform logical operations such as the complement, intersection, and union of sets. The complement corresponds to one minus the membership degree in fuzzy set theory. An AND operation, analogous to the intersection of sets, takes the minimum of given membership degrees. An OR operation, analogous to the union of sets, takes the maximum of given membership degrees.

A simplified example for a fuzzy logic ramp metering algorithm demonstrates rule evaluation. Suppose a knowledge base includes the following rules:

Rule 1: If downstream storage rate is positive big and downstream occupancy is big, then metering rate is small.

Rule 2: If queue override occupancy is big and queue duration is big, then metering rate is small.

Rule 3: If upstream occupancy is small, then metering rate is big.

Rule 4: If predicted occupancy is small, then metering rate is big.

Table A-1. Variable Descriptions for FLC Example

Variable	Description
SR	Downstream storage rate
DO	Downstream occupancy
QO	Ramp queue occupancy
QD	Ramp queue duration
UO	Upstream occupancy
PO	Predicted mainline occupancy
MR	Ramp metering rate

These rules can be rewritten more compactly using the variables in Table A-1. The variables in parentheses below represent the qualifying conditions, and the number in brackets is a hypothetical membership degree. This conditional pair is followed by the output MR to the degree in brackets:

Rule 1: If (SR_PB [.5], DO_B [.7]) then MR_S [.5]

Rule 2: If (QO_B [.6], QD_B [.2]) then MR_S [.2]

Rule 3: If (UO_S [.3]) then MR_B [.3]

Rule 4: If (PO_S [.2]) the MR_B [.2]

Two methods for further reducing rules with similar outputs are the *maximum* and *additive* methods. The *maximum* method of rule deduction takes the maximum MR_B degree of rules 1 and 2, since this corresponds to a union of the two output sets. Using the maximum method, rules 1 and 2 further reduce to

Composite Rule 1 and 2:

If (SR_PB [.5], DO_B [.7]) OR (QO_B [.6], QD_B [.2])
then MR_S [.5]

Similarly, the maximum method combines Rules 3 and 4 to

Composite Rule 3 and 4:

If (UO_S [.3]) OR (PO_S [.2]), then MR_B [.3]

The *additive* method adds the two output degrees together. With this method, rules 1 and 2 produce MR_S [.7], and rules 3 and 4 produce MR_B [.5]. These two rule deduction methods produce different results, and the one that is most appropriate depends on the application. For the ramp metering application, the additive method was chosen over the minimum method because it is less sensitive to faulty loop detector data. While the maximum method may "choose" the faulty value because it is the most extreme, the additive method uses each rule contribution.

Implication

Implication expresses the area that an output variable class activates for use in the defuzzification calculation. Two common implication mechanisms are *correlation-minimum* encoding and *correlation-product* encoding. The *correlation-minimum* method uses the *min* operator,

$$\min(w, f(x)),$$

which simply cuts off the class, $f(x)$, at the output degree, w. The *correlation-product* encoding method scales the output area by the output degree:

$$w*f(x)$$

Figure A-2 demonstrates these two implication methods graphically for an output degree of 0.5. The shaded area represents the implicated area. These two methods may produce different results, but correlation-product implication can make defuzzification easier. For this reason, correlation-product implication is used for the ramp metering application.

Defuzzification

The defuzzification process produces a crisp output given a fuzzy output variable set. The commonly used centroid method finds the crisp output by dividing the sum of the implicated areas into two equal areas:

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

Figure A-3 illustrates the centroid method for the previous ramp metering example. Using the correlation-minimum inference mechanism, the sum of the MR_S [.5] and MR_B [.3] implicated areas produces a crisp metering rate of 6 vehicles/minute.

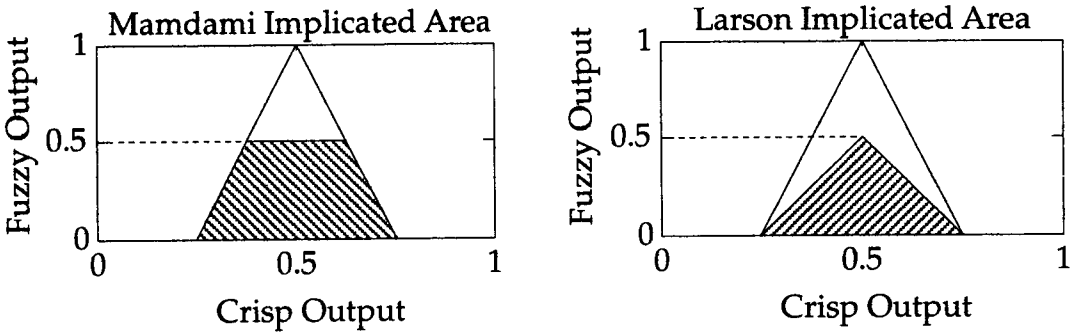

Figure A-2. Correlation-minimum (left) and correlation-product implication (right)

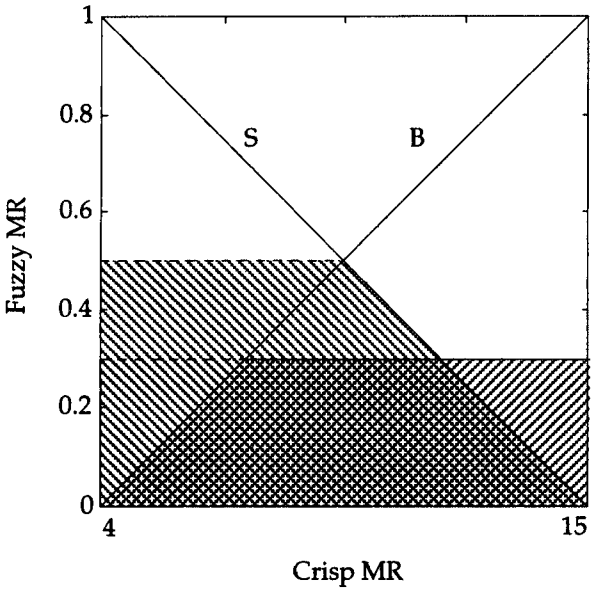

Figure A-3. Centroid defuzzification

In practice, a discrete fuzzy centroid is easier to calculate than the above continuous centroid equation. For the case in which correlation-product inference is used, the following discrete centroid equation is equivalent to the continuous centroid equation (see Kosko, 1992 for proof). The crisp output is found by

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

where c_i is the centroid and I_i is the area of the output class for the ith rule. This technique is used in the ramp metering application.

APPENDIX B: FUZZY CONTROLLER CODE

```
/*
** FUNCTION
**     linfind
**
** USAGE
**     lut_p = (struct Lut *)linfind(
**                     tok1,
**                     &rule_table,
**                     &i_tmp,
**                     sizeof(struct Lut),
**                     keycmp);
**
** DESCRIPTION
**          Performs a linear search for KEY in BASE which has *NMEMB elements
**          of SIZE bytes each.  The comparisons are done by (*COMPAR)(), which
**          must return zero if the elements being compared are equal, and
**          non-zero otherwise.
**          Returns NULL if not found
**
** CALLS FROM
**          reader
**
** CALLS TO
**          none
**
** ARGUMENTS
**     tok1: points to datum to be found
**     &rule_table: points to first element in table
**     &i_tmp:          points to integer containing current number of table elements
**          sizeof(struct Lut): size of table element in bytes
**     keycmp:          name of comparison function which user supplies
**
** RETURN VALUE
**     lut_p:  pointer into table indicating where datum can be found
**
** GLOBALS ACCESSED
**     none
**
** RESTRICTIONS
**     none
**
** BUGS
**     none
**
** FUTURE DIRECTIONS
**
** REVISION
**     $Revision: 2.0$
**
*/

/*
* RCS Identifiers:
*
*     $Source: taylorc/fuzzy/traffic/newstuff/linfind.c $
*     $Revision: 2.0 $
*     $Date: 11/23/94 $
*     $Author:  $Id: lfind.c,v 1.1 1994/05/21 13:23:03 chris Exp $
*                     Portions of lfind code were borrowed from GCC libraries above
*                     and edited by Neil Friedman $
```

```
*    $State: Exp $
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "linfind.h"

char *linfind (CONST PTR key, CONST PTR base,
      size_t *pnmemb,
      size_t size,
      int (*compar)(CONST PTR, CONST PTR))
{
 register size_t i;
 register CONST PTR p;

 for(i = 0, p = base;
    i < *pnmemb;
    p = (PTR) (((CONST char *) p) + size), i++)
 {
    if ( !((*compar)(key, p)))      /* match found */
     return (PTR) p;
 }

 return NULL;                                    /* match not found */
}



/*
** FUNCTION
**     reader
**
** USAGE
**     reader(filename, parameters, weights);
**
** DESCRIPTION
**          Reads fuzzification parameters and rule weights from
**          file ramp*.inp.
**
** CALLS FROM
**          flc
**
** CALLS TO
**          linfind
**
** ARGUMENTS
**     filename:  pointer to character array for filename
**
** RETURN VALUE
**     parameters: a 2-dimensional array of fuzzication
**          parameters that specifies the fuzzy membership class
**          for each input. The first dimension indicates
**          which input, and the second dimension indicates
**          which parameter. These parameters are specified by
**          user in ramp*.inp. Parameters are in order of
**          LL, HL, C_S, C_M, C_B, B_VS, B_S, B_M, B_B, B_VB
**          weights: An array of rule weights. These are specified
**          in ramp*.inp to emphasize or eliminate rules.
**
** GLOBALS ACCESSED
**     none
**
```

```
** RESTRICTIONS
**      none
**
** BUGS
**      none
**
** FUTURE DIRECTIONS
**
** REVISION
**      $Revision: 2.0$
**      Modifications from previous version:
**              1) linfind.c is an included file to replace library
**              function lfind.
**
**              $Revision:  2.1$
**              1) (void *) was added in front of parameter passed in
**              linfind call (lines 189 and 201)  to avoid
**              incompatible pointer type.
*/

/*
 * RCS Identifiers:
 *
 *      $Source: taylorc/fuzzy/traffic/newstuff/reader.c $
 *      $Revision: 2.0 $
 *      $Date: 11/23/94 $
 *      $Author: steve venema $
 *      $State: Exp $
 */

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "linfind.h"

#define MAX_LINELEN             100     /* Maximum # of characters/line         */
#define N_PARAMETERS            13      /* Number of elements in parameters[x]  */
#define N_P_COLS  10            /* Number of columns per parameter      */
#define N_RULES                 26      /* Number of elements in rules[x]       */
#define EX_ERROR  1             /* Error exit code      */
#define EX_OK                   0       /* No-error exit code  */

#ifndef TRUE
# define TRUE       1
# define FALSE      0
#endif

struct Lut {
        char        *key;       /* Lookup-table key   */
        int         index;      /* table index                  */
};

struct Lut  parm_table[N_PARAMETERS] = {
        { "VO", 0},
        { "OC", 1},
        { "DO", 2},
        { "UO", 3},
        { "PO", 4},
        { "SP", 5},
        { "DS", 6},
        { "SR", 7},
        { "QO", 8},
```

```
        { "QD", 9},
        { "AQO", 10},
        { "AQD", 11},
        { "MR", 12}
};

struct Lut  rule_table[N_RULES] = {
        { "R1",  0},
        { "R2",  1},
        { "R3",  2},
        { "R4",  3},
        { "R5",  4},
        { "R6",  5},
        { "R7",  6},
        { "R8",  7},
        { "R9",  8},
        { "R10", 9},
        { "R11", 10},
        { "R12", 11},
        { "R13", 12},
        { "R14", 13},
        { "R15", 14},
        { "R16", 15},
        { "R17", 16},
        { "R18", 17},
        { "R19", 18},
        { "R20", 19},
        { "R21", 20},
        { "R22", 21},
        { "R23", 22},
        { "R24", 23},
        { "R25", 24},
        { "R26", 25}
};

const char *white_space = " \t\n";

static int
keycmp(key, item)
        char                *key;
        struct Lut  *item;
{
        return(strcmp(key,item->key));
}

int reader(
        char *fname,                            /* Input filename               */
        float (*parameters)[N_P_COLS],          /* -> Place to put parameters   */
        float *weights)                         /* -> Place to put weights      */
{
        FILE *fpointer;     /* -> Input file stream                             */
        int     i_tmp;      /* temporary integer                                */
        int     is_param;   /* TRUE: Found a parameter      */
        int     index;      /* Array index for parameters/weights               */
        char    in_line[MAX_LINELEN+1];     /* Buffer for input line from file */
        char    *tok1;              /* 1st token in line                        */
        char    *tok_p;             /* Subsequent tokens in line                */
        int     cnt, i;             /* Loop counters                            */
        float   f_val;              /* Tmp float value                          */
        struct Lut *        lut_p;      /* -> to lookup table                   */
        int flag=0;             /* flag=1 if insufficient rule base */

        /*
         * Open the specification file
```

```
 */
if ((fpointer=fopen(fname, "r"))== NULL) {
        perror("Can't open input file");
        exit(EX_ERROR);
}

/*
 * Loop, reading in each line in the file until EOF
 */
while (fgets(in_line, MAX_LINELEN, fpointer) != NULL) {
        /*
         * Grab first token from input line
         */
        tok1 = strtok(in_line, white_space);
        if (tok1 == NULL) {
                /*
                 * No first token on this line
                 */
                continue;
        }

        /*
         * See if first token matches one of the tokens in the LUT's
         */
        i_tmp = N_PARAMETERS;
        lut_p = (struct Lut *)linfind(
                        tok1,
                        (void *)&parm_table,
                        &i_tmp,
                        sizeof(struct Lut),
                        keycmp);
        if (lut_p != NULL) {
                is_param = TRUE;
                index = lut_p->index;
        }
        else {
                /*
                 * No match in parameter table...try rule_table
                 */
                i_tmp = N_RULES;
                lut_p = (struct Lut *)linfind(
                        tok1,
                        (void *)&rule_table,
                        &i_tmp,
                        sizeof(struct Lut),
                        keycmp);
                if (lut_p == NULL) {
                        /*
                         * No match in either table...go to next line
                         */
                        continue;
                }
                index = lut_p->index;
                is_param = FALSE;
        }

        /*
         * So now "index" is the array index into the parameter/rule
         * table and "is_param"==TRUE indicates that you have a
         * parameter line (otherwise you have a rule line)
         */

        if (is_param) {
                /*
```

```
                    * Process a parameter line
                    */
                    for (cnt = 0; cnt < N_P_COLS; cnt++) {
                            tok_p = strtok(NULL, white_space);
                            if (tok_p == NULL) {
                                    /*
                                     * Line is too short!! Missing parameter
                                     */
                                    fprintf(stderr,
                                            "Missing  percentd parameters for  percents spec\n",
                                            N_P_COLS - cnt,
                                            lut_p->key);
                                    exit(EX_ERROR);
                            }
                            f_val = atof(tok_p);
                            parameters[index][cnt] = f_val;
                    }
            }
            else {
                    /*
                     * Process a rule (e.g., weight) line
                     */
                    tok_p = strtok(NULL, white_space);
                    if (tok_p == NULL) {
                            /*
                             * Line is too short!! Missing parameter
                             */
                            fprintf(stderr,
                                    "Missing rule weight for rule  percents\n",
                                    lut_p->key);
                            exit(EX_ERROR);
                    }
                    f_val = atof(tok_p);
                    weights[index] = f_val;
            }
} /* End while(foreach-line) */

fclose(fpointer);

/* test parameters */
/* test to make sure weights are positive */
for (i=0; i<N_RULES; i++)
            if (weights[i]<0.0)
                    printf("Weights must be>0\n
                            percentdth weight=\n", i, weights[i]);

/* test to make sure sufficient rule base */
for (i=0; i<5; i++)
            if (weights[i]==0.0)
                    flag=1;
for (i=7; i<12; i++)
            if (weights[i]==0.0 && flag==1) {
                    printf("Rule base not sufficient.\n");
                    printf("Must add rules 1-5 or rules 8-12\n");
            }
for (i=0; i<N_PARAMETERS; i++) {
            if (parameters[i][1]-parameters[i][0]<0.01) {
                    printf("HL-LL must be>0.01\n");
                    printf("HL-LL= percent7.4f\n", parameters[i][1]-parameters[i][0]);
            }
            /* test to make sure centroids are increasing and between (0,1) */
            if (parameters[i][2]>parameters[i][3] || parameters[i][3]>parameters[i][4]) {
                    printf("Class centroids for an input must be in increasing order.\n");
                    printf("Centroids for  percentdth input=[ percent7.4f,  percent7.4f,  percent7.4f]\n",
```

```
                                        i, parameters[i][2], parameters[i][3], parameters[i][4]);
                        }
                        for (cnt=2; cnt<5; cnt++)
                                if (parameters[i][cnt]<=0.0 || parameters[i][cnt]>=1.0) {
                                        printf("Centroid for classes must be between (0,1).\n");
                                        printf("Centroids for  percentdth input,  percentdth class= percent7.4f\n",
                                                i, cnt-2, parameters[i][cnt]);
                                }
                        /* test to make sure positive base width */
                        for (cnt=5; cnt<10; cnt++)
                                if (parameters[i][cnt]<=0.0) {
                                        printf("Base width for classes must be positive.\n");
                                        printf("Base width for  percentdth input,  percentdth class= percent7.4f\n",
                                                i, cnt-5, parameters[i][cnt]);
                                }
                }
}



#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#define N_PARAMETERS 13   /* first dim. in parameter */
#define N_P_COLS 10       /* second dim. in parameter  */
#define N_INPUTS 12       /* number of inputs to controller */
#define N_RULES  26       /* number of control rules */
#define N_CLASSES 5       /* number of fuzzy classes */

/* reader.c is an external file to be accessed by flc */
extern int reader(char *fname, float (*parameters)[N_P_COLS], float *weights);

/*
** FUNCTION
**      fuzzify:  fuzzifies input variables
**
** USAGE
**      fuzzify(finputs, ith_input, parameters);
**
** DESCRIPTION
**      Given an array of input variables for a particular ramp,
**      and the fuzzification parameters for that ramp, it returns
**      a set of fuzzy classes for each crisp input
**
** CALLS FROM
**      flc
**
** CALLS TO
**      none
**
** ARGUMENTS
**      ith_input:  an array of crisp input variables for
**                  a particular ramp
**      parameters:  a 2-dimensional array of fuzzication
**                  parameters that specifies the fuzzy membership class
**                  for each input.  The first dimension indicates
**                  which input, and the second dimension indicates
**                  which parameter.  These parameters are specified by
**                  user in ramp*.inp.  Parameters are in order of
**                  LL, HL, C_S, C_M, C_B, B_VS, B_S, B_M, B_B, B_VB.
**
** RETURN VALUE
**      finputs:  a 2-dim. array of fuzzy input classes.  The first
```

```
**                  dim. indicates which input, and the second dim.
**                  indicates which class.  The set of membership
**                  classes consists of VS, S, M, B, VB.
**
** GLOBALS ACCESSED
**          N_CLASSES, N_P_COLS, N_INPUTS
**
** RESTRICTIONS
**          none
**
** BUGS
**          none
**
** FUTURE DIRECTIONS
**
** REVISION
**          $Revision: 2.0$
**          No modifications from previous version
*/

/*
 * RCS Identifiers:
 *
 *          $Source: taylorc/fuzzy/traffic/newstuff/flc.c $
 *          $Revision: 2.0 $
 *          $Date: 11/23/94 $
 *          $Author: cyndi $
 *          $State: Exp $
 */

void fuzzify(float (*finput)[N_CLASSES], float *input, float (*parameter)[N_P_COLS]) {
          float crisp;/* input normalized to 0-1 range */
          float LL, HL;          /* low and high limit of crisp input */
          float centroid;        /* centroid for S, M, B classes */
          float base; /* base width of right triangle to define class */
          int i, k;    /* counters */

          for (i=0; i<N_INPUTS; i++) {              /* do for each input */
                    LL=parameter[i][0];             /* initilize LL */
                    HL=parameter[i][1];             /* initilize HL */
                    /* make sure that HL greater than LL */
                    if (HL-LL<0.001)
                              printf("Warning:  Division by small number, HL-LL= percent7.4f\n", HL-LL);
                    crisp=input[i]/(HL-LL)-LL/(HL-LL);       /* normalize to 0-1 range */
                    for (k=0; k<N_CLASSES; k++) {                    /* do for each class */
                              base=parameter[i][5+k];                /* initilize base */
                              if (k==0) {           /* VS (very small) class */
                                        if (crisp<0.0)
                                                  finput[i][k]=1.0;
                                        else if (crisp<base)
                                                  finput[i][k]=-1/base*(crisp-base);
                                        else
                                                  finput[i][k]=0.0;
                              }
                              else if (k==4) {      /* VB (very big) class */
                                        if (crisp>1.0)
                                                  finput[i][k]=1.0;
                                        else if (crisp>1-base)
                                                  finput[i][k]=1/base*(crisp-1+base);
                                        else
                                                  finput[i][k]=0.0;
                              }
                              else {                              /* S, M, B classes */
                                        centroid=parameter[i][k+1];      /* init. centroid */
```

```
                                        if (crisp>centroid-base && crisp<centroid)
                                                finput[i][k]=1/base*(crisp-centroid+base);
                                        else if (crisp>=centroid && crisp<centroid+base)
                                                finput[i][k]=-1/base*(crisp-centroid-base);
                                        else
                                                finput[i][k]=0.0;
                                }
                        }       /* end of for each class */
                }       /* end of for each input */
}       /* end of fuzzify */


/* min returns the minimum of two numbers */
float min(float y1, float y2)
{
        float minimum;

        if (y2<y1)
                minimum=y2;
        else
                minimum=y1;
        return minimum;
}


/*
** FUNCTION
**      rules--evaluates rule base
**
** USAGE
**      rules(control, fmr, finputs, weights);
**
** DESCRIPTION
**      Given fuzzy inputs and weights for each rule,
**      this module evalutes each control rule. The rule outcomes
**      are aggregated in a fuzzy metering rate, which is returned.
**      The results are written to control.txt for tuning purposes.
**
** CALLS FROM
**      flc
**
** CALLS TO
**      min
**
** ARGUMENTS
**      control: The pointer to the output file control.txt
**      finputs: A two dim. array of fuzzy inputs. The first
**               dim. indicates which input, and the second
**               dim. indicates which fuzzy class.
**      weights: An array of rule weights. These are specified
**               in ramp*.inp to emphasize or eliminate rules.
**
** RETURN VALUE
**      fmr: An array of aggregate rule outcomes, one element
**           for each fuzzy metering rate class
**
** GLOBALS ACCESSED
**      N_CLASSES, N_RULES
**
** RESTRICTIONS
**      none
**
** BUGS
**      none
```

```
**
** FUTURE DIRECTIONS
**
** REVISION
**          $Revision: 2.0$
**          Modifications from previous version:
**          1) An additional argument, control, is passed.
**          2) Control.txt is opened in the calling module flc
**          instead of reopened again in rules.
**
**          $Revision:  2.1$
**          1) Only print "\n"  to control.txt every 5
**          rule outcomes (lines 218 and 219)
**
**          $Revision:  2.2$
**          1) Change rule #23 from "If QD_PB then MR_PS" to "If QD_PB then MR_PB"
**          2) Change format of control.txt so it shows rule_weight*rule_outcome instead
**          of just rule_outcome.
*/
/*
 * RCS Identifiers:
 *
 *          $Source: taylorc/fuzzy/traffic/newstuff/flc.c $
 *          $Revision: 2.2 $
 *          $Date: 6/13/95 $
 *          $Author: cyndi $
 *          $State: Exp $
 */

void rules(FILE *control, float *fmr, float (*finput)[N_CLASSES], float *w) {
          int i, k;

          /* Evaluate each rule. */
          float rule[N_RULES]={finput[1][4], finput[1][3], finput[1][2], finput[1][1],
                              finput[1][0], finput[4][4], finput[4][0], finput[3][4],
                              finput[3][3], finput[3][2], finput[3][1], finput[3][0],
                              min(finput[5][0], finput[1][4]), finput[5][1], finput[5][3],
                              min(finput[5][4], finput[1][0]), min(finput[7][4], finput[2][4]),
                              min(finput[6][0], finput[2][4]), min(finput[6][1], finput[2][3]),
                              min(finput[6][2], finput[2][2]), min(finput[6][3], finput[2][1]),
                              min(finput[6][4], finput[2][0]), finput[8][4], finput[9][4],
                              finput[10][4], finput[11][4]};

          k=0;      /* init. counter */
          /* if rule outcome greater than zero, print outcome to file */
          for (i=0; i<N_RULES; i++) {
                    if (rule[i]>0.0) {
                              if (k percent4==0 && k!=0)
                                        fprintf(control, "\n");
                              k++;                    /* count number of rules fired */
                              fprintf(control, " percent3.1f*rule[ percent2d]= percent4.2f, ", w[i],i+1,rule[i]);
                    }
          }
          fprintf(control, "\n");

          /* calc. fuzzy metering rate for each class--
          ** the weighted sum of each rule outcome */
          fmr[0]=w[0]*rule[0]+w[5]*rule[5]+w[7]*rule[7]
                    +w[12]*rule[12]+w[16]*rule[16]+w[17]*rule[17];
          fmr[1]=w[1]*rule[1]+w[8]*rule[8]+w[13]*rule[13]+w[18]*rule[18];
          fmr[2]=w[2]*rule[2]+w[9]*rule[9]+w[19]*rule[19];
          fmr[3]=w[3]*rule[3]+w[10]*rule[10]+w[14]*rule[14]
                    +w[20]*rule[20]+w[22]*rule[22];
          fmr[4]=w[4]*rule[4]+w[6]*rule[6]+w[11]*rule[11]
```

```
                +w[15]*rule[15]+w[21]*rule[21]+w[23]*rule[23]+w[24]*rule[24]+w[25]*rule[25];

        /* print aggregate consequent to a file for tuning purposes */
        fprintf(control, "fmr={ percent6.3f, percent6.3f, percent6.3f, percent6.3f, percent6.3f}\n",
                fmr[0], fmr[1], fmr[2], fmr[3], fmr[4]);

}

/*
** FUNCTION
**      defuzz--calc. metering rate
**
** USAGE
**      MR=defuzz(fmr, parameters);
**
** DESCRIPTION
**      Given fuzzy metering rates and fuzzification
**      parameters, it calculates the crisp metering rate.
**
** CALLS FROM
**      flc
**
** CALLS TO
**      none
**
** ARGUMENTS
**      fmr:  fuzzy metering rates (aggregate rule outcomes)
**      parameters:  a 2-dimensional array of fuzzication
**              parameters that specifies the fuzzy membership class
**              for each input.  The first dimension indicates
**              which input, and the second dimension indicates
**              which parameter.  These parameters are specified by
**              user in ramp*.inp.  Parameters are in order of
**              LL, HL, C_S, C_M, C_B, B_VS, B_S, B_M, B_B, B_VB.
**              defuzz only uses the last array, which is for metering rate.
**
** RETURN VALUE
**      MR:  given in vehicles/20 secs (before returning to FRESIM,
**              this is converted to headway in secs.  flc.c is
**              independent of the time period except that
**              the user specified parameter limits for MR are
**              in vehicles/20 secs for ramp*.inp )
**
** GLOBALS ACCESSED
**      N_P_COLS, N_INPUTS
**
** RESTRICTIONS
**      none
**
** BUGS
**      none
**
** FUTURE DIRECTIONS
**
** REVISION
**      $Revision: 2.0$
**      No modifications from previous version
*/

/*
 * RCS Identifiers:
 *
 *      $Source: taylorc/fuzzy/traffic/newstuff/flc.c $
 *      $Revision: 2.0 $
```

```
*       $Date: 11/23/94 $
*       $Author: cyndi $
*       $State: Exp $
*/

float defuzz(float *fmr, float (*parameter)[N_P_COLS]) {
        float output;           /* control action--the metering rate */
        float base; /* base width of the right triangle to define class */
        float area; /* area of the class */
        float centroid;         /* centroid of the class */
        float num=0.0;
        float den=0.0;
        float LL=parameter[N_INPUTS][0];        /* low limit for metering rate */
        float HL=parameter[N_INPUTS][1];        /* high limit for metering rate */
        int i;

        /* Find areas and centroids of each fuzzy class of MR */
        for (i=0; i<N_CLASSES; i++) {
                base=parameter[N_INPUTS][i+5];
                if (i==0) {
                        area=1.0/2.0*base;
                        centroid=1.0/3.0*base;
                }
                else if (i==4) {
                        area=1.0/2.0*base;
                        centroid=1-1.0/3.0*base;
                }
                else {
                        area=base;
                        centroid=parameter[N_INPUTS][i+1];
                }
                num+=fmr[i]*area*centroid;
                den+=fmr[i]*area;
        }

        /* Check for division by zero */
        if (den<0.01) {
                printf("Warning:  division by small number\n");
                printf("den= percent7.4f\n");
                printf("May have insufficient rules firing\n");
        }
        if (HL-LL<0.01) {
                printf("Warning:  division by small number\n");
                printf("HL-LL= percent7.4f\n", HL-LL);
                printf("Need to increase range limit\n");
        }
        /* calculate metering rate and rescale to (LL, HL) range */
        output=(HL-LL)*(num/den+LL/(HL-LL));
        /* Check for output outside allowable range */
        if (output<LL || output>HL)
                printf("Warning: Metering rate is outside allowable range\nMR= percent7.4f\n", output);
        return output;
}

/*
** PROGRAM
**      flc--fuzzy logic controller
**
** USAGE
**       flc(&N_RAMPS, HW, inputs);
**
** DESCRIPTION
**      flc returns an array of headway control rates in secs. for each
**      metered ramp when given input variables from FRESIM and the
```

```
**          number of metered ramps
**
** CALLS FROM
**          getmr (in future, will be called from FRESIM instead)
**
** CALLS TO
**          reader, fuzzify, rules, defuzz
**
** ARGUMENTS
**          &N_RAMPS:  pointer to an integer that is number of metered ramps.
**          inputs:    a two dim. array containing input variables from FRESIM.
**                     The first dim. indicates which ramp, and the second dim.
**                     indicates which variable.  Variables are in order of
**                     VO, OC, DO, UO, PO, SP, DS, SR, QO, QD, AQO, AQD, MR.
**                     (see future directions below)
**
** RETURN VALUE
**          HW:  an array of control rates given in seconds headway--one
**                     for each metered ramp.
**
** GLOBALS ACCESSED
**          N_INPUTS, N_PARAMETERS, N_P_COLS, N_RULES, N_CLASSES
**
** RESTRICTIONS
**          none
**
** BUGS
**          none
**
** FUTURE DIRECTIONS
**          flc expects pointers for all arguments. FORTRAN always
**          passes by reference, so should not be a problem.  FORTRAN
**          and C index multi-dim arrays differently.  C varies
**          the last dimension most quickly (row major), while FORTRAN
**          varies the first dimension most quickly (column major).
**          Be careful that inputs are specified correctly.
**
** REVISION
**          $Revision: 2.0$
**          Modifications from previous version:
**          1) Calls reader.c directly rather than passing parameters
**     from getmr.c.  Reader is an external file from flc.c
**     2) HW is now an array of headways for each ramp in
**     secs rather than a single value.  It is now returned as
**     one of the arguments of flc
**     3) Inputs now has 2 dimensions instead of 1.
**     The first dimension indicates which ramp, and
**     The second dimension indicates which variable.
**     4) The arguments to flc are different now.
**     The first argument is the pointer to the number
**     of ramps.  The second argument is a pointer to
**     an array of metering rates.  The last argument
**     is the pointer to inputs.
**          5) reader, fuzzify, rules, defuzz are now called within
**          a for loop to find metering rate for each ramp
**          6)  There is now a user input file for each ramp:
**          user_ramp1.inp, user_ramp2.inp, etc.
**
**          $Revision:  2.1$
**          1) The names of the files were changed from user_ramp*.inp
**          to ramp*.inp.
**          2) Print input variables to control.txt (lines 463-469)
**
*/
```

```
/*
 * RCS Identifiers:
 *
 *      $Source: taylorc/fuzzy/traffic/newstuff/flc.c $
 *      $Revision: 2.0 $
 *      $Date: 12/14/94 $
 *      $Author: cyndi $
 *      $State: Exp $
 */
void flc(int *N_RAMPS, float *HW, float (*inputs)[N_INPUTS]) {
        char filename[30];                          /* input file name different for each ramp */
        float parameters[N_PARAMETERS][N_P_COLS];    /* fuzzification parameters */
        float weights[N_RULES];                     /* rule weights */
        float ith_input[N_INPUTS];                  /* inputs for ith ramp */
        float finputs[N_INPUTS][N_CLASSES];         /* fuzzified inputs */
        float fmr[N_CLASSES];                       /* accumulated consequent for each fuzzy MR class */
        float MR;                                   /* crisp metering rate in vehicles/20 secs */
        FILE *control;
        int i, j, k;                                /* loop counters */

        /* open output file to print control action for tuning purposes */
        if ((control=fopen("control.txt", "a"))==NULL) {
                printf("Output file control.txt cannot be opened\n");
                exit(1);
        }

        for (i=0; i<(*N_RAMPS); i++) {                  /* do for each metered ramp */
                /* read parameters and weights from file ramp*.inp */
                sprintf(filename, "ramp percentd.inp", i+1);
                reader(filename, parameters, weights);

                /* initialize input variables for ith ramp */
                for (j=0; j<N_INPUTS; j++) {
                        ith_input[j]=inputs[i][j];
                }

                /* print which ramp # to output file */
                fprintf(control, "METERED RAMP # percentd\n", i+1);
                /* print input variables passed from FRESIM */
                fprintf(control, "VO= percent7.4f, OC= percent7.4f,  DO= percent7.4f,  UO= percent7.4f\n",
                        ith_input[0], ith_input[1], ith_input[2], ith_input[3]);
                fprintf(control, "PO= percent7.4f, SP= percent7.4f,  DS= percent7.4f,  SR= percent7.4f\n",
                        ith_input[4], ith_input[5], ith_input[6], ith_input[7]);
                fprintf(control, "QO= percent7.4f, QD= percent7.4f, AQO= percent7.4f, AQD= percent7.4f\n",
                        ith_input[8], ith_input[9], ith_input[10], ith_input[11]);

                /* convert each crisp input into a set of fuzzy variables finput */
                fuzzify(finputs, ith_input, parameters);

                /* evaluates rule base and accumulates rule outcomes in fmr */
                rules(control, fmr, finputs, weights);

                /* defuzzifies set of fuzzy MR into a crisp MR */
                MR=defuzz(fmr, parameters);    /* vehicles/20 secs */
                HW[i]=20.0/MR;                          /* return headway in seconds */

                /* print control action to a file for tuning purposes */
                fprintf(control, "MR= percent7.4f            HW= percent7.4f\n\n", MR, HW[i]);
        }                               /* end of for loop, each ramp */
        fprintf(control, "\n");
        fclose(control);        /* close output file control.txt */
} /* end of flc */
```