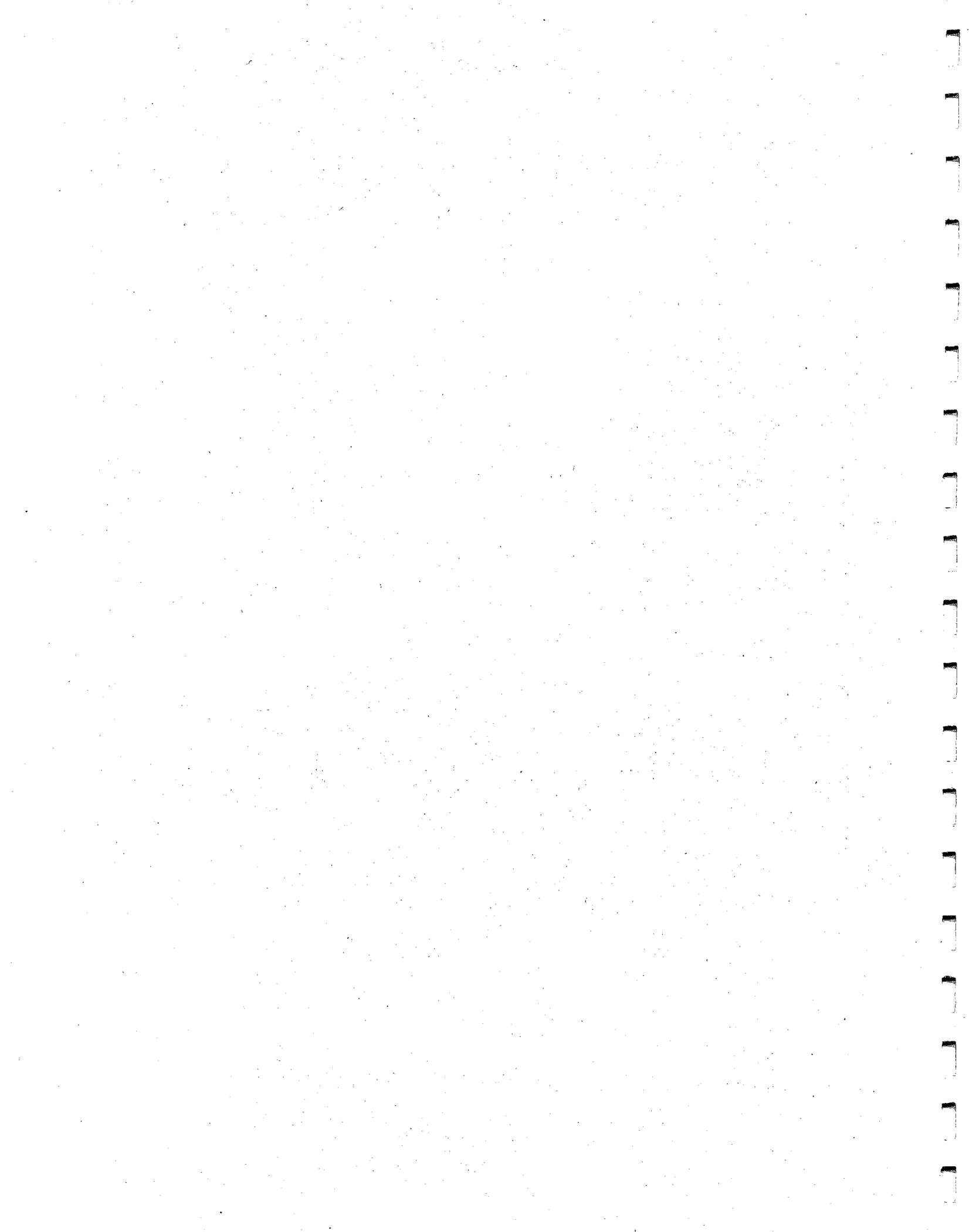


TECHNICAL REPORT STANDARD TITLE PAGE

1. REPORT NO. WA-RD 365.1	2. GOVERNMENT ACCESSION NO.	3. RECIPIENT'S CATALOG NO.	
4. TITLE AND SUBTITLE FREEWAY TRAFFIC DATA PREDICTION USING ARTIFICIAL NEURAL NETWORKS AND DEVELOPMENT OF A FUZZY LOGIC RAMP METERING ALGORITHM		5. REPORT DATE April 1995	6. PERFORMING ORGANIZATION CODE
		8. PERFORMING ORGANIZATION REPORT NO.	
7. AUTHOR(S) Deirdre R. Meldrum Cynthia E. Taylor		10. WORK UNIT NO.	
9. PERFORMING ORGANIZATION NAME AND ADDRESS Washington State Transportation Center (TRAC) University of Washington, JD-10 University District Building; 1107 NE 45th Street, Suite 535 Seattle, Washington 98105-4631		11. CONTRACT OR GRANT NO. Agreement T9903, Task 5	
		13. TYPE OF REPORT AND PERIOD COVERED Final Technical Report	
12. SPONSORING AGENCY NAME AND ADDRESS Washington State Department of Transportation Transportation Building, MS 7370 Olympia, Washington 98504-7370		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 <p style="text-align: justify;"> This research project develops a fuzzy logic ramp metering algorithm utilizing artificial neural network (ANN) traffic data predictors. Considering the highly beneficial effects of ramp metering, such as reduced travel times and lower accident rates, optimizing metering rates is of great importance. The research objective is to overcome limitations of the current Seattle ramp metering algorithm, which reacts to existing bottlenecks rather than preventing them. An algorithm with predictive capabilities can help prevent or delay bottleneck formation. Hence, an accurate 1-minute ANN prediction provides a powerful asset to the ramp metering algorithm. The research project divides into two stages: the ANN traffic data predictor and the fuzzy logic ramp metering algorithm. This research focuses primarily on the ANN traffic data predictors, but also lays the groundwork for the fuzzy logic ramp metering concepts and algorithm. </p> <p style="text-align: justify;"> The ANN predicts 1 minute in advance significantly better than previous techniques in the Seattle area, as well as demonstrates robustness to faulty loop detector data. A multi-layer perceptron type of ANN predicts congested mainline volume and occupancy for a station when given past values of volume and occupancy for that particular station and the adjacent upstream station. This data prediction provides an input to the fuzzy logic ramp metering algorithm. The ramp metering rate is then based on both current and predicted traffic flow. By considering the freeway as a control system instead of one section at a time, the new algorithm should avoid an oscillatory ramp metering rate, and achieve equilibrium more quickly and smoothly. </p>			
17. KEY WORDS artificial neural networks (ANN), fuzzy logic control (FLC), freeway traffic prediction, ramp metering		18. DISTRIBUTION STATEMENT No restrictions. This document is available to the public through the National Technical Information Service, Springfield, VA 22616	
19. SECURITY CLASSIF. (of this report) <p style="text-align: center;">None</p>	20. SECURITY CLASSIF. (of this page) <p style="text-align: center;">None</p>	21. NO. OF PAGES <p style="text-align: center;">111</p>	22. PRICE



Final Technical Report
Research Project T9903, Task 5
Neural Network Control Technology

**FREEWAY TRAFFIC DATA PREDICTION USING
ARTIFICIAL NEURAL NETWORKS AND
DEVELOPMENT OF A FUZZY LOGIC RAMP
METERING ALGORITHM**

by

Deirdre R. Meldrum
Assistant Professor
University of Washington

Cynthia E. Taylor
Research Engineer
University of Washington

Washington State Transportation Center (TRAC)
University of Washington, JD-10
University District Building
1107 NE 45th Street, Suite 535
Seattle, Washington 98105-4631

Washington State Department of Transportation
Technical Monitor
Larry Senn
Advanced Technology Engineer

Prepared for

Washington State Transportation Commission
Department of Transportation
and in cooperation with
U.S. Department of Transportation
Federal Highway Administration

April 1995

DISCLAIMER

The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the official views or policies of the Washington State Transportation Commission, Department of Transportation, or the Federal Highway Administration. This report does not constitute a standard, specification, or regulation.

TABLE OF CONTENTS

<u>Section</u>	<u>Page</u>
EXECUTIVE SUMMARY	vii
Artificial Neural Network	vii
Fuzzy Logic Ramp Metering Algorithm.....	viii
INTRODUCTION	1
Problem Description	1
Research Approach	2
Report Organization	3
ARTIFICIAL NEURAL NETWORK (ANN)	4
Previous Traffic Prediction Techniques.....	4
Time Series Model	4
Use of Artificial Networks	8
Artificial Neural Networks.....	8
Concepts	9
Capabilities.....	12
Development of the ANN Predictors	12
ANN Prediction Results.....	21
Twenty-Second Prediction Results	21
One-Minute Prediction Results	22
Long-Term Prediction Results	31
ANN Implementation Considerations.....	32
Extension to Seattle I-5	33
Role in Ramp Metering Algorithm	33
ANN Conclusions	34
FUZZY LOGIC RAMP METERING ALGORITHM	36
Seattle's Current Ramp Metering Algorithm	36
Operation of the Current Seattle Algorithm.....	36
Suitability of Fuzzy Logic to the Current Algorithm.....	37
Fuzzy Logic.....	39
Fuzzification.....	39
Rules.....	40
Implication	43
Defuzzification	45
Description of the Fuzzy Logic Ramp Metering Algorithm.....	45
Testing the Ramp Metering Algorithm	54
Freeway Model Testing	55
On-Line Testing Procedure	62
Fuzzy Ramp Metering Algorithm Conclusions	64
ACKNOWLEDGMENTS	67
REFERENCES	69

TABLE OF CONTENTS (Continued)

<u>Section</u>	<u>Page</u>
APPENDIX A: NEURAL NETWORK PREDICTOR CODE	A-1
APPENDIX B: ANN LONG TERM PREDICTIONS	B-1
APPENDIX C: FUZZY LOGIC CONTROLLER CODE	C-1

LIST OF FIGURES

1.	Time Series Model Forecasts and Actual Volume.....	5*
2.	Time Series Model Forecasts and Actual Occupancy	6*
3.	Kalman Filter Actual and Predicted Data	7*
4.	ANN Predicted and Actual Flow	7*
5.	Perceptron	10*
6.	Multi-layer Perceptron	10*
7.	Research Site for 20-second Predictions.....	14*
8.	Research Sites 1 and 2	14*
9.	MLP Architecture	16*
10.	Example of Data Scaling.....	17*
11.	Training Results for 20-second Prediction.....	23*
12.	Testing Results for 20-second Prediction	24*
13.	Site 1 Results with Training Day 8/27	26*
14.	Site 1 Results with Training on 8/27 and Testing on 8/28	27*
15.	Site 2 Results with Training Day 8/28	28*
16.	Site 2 Results with Training 8/28 and Testing on 6/1	29*
17.	Site 2 Results with Training on 8/28 and Testing on 8/27	30*
18.	Storage Rate Membership Classes	41*
19.	Correlation-Minimum and Correlation-Product Implication	44*
20.	Centroid Defuzzification	44*
21.	Location of Algorithm Variables	47*
22.	FRESIM Diagram of Study Site	60*

LIST OF TABLES

1.	One-Minute Prediction Results	25
2.	Variable Descriptions for FLC Example	42
3.	Description of Algorithm Variables.....	48*
4.	Fuzzy Classes	48*
5.	Parameter Input Card	50*
6.	Rule Base for Fuzzy Ramp Metering Algorithm	52*

* The figures and tables appear directly following the page numbers cited.

EXECUTIVE SUMMARY

This report documents the development of freeway traffic data prediction using artificial neural networks (ANNs) and the development of a predictive fuzzy logic ramp metering algorithm. The neural networks predicted volume and occupancy significantly better than previous techniques used in the Seattle area. Test results on historical data from the I-5 freeway in Seattle, Washington, demonstrated that a neural network can accurately predict volume and occupancy 1 minute in advance, as well as fill in the gaps for missing data with an appropriate prediction. The volume and occupancy predictions will be used as inputs to a fuzzy logic ramp metering algorithm currently being tested. A 1-minute data prediction will be a useful input to a ramp metering algorithm because this insight can help delay or prevent bottleneck formation. Because the ramp metering rates will be updated every 20 seconds, 1-minute data prediction will provide valuable information to determine the next few metering rates.

ARTIFICIAL NEURAL NETWORK

A multi-layer perceptron type of ANN was trained using back propagation to minimize the mean squared error of the prediction. The inputs to the ANN included the previous ten values of 1-minute volume and occupancy from the predicted station and the adjacent upstream station. Although the ANNs were trained and tested on historical data, they can be implemented for real-time prediction because the inputs are past values of volume and occupancy. For on-line implementation, a neural network will need to be trained from data for each prediction site. The training algorithm and neural network architecture should remain similar for different sites. The ANN parameters should remain similar as long as the data characteristics are similar to the research sites. If not, the network code has been written to allow easy modification of the ANN parameters. For on-line implementation, the neural network should be trained on-line to allow for

seasonal variations. A flag should constantly monitor the accuracy of the prediction to indicate whether retraining is necessary.

Predictions over 1 minute were unreliable. Because of the somewhat chaotic nature of freeway traffic data, longer term prediction with ANNs is a much more difficult problem. Given that a vehicle may travel over 5 miles in a 5-minute forecasted period, the random inputs over that time and distance make longer term prediction challenging.

FUZZY LOGIC RAMP METERING ALGORITHM

The 1-minute ANN prediction will be one of the inputs to a fuzzy logic ramp metering algorithm. The fuzzy logic ramp metering algorithm will determine the metering rates on the basis of both predicted and actual data. This research laid the groundwork for the fuzzy logic ramp metering concepts and algorithm. Fuzzy logic control is well-suited to the ramp metering application for several reasons. It requires a mathematical model of the system, and it can utilize imprecise or incomplete information. These traits are important, given that the freeway is difficult to accurately model and that loop detector data are susceptible to error. The fuzzy logic rules incorporate human expertise, considering all factors simultaneously rather than making a series of adjustments.

The fuzzy logic ramp metering algorithm was designed for flexibility and robustness. For easy algorithm modification and code simplicity, adjustable parameters define the memberships classes. A weight for each rule allows that rule to be emphasized or eliminated for tuning purposes. The fuzzy logic algorithm was also designed to overcome the disadvantages of Seattle's current ramp metering algorithm. The parallel rules of the controller promote robustness to faulty loop detector data. Fuzzy logic control can provide smooth transitions rather than threshold activations, as well as prevent queue formation through the use of qualitative queue inputs. Rules based on the premise of low downstream speed and high downstream occupancy provide a better indicator of bottlenecks than downstream storage rate.

Fuzzy logic algorithm testing and tuning is recommended for future research, first with a simulation model. FRESIM was found to be the most appropriate freeway simulation model for testing the new ramp metering algorithm. Although modifying the FRESIM source code to incorporate the fuzzy logic ramp metering algorithm could take a few months by an experienced programmer, it would be worth the effort. Testing on-line would be complicated by the fact that traffic demand and weather characteristics vary from day to day. Because simulation testing does not have this non-uniformity, it is easier to compare and tune algorithms in simulation testing than in on-line testing.

Because no model can perfectly replicate actual freeway behavior, the algorithm will need further tuning and testing on-line. On-line testing is recommended in four steps: predictor testing, metering rates generated but not used, metering rates sent to a simulation field rack, and then metering rates sent to actual ramps. Although on-line testing is time consuming, it will be necessary to maximize and verify efficiency. Algorithm testing, first through simulation and then on-line, is being continued by the authors through a TransNow 1994-1995 grant. Overall, the fuzzy logic ramp metering algorithm utilizing an ANN traffic data predictor appears quite promising.

INTRODUCTION

As anyone who drives is well aware, traffic congestion is a growing concern across the nation. Because of geographical limitations, simply building more lanes can no longer solve freeway congestion. Today's practical approach to improving traffic flow now emphasizes maximizing freeway efficiency by methods such as public transit, high occupancy vehicles, variable direction lanes, and ramp metering. Improving freeway efficiency motivated this research, which strove to develop a fuzzy logic ramp metering algorithm utilizing artificial neural network (ANN) traffic data predictors.

PROBLEM DESCRIPTION

In 20 metropolitan areas across the United States, studies have shown that ramp metering has dramatically improved travel times, decreased accident rates, and decreased fuel consumption (Robinson and Doctor, 1989). A six year study, during which ramp metering was implemented in Seattle, Washington, indicated that the travel time for a specific 11.1 km section of Interstate 5 decreased from 22 to 11.5 minutes, and the accident rate decreased by 39 percent. During this study, the mainline freeway volumes increased by 86 percent northbound and by 62 percent southbound (Henry and Mehyar, 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 produce significant returns.

Although the ramp metering algorithm currently used in Seattle is one of the most sophisticated in the country, it has limitations (Jacobson, Henry, Mehyar, 1988). The existing ramp metering algorithm has a time lag between problem detection and corrective action. For instance, a reaction to existing congestion may result in overly

restrictive metering rates. Excessive queue build-up may consequently activate the queue override, which increases the metering rate to keep cars from backing up into the arterials. The resulting increase in freeway congestion may then cause the cycle to repeat. Once the freeway starts oscillating between restrictive and high metering rates, it may have trouble escaping this cycle until the congestion dissipates. The algorithm also depends strongly on loop detector data. Induction loops, located under the freeway pavement about every 0.8 km, sample freeway data, which a central computer receives every 20 seconds. Loop detector data are not always reliable because of noisy signals, transmission problems, construction work, and mechanical failure.

RESEARCH APPROACH

The purpose of this project was to develop a predictive ramp metering algorithm to overcome the limitations of the existing ramp metering algorithm. An artificial neural network (ANN) was created to predict freeway volume and occupancy during heavily congested flow. This data prediction will provide an input to a fuzzy logic ramp metering algorithm that was also developed. The result may be a ramp metering rate that is based on both current and predicted traffic flow. Ideally, the new algorithm may help prevent bottlenecks rather than simply react to them. By considering the freeway as a control system instead of one section at a time, the fuzzy logic algorithm should avoid an oscillatory ramp metering rate and should achieve equilibrium more quickly and smoothly.

There were two stages to this research project: the development of a neural network traffic data predictor and the development of a fuzzy logic ramp metering algorithm. The project focused primarily on the ANN traffic data predictors, but it also laid the groundwork for the fuzzy logic ramp metering concepts and algorithm. Because fuzzy logic algorithm testing and fine-tuning requires a calibrated freeway model capable

of closed-loop control, which is unavailable at this time, fuzzy logic algorithm testing will require further research.

REPORT ORGANIZATION

Following the introduction, the section on ANN research includes previous freeway traffic prediction techniques, the prediction method, short- and long-term prediction results, and implementation considerations. In the section about fuzzy logic, background material on ramp metering algorithms and fuzzy logic are presented. The report presents a description of the algorithm and algorithm testing options.

ARTIFICIAL NEURAL NETWORK (ANN)

PREVIOUS TRAFFIC PREDICTION TECHNIQUES

Time Series Models

Two previous approaches for predicting traffic data in Seattle on I-5 were based on time series models (Nihan and Zhu, 1992) and Kalman filtering (Dailey, 1993). Both of these approaches forecast volume and occupancy 1 minute in advance. Volume is defined as the aggregate number of cars during a sampling interval, and occupancy is defined as the average percentage of time that a vehicle occupies the mainline during a sampling interval. Models were developed on the basis of historical loop detector data.

Of the five time series models Nihan and Zhu explored, the best two were the adaptive prediction system model and the double exponential smoothing model. The adaptive prediction model forecast is a linear weighted sum of past data values. The double exponential smoothing model forecast follows a linear trend. For these two methods, volume results are shown in Figure 1 and occupancy results are shown in Figure 2.

Dailey's system model, used by the Kalman filter predictor, was based on historical data dynamics and statistical process information. Dailey constructed the system model on the basis of correlation between state variables, such as volume and occupancy, data from adjoining lanes, and time series loop detector data. The Kalman filter prediction is useful for detecting anomalies in the data, such as an incident (defined as nonrecurrent congestion caused by blockage of one lane or more). If the prediction varies significantly from the actual data, an anomaly may have occurred. These results are shown in Figure 3.

These two approaches are traditional in the sense that they rely on human knowledge of the system and stochastic processes. They are also traditional in that they

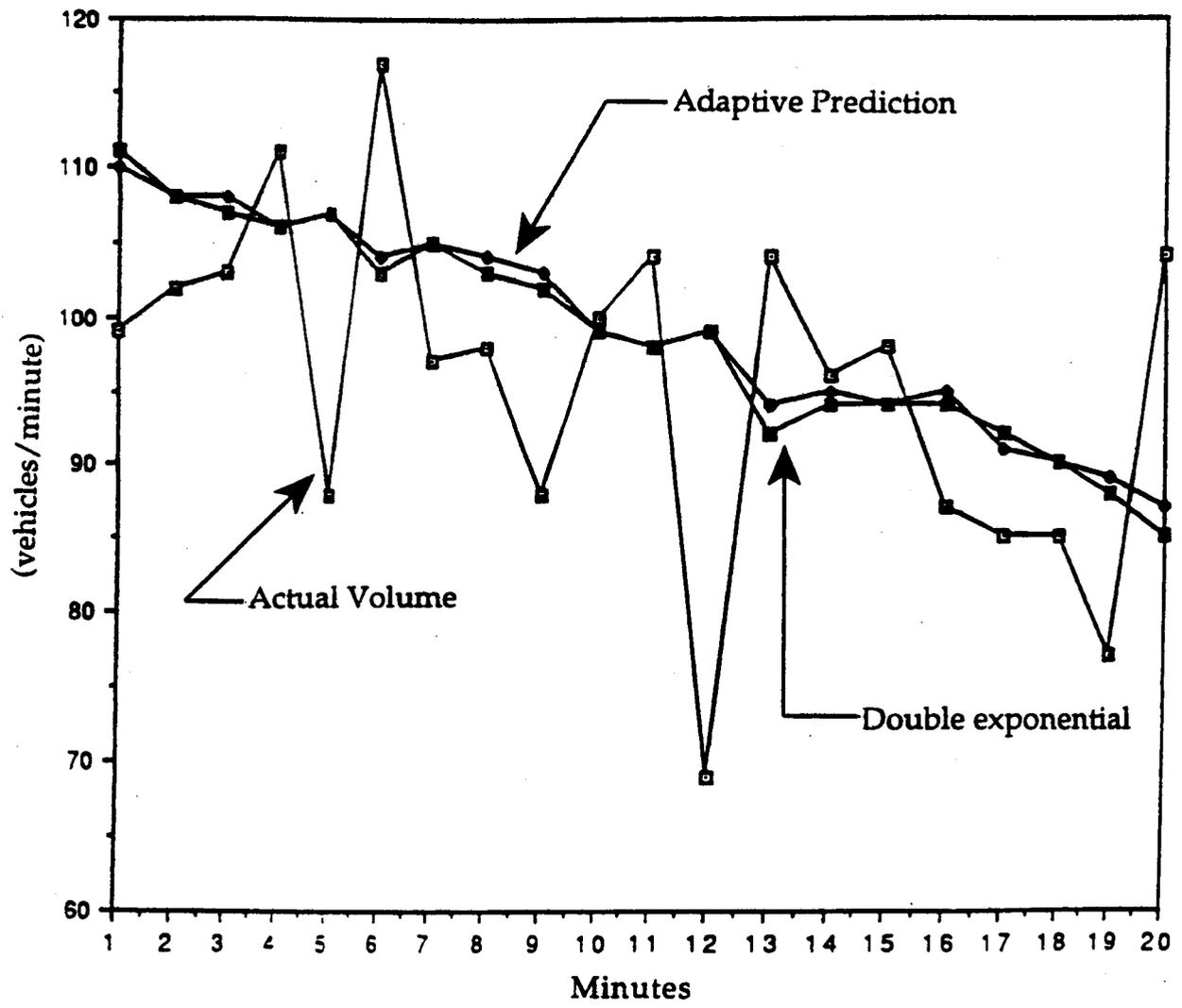


Figure 1. Time Series Model Forecasts and Actual Volume

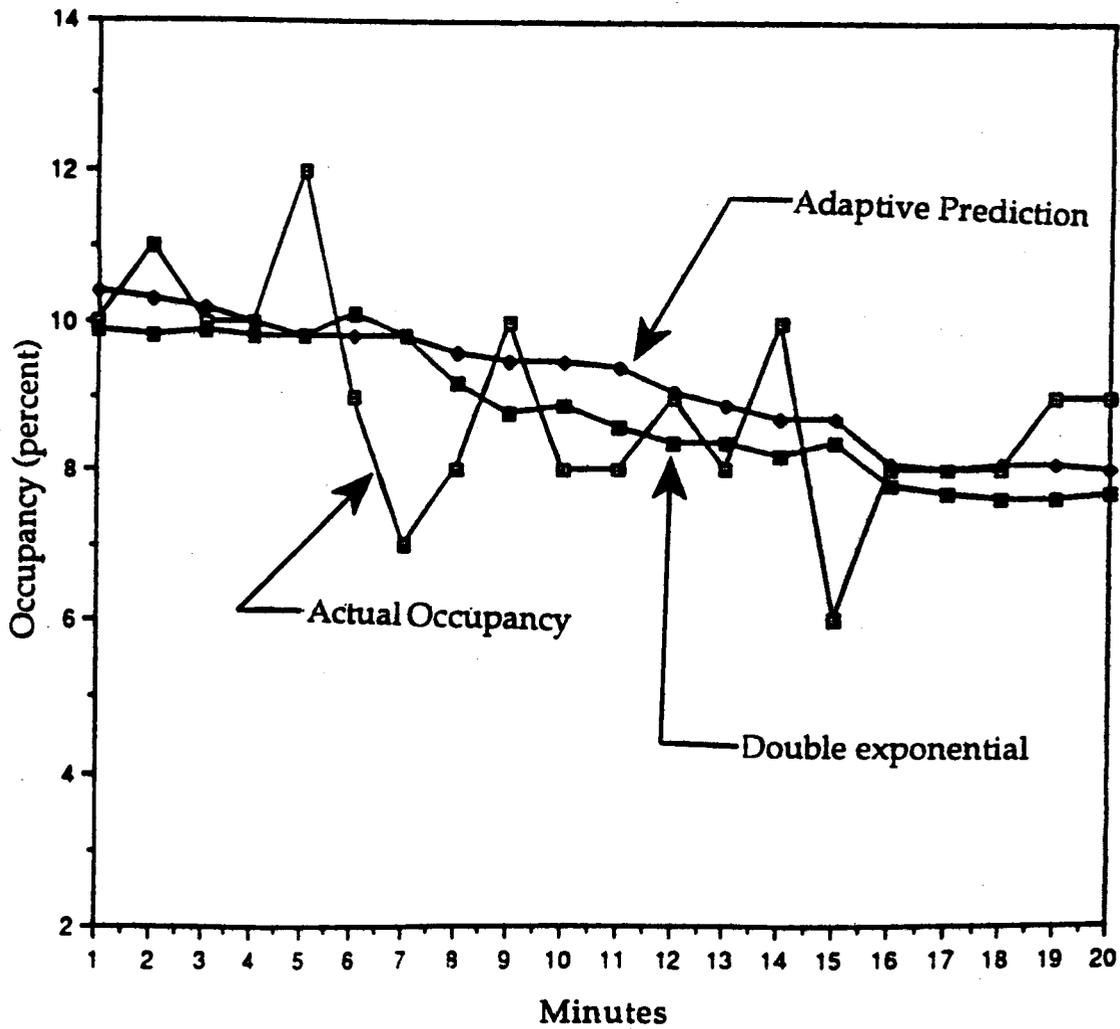


Figure 2. Time Series Model Forecasts and Actual Occupancy

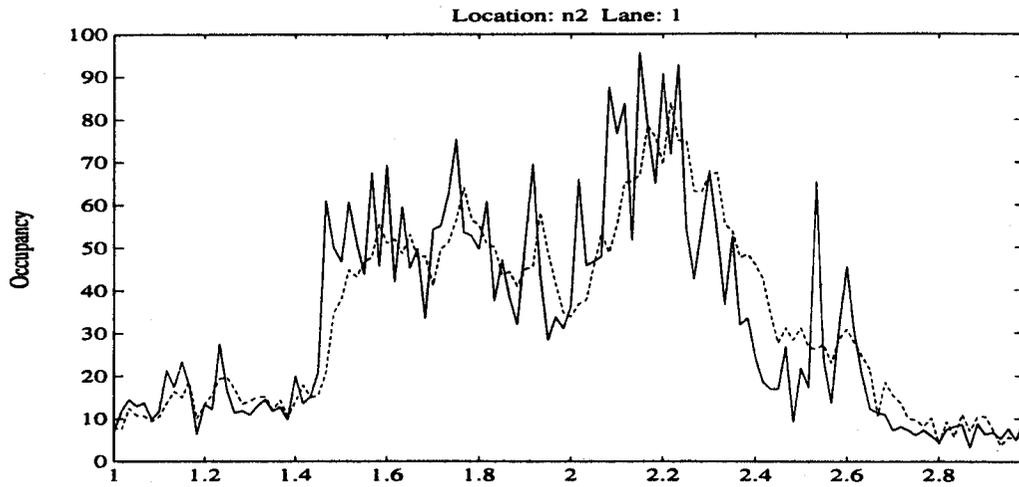
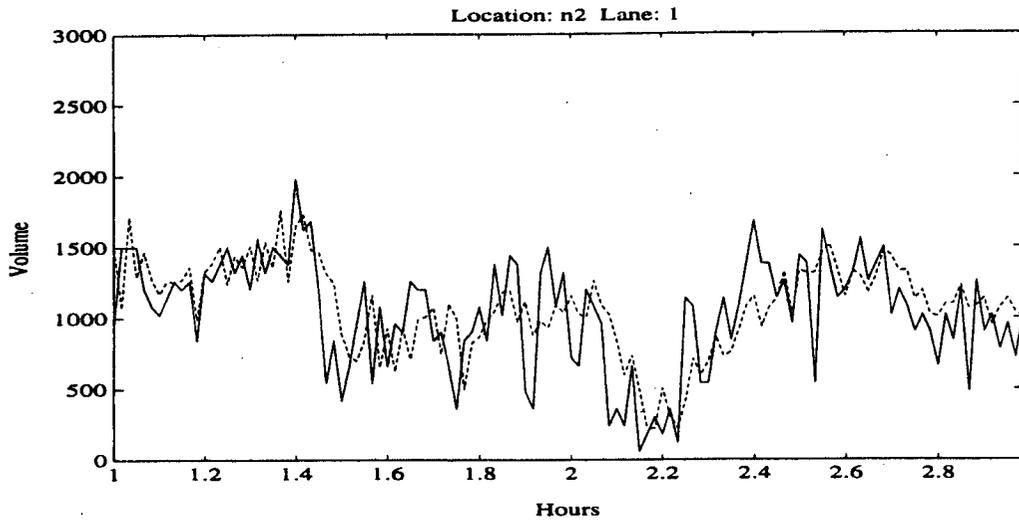


Figure 3. Kalman Filter Actual (-) and Predicted (-) Volume and Occupancy

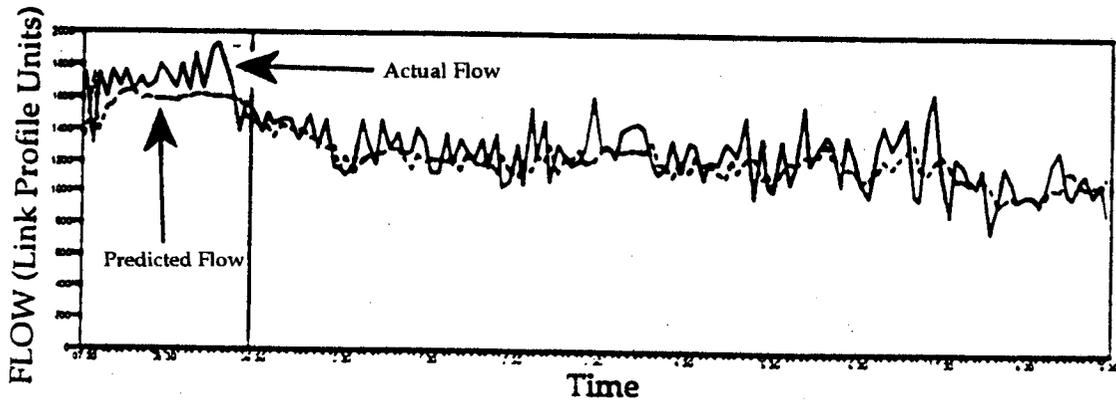


Figure 4. ANN Predicted (dotted line) and Actual (solid line) Flow

are restricted to linear system models. Since traffic flow is highly dynamic, complex, and nonlinear, one cannot expect linear, model-dependent prediction techniques to yield accurate or reliable results. Difficulty in obtaining an accurate freeway system model has suggested an alternative data prediction technique: artificial neural networks.

Use of Artificial Neural Networks

Researchers have had some success in using ANNs to predict data 5 minutes in advance for the city of Leicester in England (Clark, Dougherty, and Kirby, 1993). The ANN is trained on 5-minute historical data collected from the SCOOT traffic control system (Bretherton, 1989) between 7:30 a.m. and 7:45 p.m. The flow is measured in Link Profile Units, each vehicle having a value of about 18. The ANN is trained on approximately 500 examples. The back propagation algorithm discussed later minimizes the mean squared error of the data prediction. Figure 4 shows the predicted and actual flow versus time for a particular link.

These researchers have also had some success predicting 5 minutes in advance for the A2 motorway in the Netherlands (Dougherty and Kirby, 1993). For this problem, the 240 network inputs include upstream speed, volume, and occupancy. The data are preprocessed using a moving average from the previous 5 minutes.

ARTIFICIAL NEURAL NETWORKS

Interest in artificial neural networks (ANNs) has increased over the years as they have proved adept at solving a variety of problems. In the transportation area, ANNs have been used to detect freeway incidents (Chang and Huarng, 1993; Wiederholt, Okunieff, and Wang, 1993), classify dynamic traffic patterns (Hua and Faghri, 1993; Mead, Fisher, Jones, Bisset, and Lee, 1994), obtain macroscopic models of freeway traffic (Zhang and Ritchie, 1993), and estimate multi-period travel times in transportation networks (Wei and Schonfeld, 1993). This section introduces basic neural network

concepts and explains why neural networks are suitable for the traffic data prediction problem.

Concepts

An artificial neural network, composed of connected artificial neurons, abstractly emulates the behavior of a biological nervous system. Information is stored in the strength of the artificial neuron connections, called weights. The ANN process of adjusting the weights is called learning. ANN learning can be classified as supervised or unsupervised. Supervised ANNs are given input/output data to train the network. In contrast, unsupervised ANNs are only given input data and must cluster similar data together during training. Once trained, the ANN can provide an appropriate output for a given input, if the input is within the training domain. When data are outside the training domain, the problem is one of prediction. If the underlying dynamics within the training domain and outside the training domain remain reasonably similar, then the ANN should perform well. ANN prediction on data pairs outside the training set is called *generalization*.

The perceptron (Figure 5), also called a neuron, is the basic building block for a type of ANN called a multi-layer perceptron (MLP). Cascaded perceptron layers form an MLP (Figure 6), which can learn complex functions. Each neuron in the MLP sums a weighted input vector according to

$$y_j = \sum_{i=0}^N w_{ji} x_i \quad \text{Eq. 1}$$

where the weight, w_{ji} , is the gain from the i th neuron in a layer to the j th neuron in the next layer. The number of neuron inputs is N , and x_i is the i th input to the neuron. A function $f(y)$ of this sum, called the activation function, is the perceptron output, u . A sigmoidal function

$$u=f(y) = \frac{1}{1+e^{-y}} \quad \text{Eq. 2}$$

is the most commonly used activation function in MLPs.

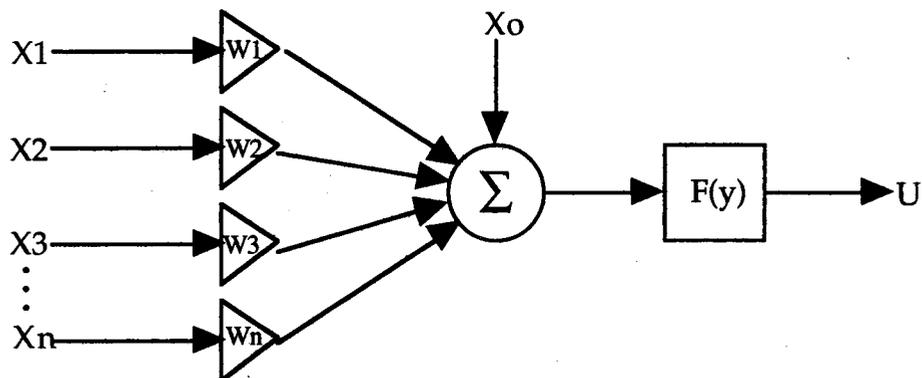


Figure 5. Perceptron

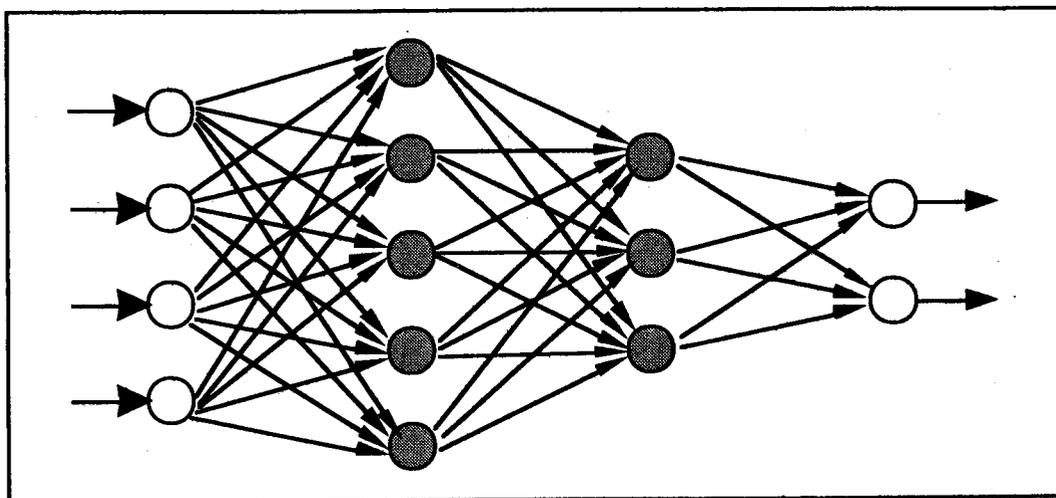


Figure 6. Multi-layer Perceptron

During the learning process, adjustment of the weights, or gains of neuron inputs, minimizes the error between the neural network's actual output and the desired output. A commonly used learning algorithm that minimizes the output error is back propagation. Back propagation minimizes the 2-norm of the output error (mean squared error), represented below by the cost function J . The desired output is t and the actual output is u . The ANN has M outputs total.

$$J = \left((t_1 - u_1)^2 + (t_2 - u_2)^2 + \dots + (t_M - u_M)^2 \right)^{1/2} \quad \text{Eq. 3}$$

The 2-norm is the most common cost function to minimize, but other cost functions are possible and sometimes highly desirable. Back propagation adjusts each weight in the ANN after every training example according to

$$\Delta w_{ji} = -\eta \frac{\partial J}{\partial w_{ji}} \quad \text{Eq. 4}$$

where the weight change is proportional to the learning rate, η . The error for the output layer is calculated first and is then propagated backward using the chain rule

$$\Delta w_{ji} = \eta \delta_j \frac{\partial u_j}{\partial y_j} \frac{\partial y_j}{\partial w_{ji}} \quad \text{Eq. 5}$$

The effective error, δ_j , is found by

$$\delta_j = -\frac{\partial J}{\partial u_j} \quad \text{Eq. 6}$$

for the j th neuron. The weight change further reduces to

$$\Delta w_{ji} = \eta \delta_j f'(y_j) u_i \quad \text{Eq. 7}$$

For more information regarding types of ANNs, the reader is referred elsewhere (Hush and Horne, 1993; Kung, 1993).

Capabilities

One reason for the growing interest in ANNs is that they overcome many of the drawbacks associated with traditional problem solving techniques. Because most traditional freeway data prediction techniques rely on the accuracy of the system model or knowledge of the underlying stochastic process (such as correlation between state variables used by Dailey, 1993), the success of the prediction is limited by human knowledge of the system. With ANNs, no knowledge of the system model is necessary. The multi-layer perceptron (MLP) type of ANN requires only a training set of inputs paired with appropriate outputs to learn the function. Thus, the ANN can be treated as a black box independent of the particular geometry of a highway section.

A second advantage is that properly trained ANNs are relatively insensitive to erroneous or missing data; this is a valuable asset in traffic data prediction, as loop detector data are often unreliable. A third advantage to ANNs is that they readily handle nonlinear systems, an important trait for dealing with highly dynamic traffic data. ANNs are also comparatively easy to program.

Although ANNs offer a viable alternative to traditional prediction techniques, disadvantages do exist. The ANN requires a long time to learn the training data and may have trouble generalizing to new data outside the training set, as noted previously. In addition, no standard method exists for finding the optimal architecture, such as choosing the number of hidden neurons, activation function, and cost function for a MLP. In some cases, another disadvantage is finding something to serve as the teacher, defined as the desired output for a given input in the training set.

DEVELOPMENT OF THE ANN PREDICTORS

For the freeway volume and occupancy prediction problem, a MLP trained by a back propagation algorithm was found to be appropriate. The artificial neural network

code was written in C and runs on a Sun Workstation; Appendix A contains a version of the program.

The ANN used historical data to train the ANNs. The 20-second predictors were trained and tested on 20-second data from the 2:30 to 6:00 p.m. weekday period from December 14 and 15, 1993. The 20-second data were from a section of northbound I-5 near the Northgate exit (station 159 in Figure 7). The 1-minute and 2-minute predictors were trained and tested on 1-minute data from the 6:00 to 9:30 a.m. weekday period from June 1 (Monday), 15, 18 and August 27 (Thursday) and 28, 1992. These data were from sites 1 and 2 (Figure 8). Sites 1 and 2 were sections of southbound I-5 near the NE 195th Street and NE 205 Street interchange in Seattle. The 5-minute predictors were trained and tested on 5-minute historical data from the 6:00 to 10:00 a.m. weekday period from August 3 (Monday), 4, 5, 6, 7, 10, 24, 25, 26, 27, 28, and 31, 1992. The 5-minute data prediction was for site 2 (Figure 8). All research sites were chosen for their recurrent congestion, proximity to ramp metering, and data availability.

Volume and occupancy data were collected with loop detectors every 1/60th of a second, sent to a substation, and then sent to a central VAX computer system every 20 seconds. The volume was the aggregate number of cars over all mainline lanes during the past sample. The occupancy was the time percentage in which the mainline was occupied, averaged over the past sampling period and all lanes. Note that the data were not a true sample but an *accumulation* or *average* over the sampling interval. For this reason, Z-transforms, fast Fourier transforms, and other digital techniques did not necessarily apply.

The multi-layer perceptron had one hidden layer with a final configuration of 28 hidden neurons. The MLP trained over the historical weekday periods given above. These training periods turned out to be sufficient for generalization to different days of the week. Inputs to the neural network included the past sampled volumes and

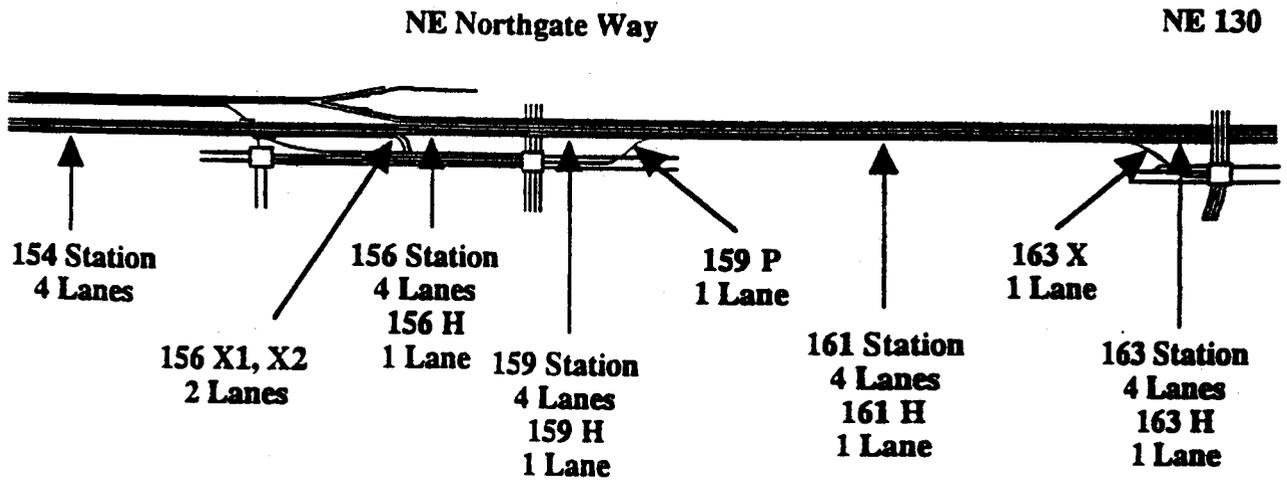


Figure 7. Research Site for 20-second Predictions

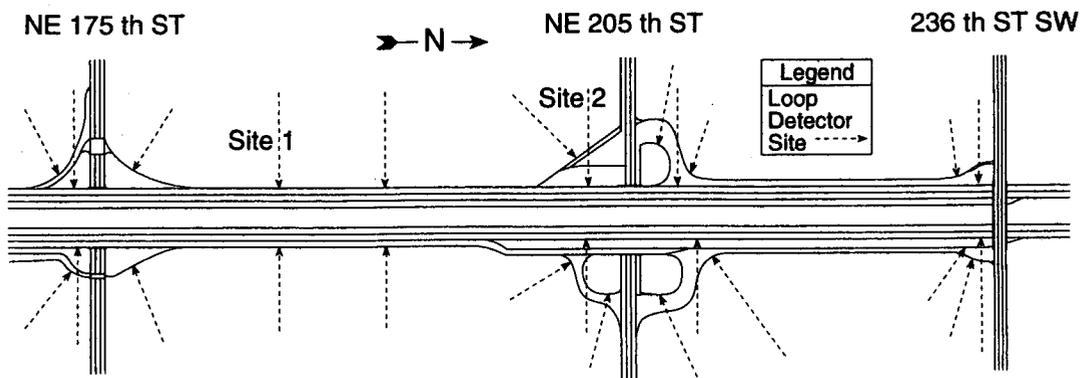


Figure 8. Research Sites 1 and 2

occupancies of the station, D , for which data were to be predicted and for the adjacent upstream station, A . The total number of inputs was given by

$$\# \text{ of Inputs} = \text{LENGTH} * 2 * \text{STATIONS} \quad \text{Eq. 8}$$

where LENGTH was the length of the tapped delay line (how many past samples to use as inputs), the factor of 2 accounted for volume and occupancy, and STATIONS was the number of data stations used as inputs. In Figure 9, LENGTH is 10 and two STATIONS D and A are used as inputs, for a total of 40 inputs. The two outputs from the ANN are volume, VOL , and occupancy, OCC , at the next sample. With this configuration, the ANN predicts the volume and occupancy 1 minute in advance.

Many different architectures were compared by varying the number of hidden neurons, the number of past data values used for inputs, the number of training iterations, and the learning rates. Altering one of these parameters at a time and comparing the new results with the best previous results helped indicate trends. Even so, extensive trial and error were necessary before successful results were obtained. Over 100 variations on architecture and learning technique were compared.

The ANN has a sigmoid activation function $f(y)$ given by

$$f(y) = \frac{1}{1 + e^{-y}} \quad \text{Eq. 9}$$

which required the teacher to be scaled between 0 and 1. Originally, the data were scaled between 0 and 1 by dividing the data by the maximum absolute value of the data set. However, this method turned out to be fallible to irregular or erroneous data. For example, the three sharp drops in raw volume data shown in Figure 10 were probably due to transmission errors, as both the volume and occupancy were exceptionally low for the peak rush hour. If the anomaly had been caused by stalled traffic, the occupancy should have been high while the volume was low. Data scaled by the maximum absolute value scaling method would have been skewed to near 1 because of the irregular low peaks.

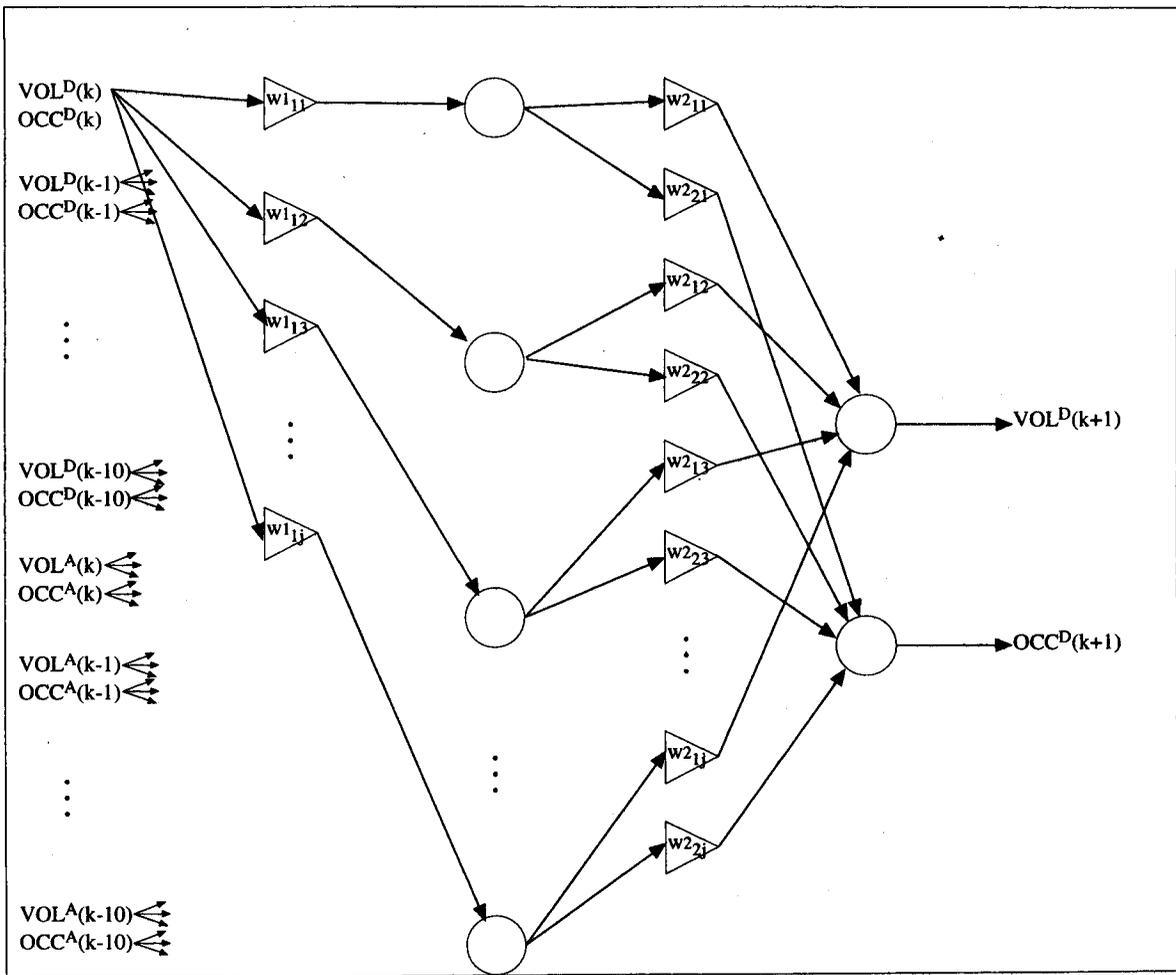


Figure 9. MLP Architecture

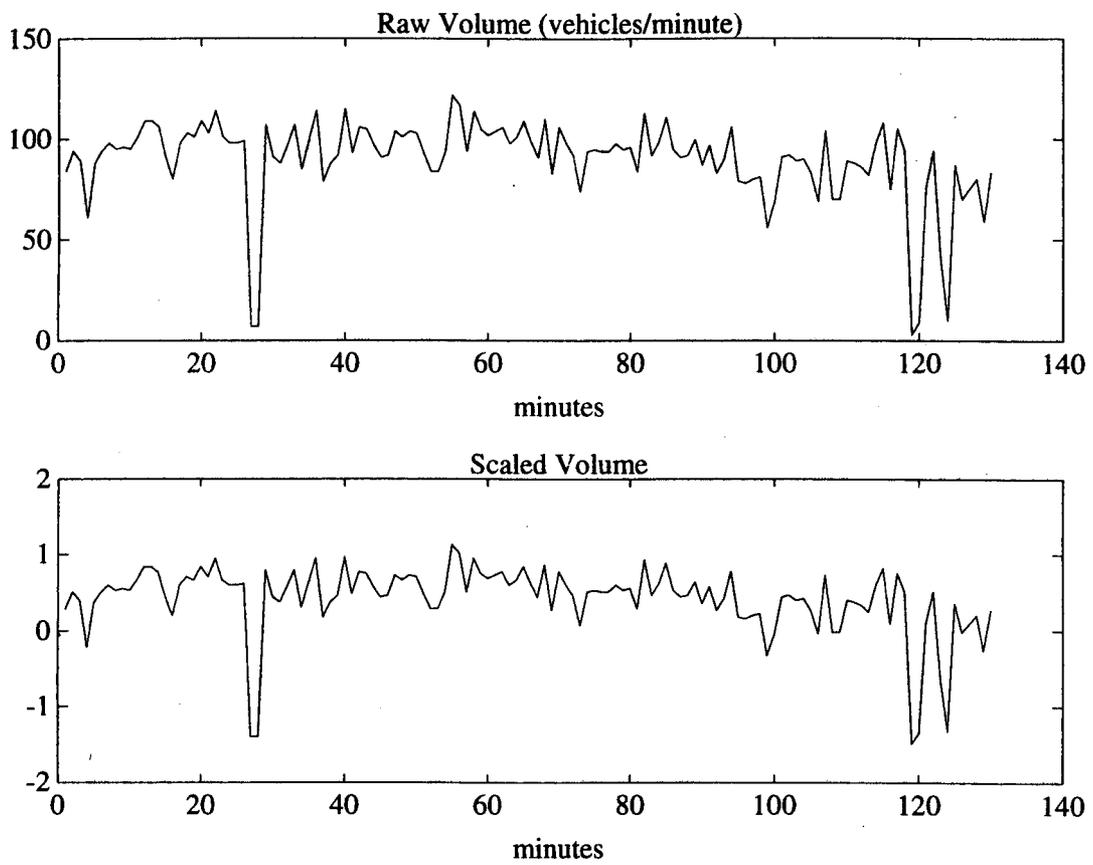


Figure 10. Example of Data Scaling

The MLP does not generalize well when *trained* on data scaled this way, nor does it *test* well on data scaled this way.

Consequently, an improved scaling scheme was devised to better handle irregular and erroneous data. Assuming a Gaussian data distribution, the mean and standard deviation of a typical data set scales 98 percent of a typical data set between 0 and 1 using the following equations:

$$\text{Scaled } VOL = \frac{VOL - \text{mean}(VOL)}{4\text{std}(VOL)} + 0.5 \quad \text{Eq. 10}$$

and

$$\text{Scaled } OCC = \frac{OCC - \text{mean}(OCC)}{4\text{std}(OCC)} + 0.5 \quad \text{Eq. 11}$$

The scaled volume data in Figure 10 displays how this scaling method handled irregular data. If trained on these scaled data, the neural network learned a zero output for those low peaks rather than skewing the entire data set to one extreme. If the desired output was less than 0, the teacher became 0. Likewise, if the desired output exceeded 1, the teacher became 1. Scaled data that fell outside the 0 to 1 range may have represented heavy congestion, an incident, or erroneous data.

The back propagation algorithm described previously minimized the 2-norm of the output error. When learning stalled, the learning rate, η , was decreased to avoid bypassing the local minimum. For example, if the mean squared error (MSE) did not decrease for more than 200 iterations, the learning rate may have been too large. After numerous trials, the following learning rate pattern was found to yield the best results for the 20-second predictors:

$$\eta = 2.0 \text{ for iterations} < 3000$$

$$\eta = 0.9 \text{ for } 3000 \leq \text{iterations} < 5000$$

$$\eta = 0.5 \text{ for } 5000 \leq \text{iterations} < 10,000$$

$$\eta = 0.2 \text{ for iterations} \geq 10,000$$

The 1-minute predictors performed best with the learning rate pattern below:

$\eta=2.0$ for iterations <1000

$\eta=0.9$ for $1000 \leq \text{iterations} < 1400$

$\eta=0.5$ for $1400 \leq \text{iterations} < 1600$

$\eta=0.2$ for iterations ≥ 1600

Appendix B contains details about the learning technique and architecture for the 2-minute and 5-minute predictors.

Training was halted when the testing MSE reached a minimum in order to generalize to new data. For the 20-second data, the best generalization occurred at 15 000 training iterations. About 3000 iterations produced the lowest testing MSE for the 1-minute predictors. Fewer sweeps did not allow the MLP to learn the training set well enough to generalize to the testing set, while more sweeps caused the MLP to memorize the training set and generalize poorly. For the 2-minute predictors, using the past four samples for the input worked best. Using the past 10 samples for the input worked best for the 1-minute predictors.

Surprisingly, a general trend between performance and architecture variation was not evident. For instance, reducing the number of past input samples from ten to nine decreased the prediction performance, so the researchers expected that reducing the number of past input samples to eight would further decrease prediction abilities. However, the architecture with eight past input samples outperformed the architecture with nine past input samples. Similarly, the number of hidden neurons did not correlate generally with prediction performance. Although prediction performance was best with 28 hidden neurons for both the 20-second and 1-minute predictors, the architecture with 30 hidden neurons was better than that with 29 hidden neurons. With obscure relationships between architecture and prediction performance, trial and error was necessary to find the most successful architecture.

A momentum term (an additional weight change in the same direction as the previous weight change) could have been included in the weight change equation to speed learning:

$$\Delta w_{ji}(t+1) = \eta \delta_j f'(y_j) u_i + \alpha \Delta w_{ji}(t) \quad \text{Eq. 12}$$

where α represents momentum. For most ANNs, no momentum is used because it degrades generalization.

Once the ANN had been trained, generalization ability was tested on a new data set. Running the ANN on another congested historical data set from a different day, but at the same section and same time of day, tested the performance of the ANN. Generalization was the most challenging aspect of the ANN traffic predictor. For this reason, the mean squared error of the testing data best indicated the success of the MLP. For the ANN to generalize well, it had to learn the general data trend rather than learn each training point exactly. If the ANN had an excessive number of hidden neurons or was trained over too many iterations, it might have memorized the training set. In this case, the network would have difficulty generalizing to new data. Instead, the ANN needed just enough hidden neurons and training iterations to learn the general data trend. The architecture in Figure 9 with the above learning rates best met this need.

To improve generalization abilities, the ANN was cyclically pruned and retrained, but with limited success. The pruning process eliminated the smallest weights. For two reasons, only weights between the input and hidden layer were pruned. The output was less sensitive to weights between the input and hidden layer than weights between the hidden and output layer. With 40 inputs, at least some of the 1120 weight connections to the hidden layer were likely to be inconsequential. Also, pruning a weight between the input to hidden layer effectively eliminated that input.

After 1000 iterations, the pruning cycle began. Every 100 sweeps, the three smallest weights in the input to hidden layer were eliminated, and then training continued. However, this pruning method produced results that were slightly inferior to

no pruning at all for most predictors. The 1-minute predictor functioned exceptionally well anyway, so pruning was not used. Apparently, the 1-minute predictor was already a minimal network.

ANN PREDICTION RESULTS

Short-term prediction of freeway volume and occupancy (1 minute or less) is useful to a ramp metering algorithm to indicate approaching gaps in the traffic. With this knowledge, the ramp metering rate can increase when low occupancy is anticipated to fit more vehicles into traffic gaps. A longer term prediction (over 1 minute) is useful to indicate general traffic trends. This section reports on 20-second and 1-minute prediction results, and Appendix B contains 2-minute and 5-minute prediction results.

The figures in this section show the actual volume and occupancy (dotted line) and the predicted volume and occupancy (solid line) versus time. The training data sets show how well the neural network learned the input/output examples given to it. The testing data sets show how well the neural network predicted the outputs for inputs outside the training set. The testing set MSE, as discussed previously, best measured the ANN's prediction performance, so the ANN was trained until the testing MSE reached its minimum. The input data were scaled before neural network operations, as described in the previous section. Using the same equation, the output data were then scaled back to the original dimensions after neural network operation, which is how these figures show it.

Twenty-Second Prediction Results

The 20-second predictions performed reasonably well. The 20-second predictors were trained on data from Site 3, located near the northbound Northgate Exit. The architecture from Figure 9 was used to predict 20 seconds in advance using 20-second data for the ANN inputs. The ANN was trained on historical data from a 2:30 to 6:00 p.m. period with about 700 examples in both the training and testing set. The recurrent

congestion at this site and time is one of the worst on the Seattle I-5 freeway. The data reached bottleneck conditions from sample 520 to 550 in the training set and from sample 500 to 650 in the testing set, as evident by the high occupancy. For better graphing resolution, the training data were broken into two graphs (Figure 11), and the testing data were broken into two graphs (Figure 12).

The trade-off between learning the training data and generalizing well to the testing set was prevalent for the 20-second prediction problem. The ANN did not learn the testing set very well, but further training would have increased the testing MSE. The ANN learned and predicted occupancy better than it did volume. Occupancy is a better indicator of congestion than volume, so this prediction is more valuable to the ramp metering algorithm, anyway. The ANN predicted occupancy fairly well for samples 350 to 700, except for the bottleneck conditions between samples 500 and 650. For this period, the ANN produced a maximum occupancy prediction of 1, which was then scaled up to a value of near 30 percent using the scaling equations. The ANN also produced a minimum volume prediction of 0, which was then scaled back to a value of near 20. This ANN could be used to detect bottlenecks by looking for a sustained prediction of maximum occupancy near 30 percent and a minimum volume of near 20. If all data were scaled within the 0 to 1 range, as discussed previously, the sensitivity of the prediction would be reduced to accommodate the extreme points. For this reason, the most appropriate ANN for incident detection (for which the extreme points would be scaled to within the 0 to 1 range) would require a different scaling technique than the ANN most suitable for data prediction.

One-Minute Prediction Results

The ANN architecture in Figure 9 predicted 1 minute in advance using 1-minute data for the inputs. Training and testing took place over a historical data set from the 6:30 to 9:00 a.m. period. Overall, the 1-minute ANN predictors performed extremely well, especially given the chaotic nature of 1-minute traffic data. Given that a car travels

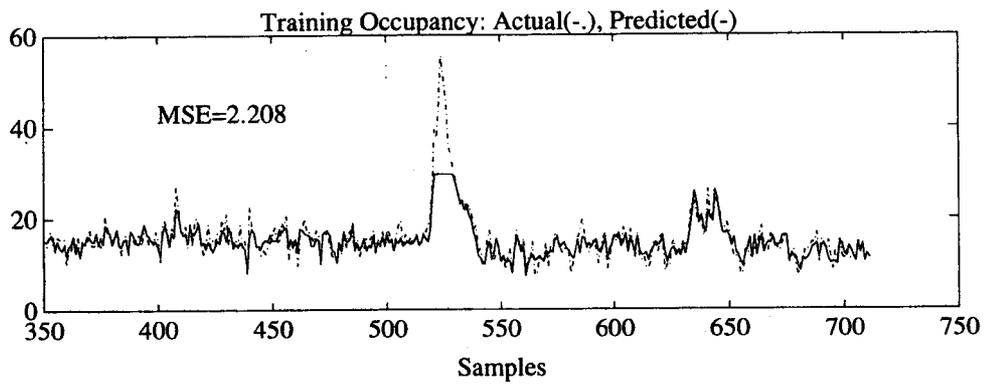
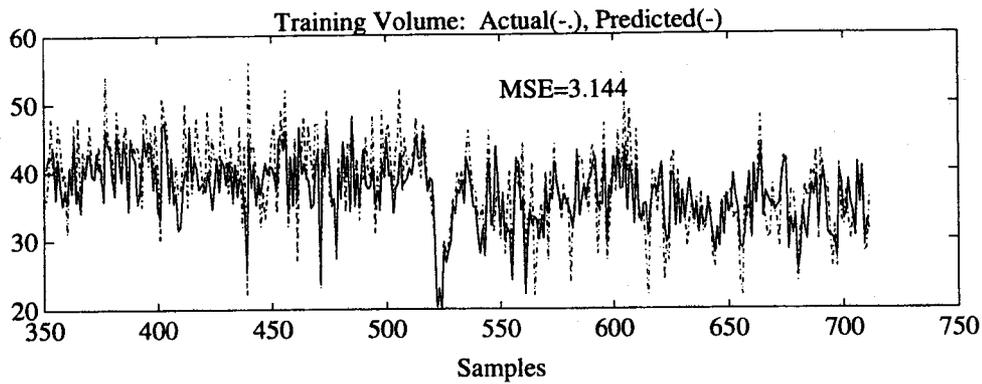
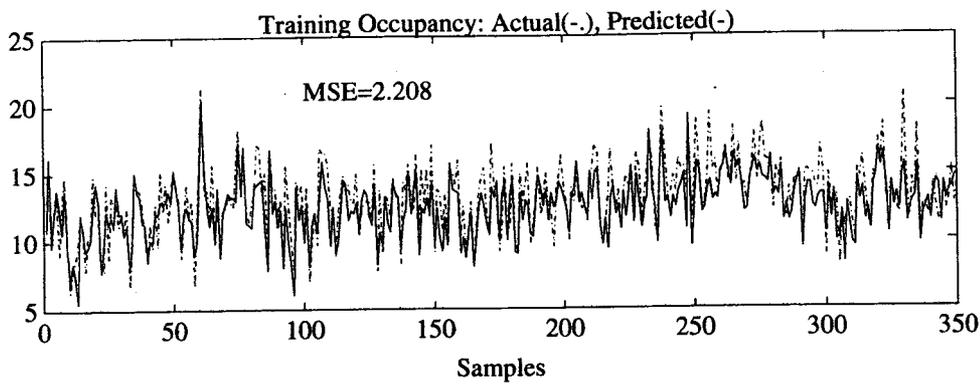
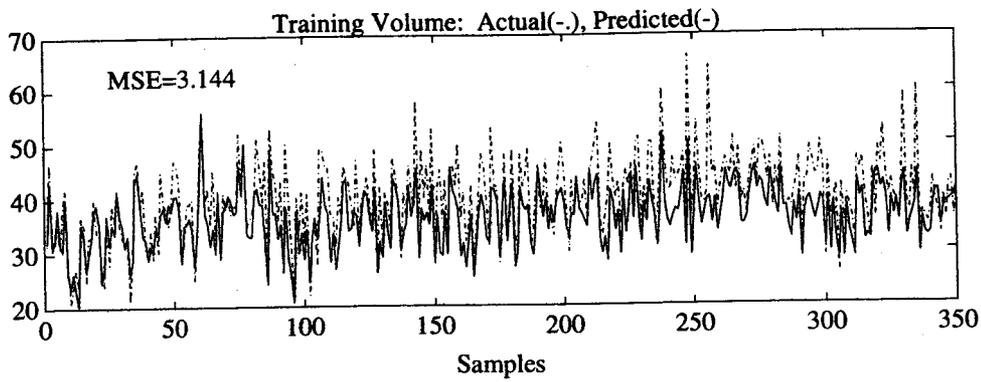


Figure 11. Training Results for 20-second Prediction

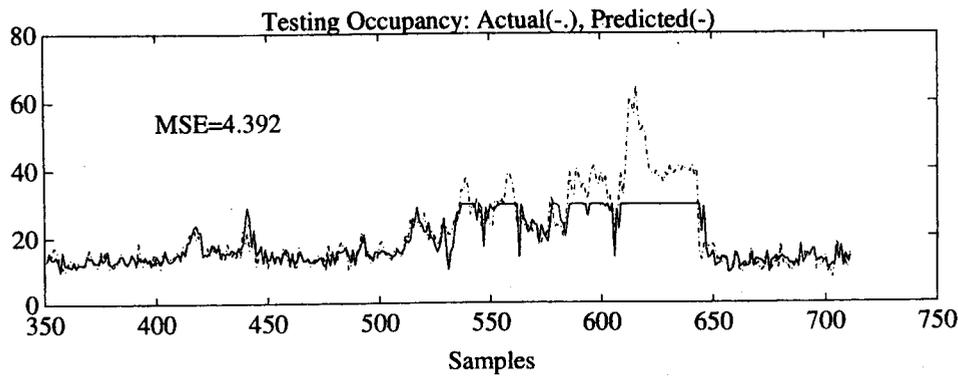
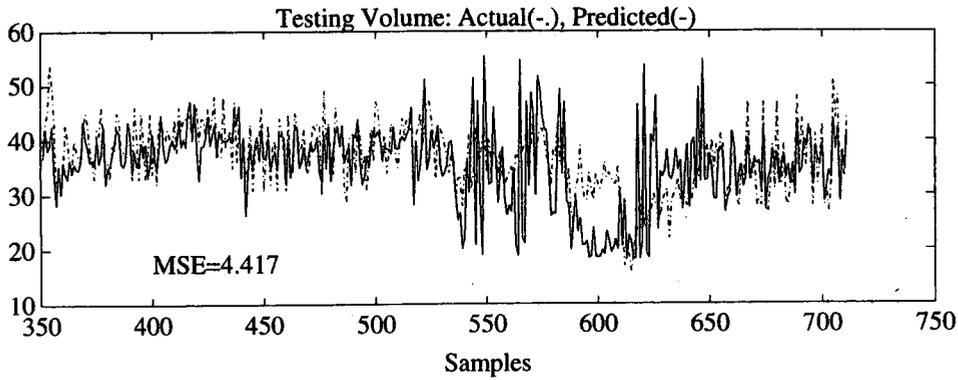
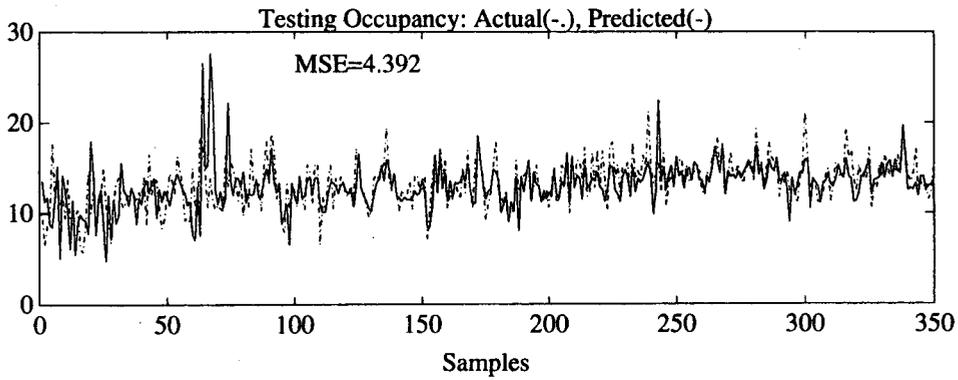
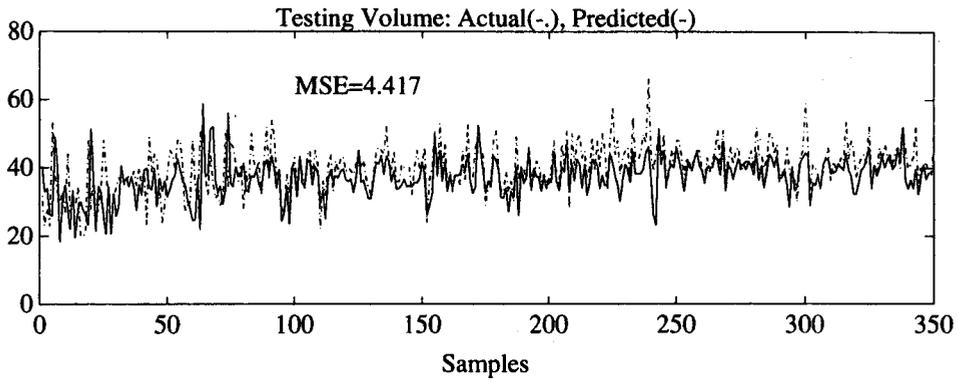


Figure 12. Testing Results for 20-second Prediction

approximately 1.6 km in 1 minute, such an accurate short-term forecast is remarkable. Because the ramp metering rates are updated every 20 seconds, a 1-minute prediction would provide valuable insight in determining appropriate ramp metering rates. Table 1 summarizes representative results from research sites 1 and 2.

The figures show results from one ANN for each of the two sites. Figure 13 shows the training set for the site 1 ANN, and figure 14 shows the prediction results for that ANN. Figure 15 shows the training set for the site 2 ANN, and figures 16 and 17 show the prediction results for that ANN on two different testing sets.

The ANN at site 1, located slightly downstream of an on-ramp, trained on data from Thursday, August 27, 1992. The mean squared error (MSE) for the training set was relatively low because of an absence of irregular valleys (Figure 13).

The site 1 ANN generalizes very well to the data from Friday August 28, 1992, with testing MSEs nearly as low as the training MSEs (Figure 14).

Predicting traffic flow at site 2, located next to an on-ramp, turned out to be more challenging than at site 1. The site 2 data set contained sharper fluctuations. The ANN trained on data from Friday, August 28, 1992 (Figure 15).

The ANN accurately predicted volume and occupancy for the test day, Monday, June 1, 1992 (Figure 16). Although the ANN did not quite reach all the valleys, the performance was still far better than that of traditional traffic prediction techniques.

Table 1. One-Minute Prediction Results

Site	Training Day	Testing Day	Training MSE VOL	Training MSE OCC	Testing MSE VOL	Testing MSE OCC
1	8/27	8/28	0.2204	0.3980	0.3002	0.3341
2	8/28	6/01	0.2097	0.4605	0.3322	0.4014
2	8/28	8/27	0.2097	0.4605	0.5295	0.6873

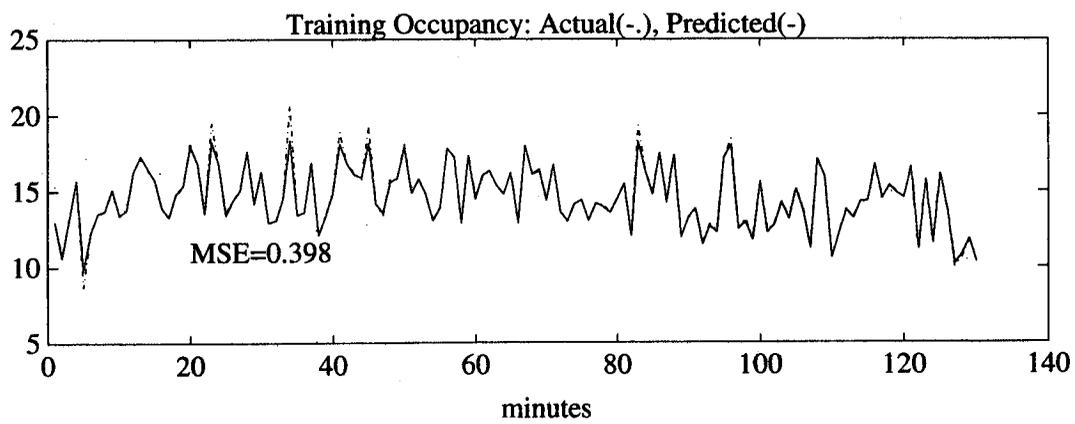
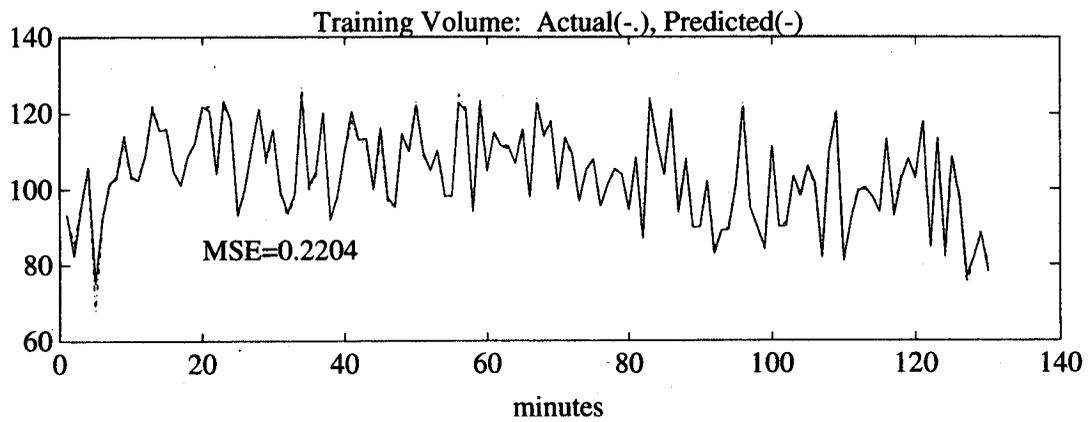


Figure 13. Site 1 Results with Training Day 8/27

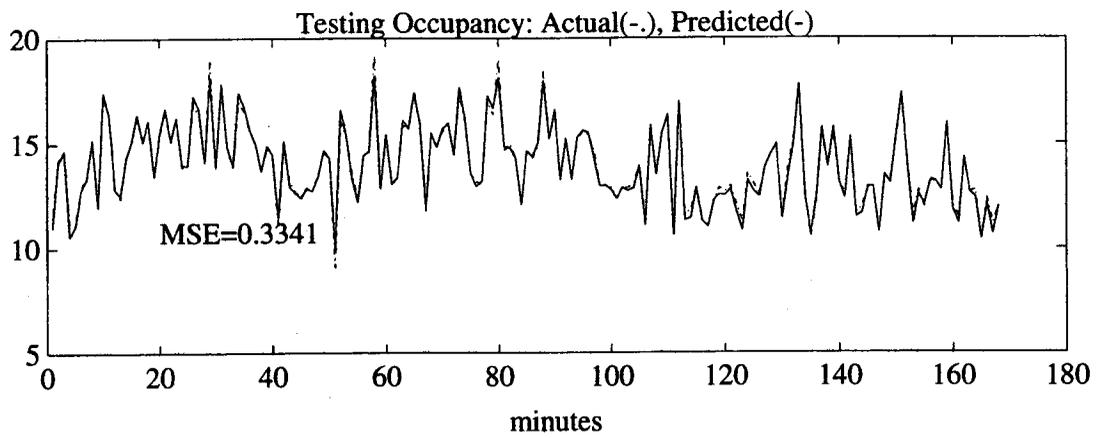
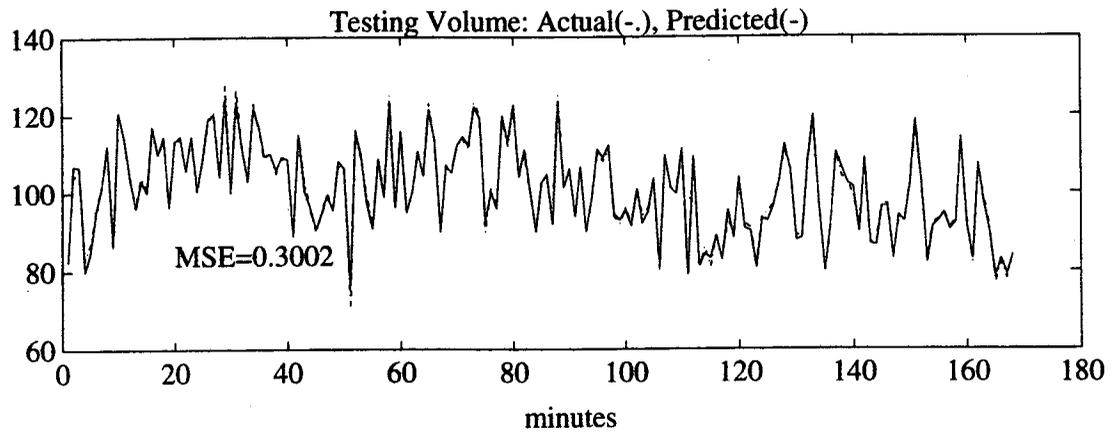


Figure 14. Site 1 Results with Training Day 8/27 and Testing Day 8/28

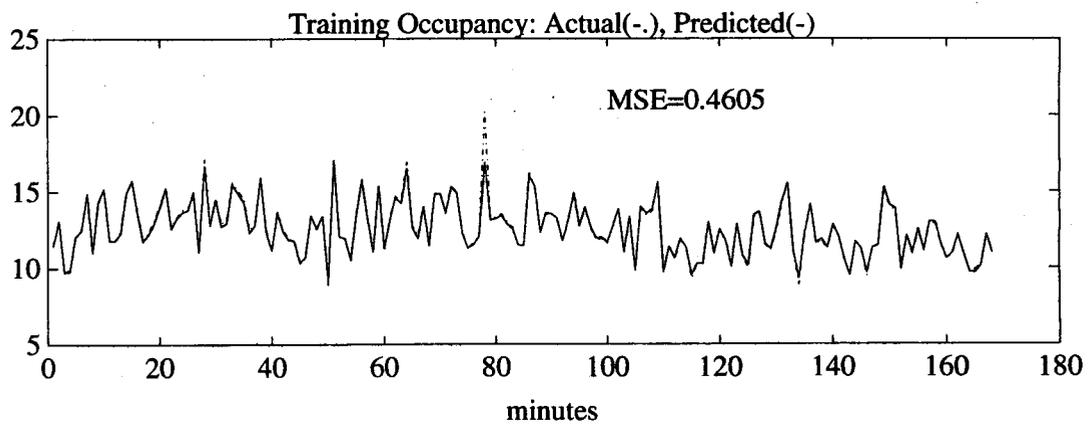
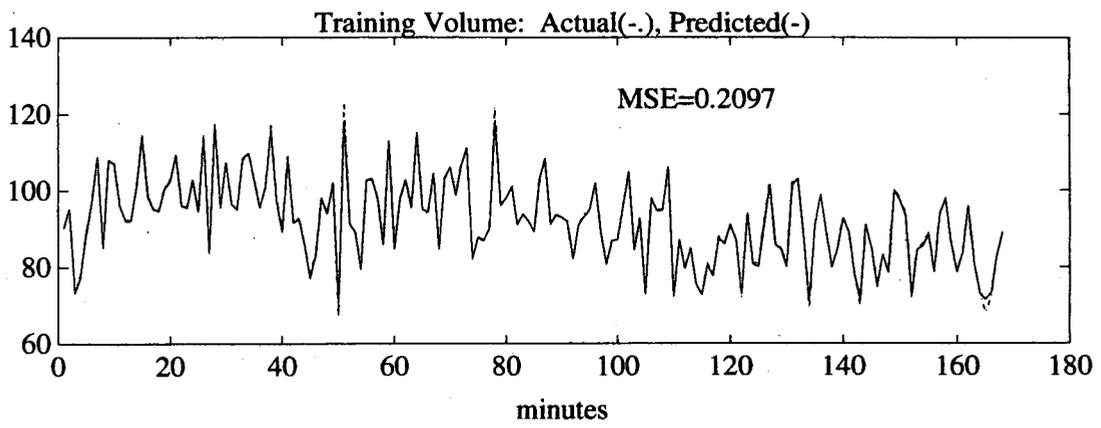


Figure 15. Site 2 Results with Training Day 8/28

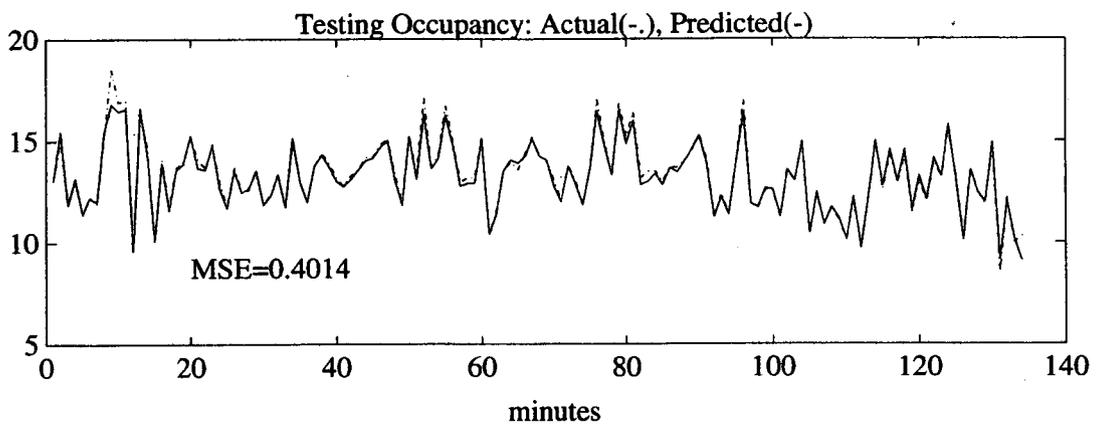
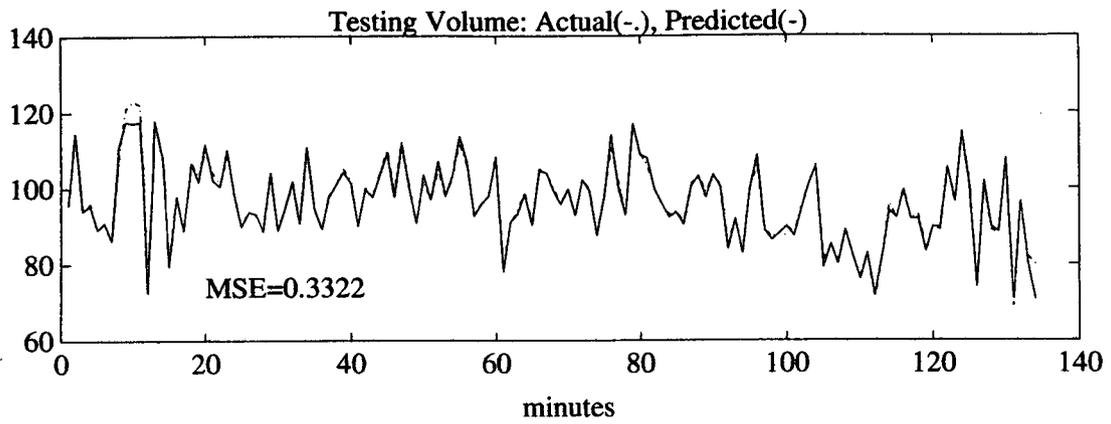


Figure 16. Site 2 Results with Training Day 8/28 and Testing Day 6/1

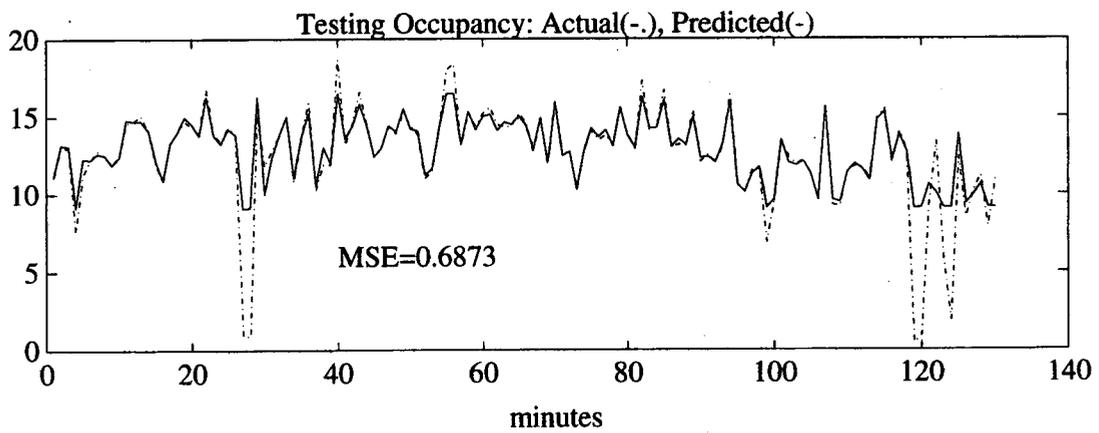
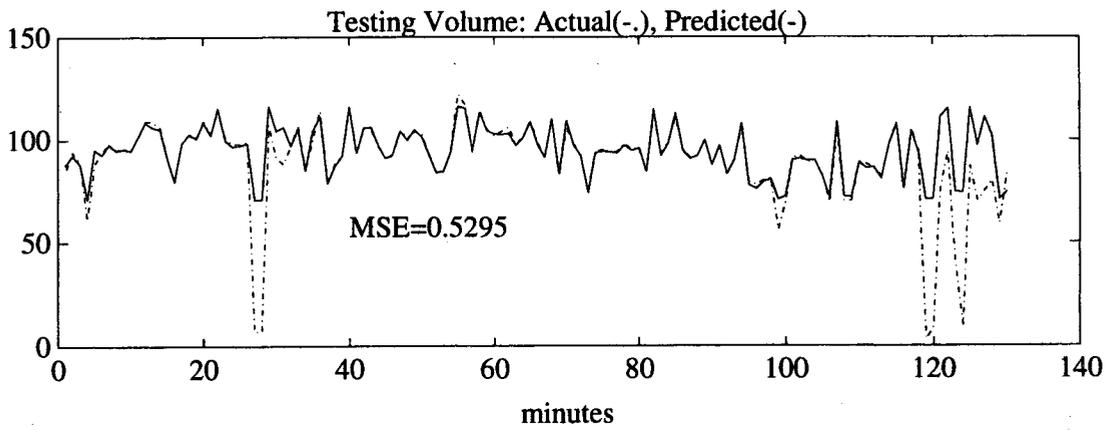


Figure 17. Site 2 Results with Training Day 8/28 and Testing Day 8/27

The next figure shows the same ANN trained on the same day, but tested on a different day. Convergence, defined as the optimal number of sweeps to minimize testing MSE, was determined on testing day June 1, 1992, so this second testing set on August 27 further verified generalization. It illustrated how this ANN responded to irregular data, demonstrating robustness. The three valleys near zero, as discussed in the data scaling discussion, were probably transmission errors. The ANN output at these valleys was nearly zero before rescaling. Thus, the ANN produced the minimal allowable value when an unusually low peak occurred. Notice how the ANN only required a few minutes to regain an accurate prediction.

Long-Term Prediction Results

Longer term prediction would be useful to a ramp metering algorithm to indicate general trends, but it is more difficult to obtain accurately. The prediction difficulty increases dramatically with time. For example, predicting 2 minutes in advance turns out to be more than twice as difficult as predicting 1 minute in advance. The reason is random variation in traffic flow and driver behavior. As the distance that a vehicle can travel during the forecasted period increases, the random inputs add cumulatively. Because of the somewhat chaotic nature of traffic data, the length of time over which traffic behavior can be predicted is limited. Although several methods for longer term prediction were explored, none of these ANNs successfully generalized to new data. Appendix B shows results from 2 and 5 minute predictors, discusses why various techniques were inadequate, and recommends methods for future long-term prediction attempts.

Despite these results, longer term prediction may still be possible. In fact, Dougherty's research group, discussed previously, was able to predict 5 minutes in advance, but was unsuccessful with 1-minute predictions for the A2 Motorway in the Netherlands (Dougherty and Kirby, 1993). An explanation for this paradox is that a trade-off exists between smoother data with a greater sampling period, and the ease in

predicting over a shorter time span. Five-minute data have smoother trends because the volume is an accumulation over the sampling period, while the occupancy is an average over the sampling period. However, an ANN requires a vast amount of past upstream information to predict 5 minutes in advance using 5-minute data because a vehicle can travel over 5 miles during the forecasted period. Twenty second data turn out to be more difficult to predict than 1-minute data because of the more extreme fluctuations in volume and occupancy. During an incident, volume can decrease to zero, and occupancy may reach 100 percent. For the Seattle area, 1-minute data predictions appeared to be a satisfactory compromise between these trade-offs. Dougherty's research used a different type of data (speed, as well as volume and occupancy) with a different preprocessing technique (a moving average of the past 1-minute samples). Because of variations among sites, it is not surprising that the most practical prediction period would vary for specific applications.

ANN IMPLEMENTATION CONSIDERATIONS

This research was conducted with implementation on Seattle's Traffic Systems Management Center (TSMC) in mind. Because the ramp metering algorithm the TSMC currently uses is written in C, all neural network code was written in C. Although training the neural network may require an hour or so, the trained neural network produces its prediction in a fraction of a second. Thus, the network would have no problem performing in the 20-second sampling period. These neural networks use historical data to test prediction accuracy, but real-time prediction would be ideal for TSMC implementation. For on-line predictions, the input/output portion of the neural network code would need modification. Currently, the network reads files containing the information it needs. Each data file contains volume and occupancy of a particular station for one day during the desired time frame. Since the incoming on-line data would be in a different form, the file reader subroutine would also have to be altered. The network code outputs training and prediction results to a file. The TSMC would require a

different results format. The training and testing portion of the neural network should need only slight modifications.

Extension to Seattle I-5

Because this research only predicted traffic data at two sites on I-5, more neural networks would have to be trained to predict traffic at the remaining mainline sites with metered on-ramps. Each neural network would predict volume and occupancy at one mainline site. The training algorithm and basic MLP architecture should remain the same for different sites. Although each neural network would need to be trained with data from that particular site, the parameters should be similar as long as the data characteristics were similar to the research site. If not, the number of hidden neurons, tapped delay line length, number of training iterations, and learning rates might need alteration. The network code was written to allow easy modification of these parameters.

Role in Ramp Metering Algorithm

The fuzzy logic ramp metering algorithm would simply use the neural network prediction as one of the algorithm inputs. The metering rates would then be based on both actual and predicted data. When the ramp metering algorithm used the prediction as an input, a flag would have to constantly monitor the accuracy of the prediction. The flag could monitor the mean squared error of the past several predictions from the actual data. If the prediction accuracy did not maintain certain criteria, the flag would notify the ramp metering algorithm, which would then no longer rely on the prediction input. When the prediction regained accuracy either by a change in data characteristics or retraining of the ANN, the flag would turn off and the ramp metering algorithm would start using the prediction again.

One complication with the neural network predictor is that it was trained on bottleneck ramp metered data. The fuzzy logic ramp metering algorithm would alter the traffic characteristics. Although the neural network successfully predicts bottleneck ramp metered data, it would need to be retrained on the fuzzy logic ramp metered data to

accurately predict the new traffic patterns. Initially, a cyclic settling process would be expected, in which the retrained network would predict more accurately, which, in turn, would produce better metering rates. These new metering rates would again alter the traffic characteristics. Eventually, the neural network would settle into a prediction of the fuzzy logic ramp metered traffic rather than the bottleneck ramp metered traffic, and should then seldom require retraining. Training would occur continually using a recent set of historical data, but the most recently trained network would only replace the old network when the prediction remained inaccurate for a specified time. It might be useful to keep track of weight sets from each neural network. Over time, combining the weight sets from previous neural networks by some form of averaging might produce a more robust predictor. Weight sets from several months might provide insight into the underlying statistical process throughout seasonal variations.

ANN CONCLUSIONS

The neural networks predicted volume and occupancy significantly better than previous techniques used in the Seattle area. Test results on historical data from the I-5 freeway in Seattle, Washington, demonstrated that a neural network can accurately predict volume and occupancy 1 minute in advance, as well as fill in the gaps for missing data with an appropriate prediction. The volume and occupancy predictions will be used as inputs to a fuzzy logic ramp metering algorithm currently being tested. A 1-minute data prediction is a useful input to a ramp metering algorithm because this insight can help delay or prevent bottleneck formation. Since the ramp metering rates are updated every 20 seconds, a 1-minute data prediction can provide valuable information to determine the next few metering rates.

A multi-layer perceptron type of ANN was trained using back propagation to minimize the mean squared error of the prediction. The inputs to the ANN included the previous ten values of 1-minute volume and occupancy from the predicted station and the adjacent upstream station. Although the ANNs were trained and tested on historical data,

they can be implemented for real-time prediction because the inputs are past values of volume and occupancy. For on-line implementation, a neural network will need to be trained from data for each prediction site. The training algorithm and neural network architecture should remain similar for different sites. The ANN parameters should remain similar as long as the data characteristics are similar to the research sites. If not, the network code has been written to allow easy modification of the ANN parameters. For on-line implementation, the neural network should be trained on-line to allow for seasonal variations. A flag should constantly monitor the accuracy of the prediction to indicate whether retraining is necessary.

Predictions over 1 minute were unreliable. Because of the somewhat chaotic nature of freeway traffic data, a longer term prediction using ANNs is a much more difficult problem. Given that a vehicle may travel over 5 miles in a 5-minute forecasted period, the random inputs over that time and distance make longer term prediction challenging.

FUZZY LOGIC RAMP METERING ALGORITHM

SEATTLE'S CURRENT RAMP METERING ALGORITHM

Seattle's current ramp metering algorithm, one of the most sophisticated in the country, helps ease freeway congestion, but it could benefit from further improvements. This section encompasses the operation of the existing Seattle ramp metering algorithm and the reasons that fuzzy logic control could potentially overcome the algorithm's shortcomings.

Operation of the Current Seattle Algorithm

The factors that make Seattle's 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 currently 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. 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 is activated when the following two criteria are met: 1) a downstream bottleneck-prone section surpasses a predetermined occupancy threshold, and 2) that section stores vehicles. A section stores vehicles when more vehicles enter the section than leave the section. When these two criteria are met, the algorithm reduces the number of vehicles entering the freeway by the number of vehicles being stored in the bottleneck 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 the volume reduction targeted for that ramp.

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 entries per minute from the metering rate. For more details on Seattle's ramp metering system, see other literature (Jacobson, Henry, and Mehyar, 1988).

Suitability of Fuzzy Logic to the Current Algorithm

This research sought to overcome limitations in the current Seattle algorithm. One problem with the current algorithm is that it reacts to existing bottlenecks rather than preventing them. An algorithm with predictive capabilities could help prevent or delay bottleneck formation. Hence, the accurate, 1-minute neural network prediction would be a powerful asset to the ramp metering algorithm. An additional problem with the existing algorithm is that it gets caught in a cycle between queue override activations and restrictive metering rates. The fuzzy logic ramp metering algorithm could combat this problem by providing smooth transitions rather than threshold activation. In addition, a fuzzy logic controller could evaluate several rules in parallel and then determine one metering rate based on all factors rather than making a series of adjustments.

Another disadvantage of the current algorithm is its strong dependency on loop detector data. Loop detector data are commonly unavailable or erroneous. For this

reason, algorithm robustness is essential. Fuzzy logic controllers produce appropriate outputs given uncertain or incomplete information. The defuzzification process, which uses a centroid calculation, helps suppress parameter variations and stochastic disturbances. A parallel rule evaluation also promotes robustness. Even if an appropriate rule does not fire because of faulty input data, a different rule can produce the same output. For instance, two of the rules within the knowledge bank might be "If upstream occupancy is high, reduce the metering rate" and "If predicted occupancy is high, reduce the metering rate." Even if the loop detector upstream temporarily failed, the predicted occupancy could still reduce the metering rate, since the prediction would be based on data from the past 10 minutes of two stations.

Fuzzy logic is well-suited to ramp metering for several more reasons. The rule base, defined as the set of rules in the fuzzy logic controller, incorporates human expertise. For example, a ramp metering expert might say, "If a downstream bottleneck is forming, reduce the metering rate." Since these rules are easy to alter or eliminate, fuzzy logic allows simple development and modification. Fuzzy logic control is especially suitable when an accurate system model is unavailable. Without doubt, the freeway's complexity, nonlinear nature, and nonstationary behavior makes obtaining a model extremely difficult. Most traditional controllers are only as good as the system model and usually force nonlinear systems into a linear context. Because a fuzzy logic controller can handle nonlinear systems with unknown models, it has a distinct advantage over traditional controllers for the ramp metering problem. In addition, higher order, nonstationary, and MIMO (multi input/multi output) systems do not impose limitations on a fuzzy logic controller.

There are also disadvantages to fuzzy logic controllers. One disadvantage is that the design process depends on the developer's knowledge and abilities. Fuzzy logic controller design is not as methodical as traditional controller design; the design process requires on-line tuning of rules and membership functions. However, on-line tuning of

any ramp metering algorithm is expected, as no model can perfectly replicate freeway behavior. The completeness of the rule base when a MIMO system is encountered needs consideration. With several inputs to the fuzzy logic ramp metering algorithm, the designer must be careful that the algorithm accounts for any possible situation. Overall, the disadvantages are minor when compared to the advantages of using fuzzy logic control for the ramp metering application.

FUZZY LOGIC

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 clothing washers 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 18 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
PB	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 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. Complementation 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.

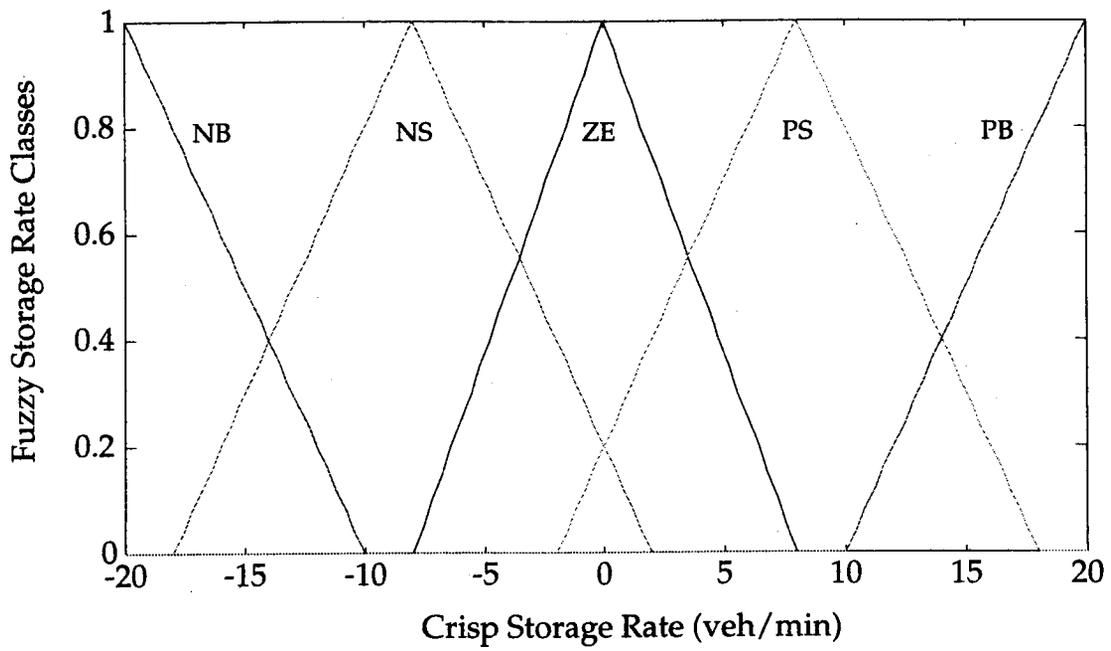


Figure 18. Storage Rate Membership Classes

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.

These rules can be rewritten more compactly using the variables in Table 2. The variables in parenthesis 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]

Table 2. 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

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. At this point, each output variable class is implicated to a degree.

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,

$$\min(w, f(x)), \quad \text{Eq. 13}$$

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:

$$w * f(x) \quad \text{Eq. 14}$$

Figure 19 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.

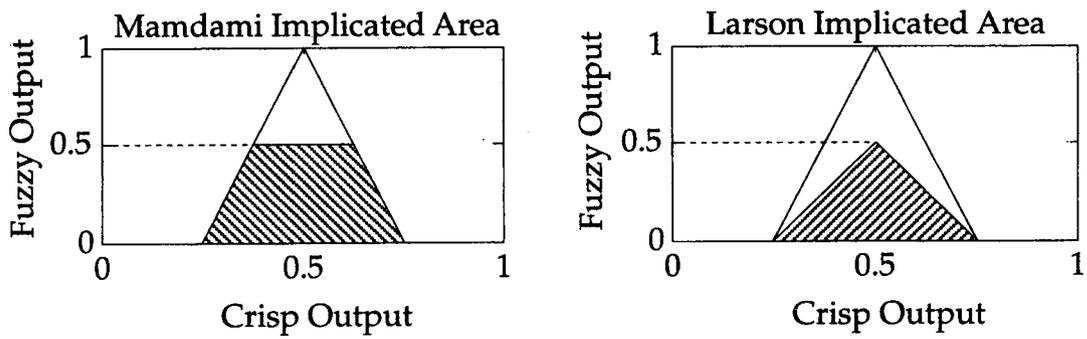


Figure 19. Correlation-Minimum (left) and Correlation-Product (right) Implication Methods

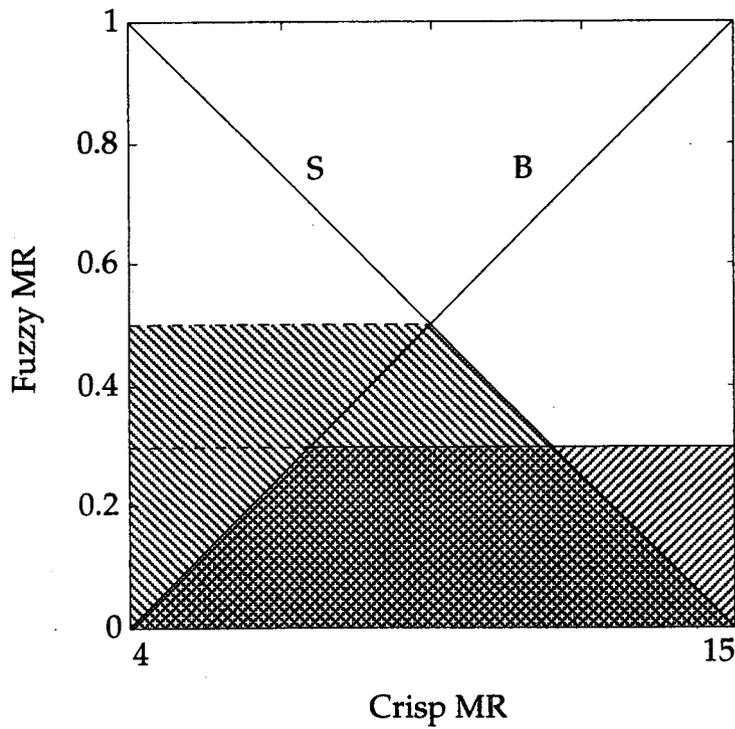


Figure 20. Centroid Defuzzification

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:

$$\frac{\int xf(x)dx}{\int f(x)dx} \quad \text{Eq. 15}$$

Figure 20 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.

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):

$$\frac{\sum_{i=1}^N w_i c_i I_i}{\sum_{i=1}^N w_i I_i} \quad \text{Eq. 16}$$

where c_i is the centroid and I_i is the area of the output class for the i th rule.

DESCRIPTION OF THE FUZZY LOGIC RAMP METERING ALGORITHM

Flexibility was a key issue in the design of the fuzzy logic ramp metering algorithm. For reasons that will be discussed, WSDOT is contracting out the work to incorporate the fuzzy logic algorithm into a freeway model. For this reason, additional source code modifications would be time consuming, so the initial algorithm had to be designed to be flexible. Because the purpose of model testing is to tune the algorithm by trial and error, the class definitions and rules had to be designed for easy modification.

It is desirable to have a minimum number of inputs and rules. The initial fuzzy ramp metering algorithm contains a reasonable number of inputs and rules that may be useful for control, and the testing and tuning process with a model will determine which of these inputs and rules are useful. The inputs and rules that are unnecessary may be eliminated without making extensive source code modifications.

Table 3 describes the variables used as inputs to the fuzzy logic controller, and Figure 21 shows the location of each input variable. All input variables are based on a 20-second sampling period, unless stated otherwise in the table. Mainline variables are based on loop detectors across all lanes. For example, volume is in vehicles per 20 seconds accumulated across all lanes. The predicted occupancy is a neural network output.

Each variable has several parameters that add flexibility to the class definitions. 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:

$$\text{scaled crisp variable} = \frac{\text{crisp variable} - LL}{HL - LL} \quad \text{Eq. 17}$$

The scaling simplifies the code by allowing all variables to use the same fuzzification equations, as well as allowing easy class modification. By increasing *LL* and decreasing *HL*, the sensitivity to a particular input can be increased, which causes the rules with that premise to fire to a greater degree.

The fuzzification process translates each of the controller input variables into 5 classes (Table 4). In addition to limit parameters, each variable class has a centroid (C_i) and width (β_i) parameter to define the *i*th class. The NS, ZE, and PS classes are defined by an isosceles triangle with a base of $2\beta_i$ and a height of 1. The triangle is centered at

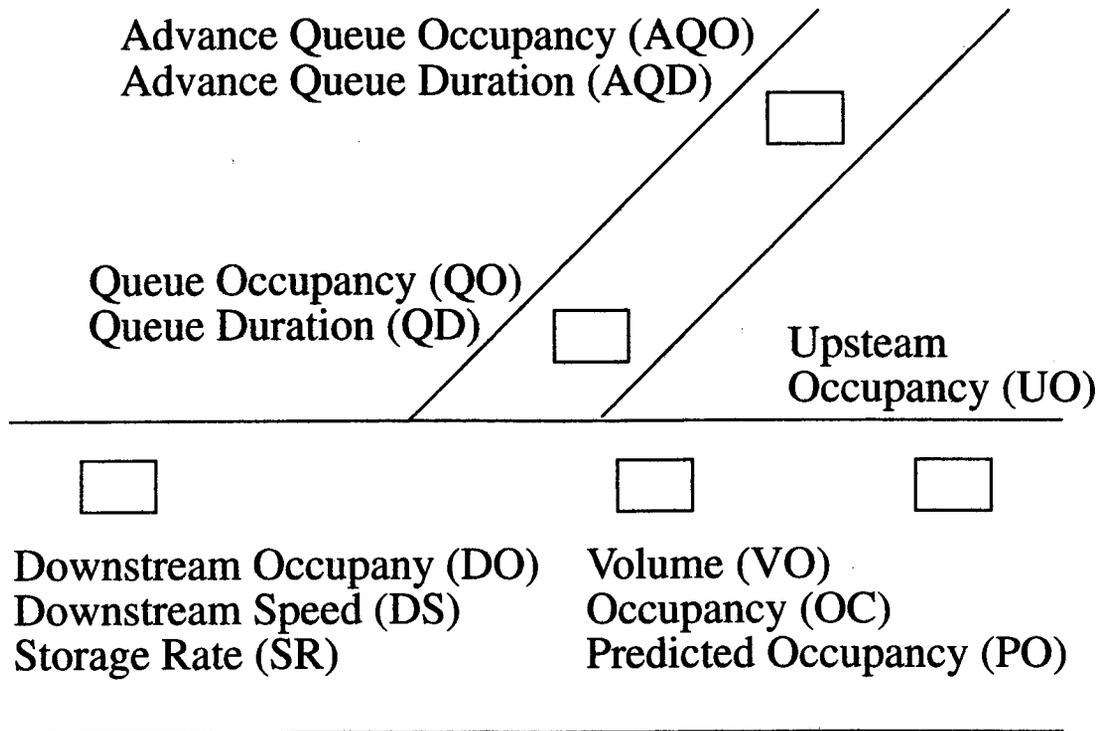


Figure 21. Location of Algorithm Variables

Table 3. Description of Algorithm Variables

Variable	Description
VO	mainline volume just before ramp outlet
OC	mainline occupancy just before ramp outlet
DO	downstream occupancy of nearest bottleneck prone section
UO	upstream occupancy for adjacent 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
SR	downstream storage rate of nearest bottleneck prone section
QO	ramp queue occupancy over the past sample
QD	ramp queue occupancy averaged over the past 6 samples
AQO	advanced queue detector occupancy over past sample
AQD	advanced queue detector occupancy averaged over past 3 samples
MR	metering rate (the control action)

Table 4. Fuzzy Classes

i	Class	Description
1	NB	negative big
2	NS	negative small
3	ZE	zero
4	PS	positive small
5	PB	positive big

C_i and has slopes of $\pm \frac{1}{\beta_i}$. The resulting fuzzy classes are defined by the scaled crisp variable, x . For NS, ZE, and PS,

$$f_i(x) = \begin{cases} \frac{1}{\beta_i}(x - C_i + \beta_i) & \text{for } C_i - \beta_i < x < C_i \\ -\frac{1}{\beta_i}(x - C_i - \beta_i) & \text{for } C_i < x < C_i + \beta_i \end{cases} \quad \text{Eq. 18}$$

As in Figure 20, a right triangle defines NB and PB. For NB, the peak is at 0, so C_i is $\frac{\beta_i}{3}$. The class is 1 if x is less than 0. For NB,

$$f_i(x) = \begin{cases} 1 & \text{for } x < 0 \\ -\frac{1}{\beta_i}(x - \beta_i) & \text{for } 0 < x < \beta_i \end{cases} \quad \text{Eq. 19}$$

For PB, 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 PB,

$$f_i(x) = \begin{cases} \frac{1}{\beta_i}(x - 1 + \beta_i) & \text{for } 1 - \beta_i < x < 1 \\ 1 & \text{for } x > 1 \end{cases} \quad \text{Eq. 20}$$

An example of parameter values to define the set of fuzzy classes for SR (shown in Figure 18) are $LL=-20$, $HL=20$, $C=[0.083, 0.3, 0.5, 0.7, 0.916]$, and $\beta=[0.25, 0.25, 0.2, 0.25, 0.25]$. Table 5 shows a sample input card containing the parameters that are user specified. Default values are shown. One input card is required for each on-ramp that uses fuzzy control metering. Notice that C_NB and C_PB are not user specified, as they can be calculated from B_NB and B_PB .

Like the class definitions for each variable, the rules need flexibility. Hence, each rule is assigned a weight, which can be set to zero to eliminate that rule (See Table 5). If a rule is more important than other rules, it can have a greater weight. For example, the advance queue override rules 7c and 7d in Table 5 may have a weight of 2, while most other rules have a weight of 1. To avoid future source code modifications, the initial

Table 5. Parameter Input Card

Fuzzification Parameters

	LL	HL	C_NS	C_ZE	C_PS	B_NB	B_NS	B_ZE	B_PS	B_PB	
V	VO	150	185	0.3	0.5	0.7	0.25	0.25	0.2	0.25	0.25
a	OC	8	18	0.3	0.5	0.7	0.25	0.25	0.2	0.25	0.25
r	DO	8	18	0.3	0.5	0.7	0.25	0.25	0.2	0.25	0.25
i	UO	8	18	0.3	0.5	0.7	0.25	0.25	0.2	0.25	0.25
a	PO	8	18	0.3	0.5	0.7	0.25	0.25	0.2	0.25	0.25
b	SP	45	65	0.3	0.5	0.7	0.25	0.25	0.2	0.25	0.25
l	DS	45	65	0.3	0.5	0.7	0.25	0.25	0.2	0.25	0.25
e	SR	-15	15	0.3	0.5	0.7	0.25	0.25	0.2	0.25	0.25
s	QO	10	60	0.3	0.5	0.7	0.25	0.25	0.2	0.25	0.25
	QD	10	60	0.3	0.5	0.7	0.25	0.25	0.2	0.25	0.25
	AQO	5	10	0.3	0.5	0.7	0.25	0.25	0.2	0.25	0.25
	AQD	5	10	0.3	0.5	0.7	0.25	0.25	0.2	0.25	0.25
	MR	2	5	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
Weight	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Rule #	15	16	17	18	19	20	21	22	23	24	25	26	27
Weight	1	1	1	1	1	1	1	1	1	1	1	1	1

ramp metering algorithm contains a large number of rules which that will be tested in simulations (Table 6). By testing this algorithm on a freeway model, this large rule base can be pared down to as few rules as possible. For each rule in the table, the intersection of the premises involves using the minimum of the premise degrees as the output degree. To combine rules that produce the same output class, the *additive* method is used rather than the *maximum* method. The *additive* method is expected to be 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 will average together each rule contribution. Another factor to consider is that this rule base does not individually consider all of the possible 5^{13} input combinations. Instead, the rule base is completed by specifying a metering rate given any value of occupancy or upstream occupancy. By averaging rule outcomes together, the control action should be smoother.

Related rules are grouped together. Rules 1a through 1e address the concern for the completeness of the rule base. Because the entire range of occupancy inputs is considered, at least one of these five rules should fire. If OC is not available, the predicted occupancy rules 2a and 2b and the upstream occupancy rules 3a through 3e can produce a similar output. Volume is not used as a premise for rule 1 because occupancy is a more reliable indicator of congestion. However, volume is used in the speed calculation, which is also a reliable indicator of congestion. Rules 4a through 4d adjust the metering rate on the basis of mainline speed.

Rules 5 and 6a through 6d are devoted to preventing or delaying downstream bottleneck formation. Rule 5 emulates Seattle's existing bottleneck algorithm, in which high storage rate and high occupancy downstream indicate a bottleneck. However, plots of storage rate show that it is not an accurate indicator of congestion (Nihan 95). Storage rate fluctuates around zero whether in light traffic or heavy congestion. This traffic behavior agrees with intuition. Consider that storage rate is the number of vehicles being added to a freeway section during a sampling period. The number of vehicles that can fit

Table 6. Rule Base for Fuzzy Ramp Metering Algorithm

Rule	Premise	MR outcome
1a	OC_PB	NB
1b	OC_PS	NS
1c	OC_ZE	ZE
1d	OC_NS	PS
1e	OC_NB	PB
2a	PO_PB	NB
2b	PO_NB	PB
3a	UO_PB	NB
3b	UO_PS	NS
3c	UO_ZE	ZE
3d	UO_NS	PS
3e	UO_NB	PB
4a	SP_NB, OC_PB	NB
4b	SP_NS	NS
4c	SP_PS	PS
4d	SP_PB, OC_NB	PB
5	SR_PB, DO_PB	NB
6a	DS_NB, DO_PB	NB
6b	DS_NS, DO_PS	NS
6c	DS_ZE, DO_ZE	ZE
6d	DS_PS, DO_NS	PS
6e	DS_PB, DO_NB	PB
7a	QO_PB	PS
7b	QD_PB	PB
7c	AQO_PB	PB
7d	AQD_PB	PB

into a freeway section has a limit, so the average storage rate over a long time is zero. Even in stop and go traffic, vehicles must still exit the bottleneck section. Because vehicles tend to travel in platoons, the storage rate oscillates rapidly between positive and negative. The fuzzy algorithm uses downstream speed and occupancy inputs, which are superior indicators of mainline congestion (Iwasaki 1991; Masher, Ross, Wong, Tuan, Zeidler and Petracek, 1975)." Rules 6a through 6e use the premise that high downstream occupancy and low downstream speed indicate bottleneck conditions. For an improved bottleneck indicator, rules 6a through 6e can be used in place of rule 5.

Rules 7a through 7d in Table 5 address ramp queue occupancy and duration. Seattle's current algorithm is susceptible to cycling between restrictive and high metering rates during peak hours. Rules 7a through 7d provide smooth transitions rather than threshold activations. When the queue occupancy is high, the metering rate increases. A high queue duration over the past six samples indicates that the queue is building up over time. Unlike the threshold activation of Seattle's current algorithm, the queue duration input provides qualitative information regarding the queue formation. With this information, rules 7a through 7d can help prevent queue formation and avoid an oscillatory metering rate. The advance queue override detector for rules 7c through 7d is located closer to the arterial than the queue override detector for rules 7a and 7b, so rules 7c and 7d should be weighted more heavily to prevent vehicles from backing up into the arterial.

For calculation simplicity, the implication method used is *correlation-product* encoding rather than *correlation-minimum* encoding. Correlation-product inference allows use of the discrete centroid equation,

$$\frac{\sum_{i=1}^N w_i c_i I_i}{\sum_{i=1}^N w_i I_i} \quad \text{Eq. 21}$$

where c_i is the centroid and I_i is the area of the output class for the i th rule. Given that this FLC has a large number of inputs and rules and requires flexibility, the discrete centroid equation is used for code simplicity. For MR classes of NS, ZE, and PS, the class area I_i equals β_i . For MR classes of NB and PB, the class area I_i equals $\beta_i/2$. Once the centroid of the crisp MR has been found, this control action must be rescaled back to its dynamic range using the scaling equation. Like Seattle's bottleneck algorithm, the MR should then be adjusted to account for the number of HOVs 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, for limits of LL=2 and HL=5 vehicles/sample. Appendix C contains the C code for the fuzzy logic controller described in this section.

TESTING THE RAMP METERING ALGORITHM

Because fuzzy logic controllers require tuning, the fuzzy logic ramp metering algorithm must be tested with a model and/or on-line. Testing the algorithm with a model is highly desirable before on-line testing. The absence of random variations with simulation testing allows algorithms to be tuned and compared more easily than does on-line testing. However, problems such as the model inaccuracy, limitation, and calibration difficulty reduce the validity of model tested results.

The freeway models currently available for testing ramp metering with closed-loop control are limited. Significant source code modifications would be necessary to incorporate the fuzzy logic controller into any of the freeway models explored. Although freeway models have been used in Seattle to determine HOV lane entry and exit

placement and to determine whether ramp metering should be added to a specific freeway section, model testing of comprehensive closed-loop control ramp metering has not been done in Seattle. To the authors' knowledge, no models have been calibrated for a ramp metered section of a Seattle freeway to simulate the bottleneck ramp metering algorithm. Because of model limitations, the existing Seattle ramp metering algorithm was tuned on-line. Similarly, other researchers have bypassed model testing for their ramp metering algorithms (Nihan and Berg, 1991; Washburn, 1993). Nevertheless, because of driver impact concerns and other benefits of simulation testing, it is worthwhile.

After the algorithm has been tuned through simulation testing, the algorithm should then be tested further on-line. Unfortunately, on-line ramp metering testing can potentially affect thousands of drivers. Any on-line testing must be conducted with great caution to reduce commuter impact and safety hazards. Variable traffic conditions further complicate on-line testing. An algorithm tester has no way of knowing what the results would have been if one algorithm had been used instead of another. If new algorithm testing happens to fall on less congested days, the new algorithm performance may be deceptively better. Hence, the algorithm testing must take place over a several-day span so that average performance can be compared to other algorithms.

Given its time constraints, this project did not test the algorithm's performance, but it did explore the best model testing options and on-line testing options for future research. Algorithm testing, first through simulation and then on-line, is being continued through a TransNow 1994-95 grant. The following sections 10.1 through 10.3 elaborate on the work in progress for simulation testing and recommend an on-line testing procedure.

Freeway Model Testing

Freeway Model Options. Developing a freeway model is quite involved, so using existing models is most logical. There are two types of freeway models: macroscopic and microscopic. While microscopic model calibration requires adjusting

driver behavior, macroscopic model calibration requires adjusting capacity, the number of vehicles per lane per hour. Macroscopic models consider platoons, rather than individual vehicles. Both macroscopic and microscopic calibration require adjusting speeds, which means driving the freeway section numerous times to determine typical speeds. The freeway models most appropriate for this application are FREQ (May 1993), FRESIM (Hassan and Torres, 1990), and INTRAS (Wicks and Lieberman, 1980).

FREQ, a macroscopic freeway model, allows demand, supply, and control modification. Although FREQ presents the best macroscopic model testing option, it is limited in two basic ways for this application: length of run and closed-loop control unavailability. FREQ can simulate up to 24 continuous time slices, and the intervals can be set by the user. This application requires a 20-second time slice interval to equal the sampling period for the existing Seattle freeway system, so a maximum length run is only 8 minutes. Ideally, the algorithm needs to be tested over a continuous 3-hour peak commute. If runs could be made continuous, FREQ could simulate more than 8 minutes. Disappointingly, FREQ simulations cannot be made continuous, as queues cannot be initialized to the final conditions of the previous run. Although demand can be specified, FREQ initializes the queues to zero. With queues forming during each simulation, this initialization set-up is not realistic for long-term testing. Despite this drawback, an 8-minute simulation may still be useful if the model is accurate and well-calibrated. Using different demands for each simulation, FREQ could test algorithm performance under various conditions.

FREQ's other limiting factor is the unavailability of closed-loop control. FREQ accepts an external input file with ramp metering rates that are specified for each time slice. However, the metering rates cannot be altered once a simulation begins. In other words, the control is open-loop rather than closed-loop. The ramp metering rates cannot be specified at every sampling period on the basis of current information. The best scenario for imitating closed-loop control would involve doing 24 simulations, altering

the inputs for one time slice between each run. After the first simulation, the fuzzy logic controller would determine the metering rates for the first interval on the basis of data from that time, and it would alter the input file accordingly. After the second simulation, the controller would determine the metering rates for the second interval, and again it would alter the input file. In this way, the input file would contain the closed-loop metering rates after 24 simulations for an 8-minute simulated run. The other user interface option specifies objectives and lets FREQ determine the metering rates. This second method would allow comparison of the fuzzy logic algorithm results with FREQ generated results.

A microscopic model is more practical for the ramp metering application because it keeps track of individual vehicles. FRESIM, which is written in FORTRAN for a PC, is the most suitable of the microscopic models. Although FRESIM does not currently allow an external closed-loop controller, adding it would not require an unreasonable amount of source code modification. FRESIM allows the user to choose one of four control criteria: pretimed, speed, gap acceptance merge, and capacity. FRESIM decides the metering rate on the basis of the chosen criteria. Testing fuzzy logic ramp metering would require adding the new controller as a user option within the source code. The fuzzy logic controller results could be compared to that of the other four controllers. Like FREQ, the simulation is limited to a certain number of periods. But with source code modifications, the simulation could be given metering rates at a specified time step within each time interval. Point processing would calculate data at these time steps.

INTRAS is another suitable microscopic freeway model for testing ramp metering. FRESIM is an enhanced version of INTRAS that can interface with a surface street model, among other improvements. Like FRESIM, a closed-loop controller could be added to INTRAS with source code modification. Since INTRAS and FRESIM both have over 90,000 lines of code with very few comments, adding another controller would be no small task to someone unfamiliar with the source code. The portions of code

needing modification are not isolated, so locating the relevant code to add a controller could be difficult. The preference for FRESIM over INTRAS is based on programming support for FRESIM. The Federal Highway Administration (FHWA) has agreed to support the source code modifications necessary to incorporate the fuzzy logic controller into FRESIM. The senior technical advisor for the programming modifications is Hassan Halati, a FRESIM developer (Hassan and Torres, 1990) of the Viggen Corporation, while the actual programming would be done by Steven Chien of the Viggen Corporation. Although the fuzzy ramp metering algorithm is designed to allow any foreseeable adjustments, additional source code changes would most likely be necessary. Consequently, model testing would require a copy of the source code, with permission from FHWA, and the ability to compile it. The programming support for FRESIM makes FRESIM a more efficient and cost-effective choice.

FRESIM Modifications. The modifications to incorporate the fuzzy logic controller (code in Appendix C) into FRESIM are estimated to take an experienced FRESIM programmer two months. The first capability that would have to be added to FRESIM is the ability to call the fuzzy logic controller to obtain a metering rate every 20 seconds. The fuzzy logic controller would be added as one of the ramp metering options. The user specified input parameters (Table 5) would simply have to be passed as arguments to the controller.

The most formidable FRESIM modification necessary is obtaining the inputs to the controller every 20 seconds. Currently, FRESIM does not allow a time interval of less than 1 minute. Point processing would be needed to bring information out of memory in between time intervals. FRESIM does not presently offer any ramp data, so adding the queue occupancy and duration inputs would require significant programming.

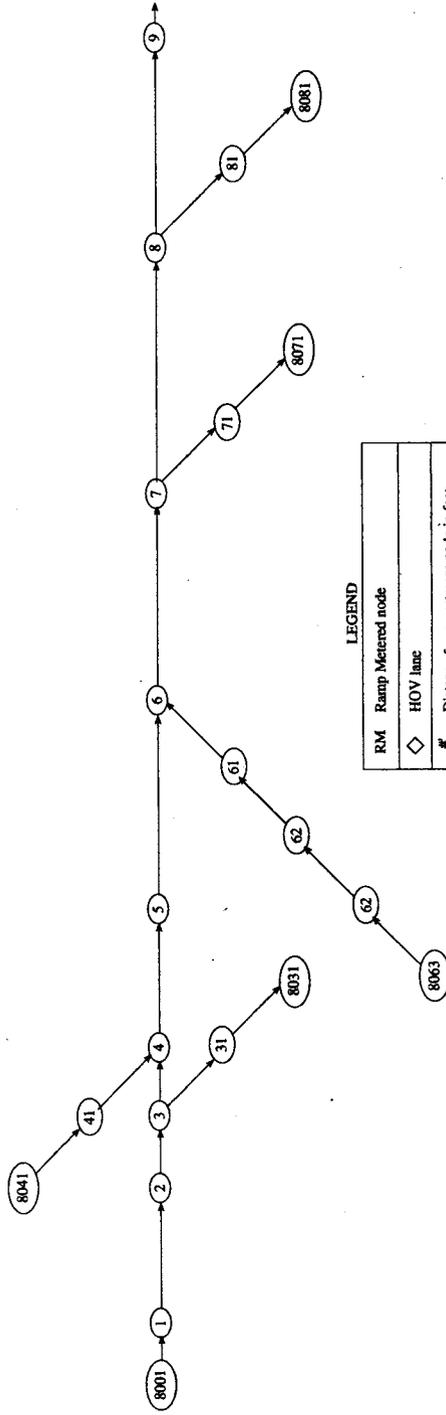
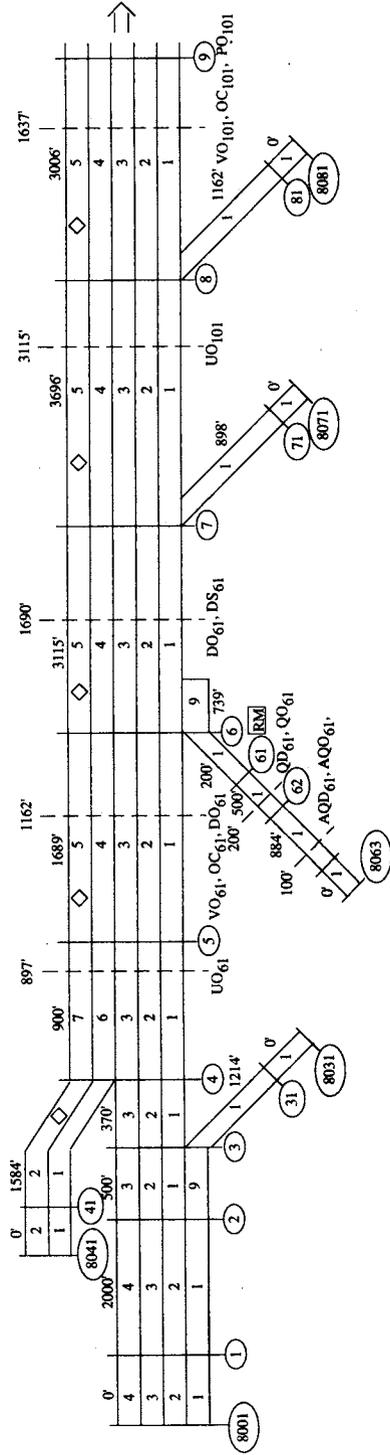
Obtaining the prediction of occupancy 1 minute in advance is another notable programming task. Because the neural networks have already been tested on actual historical freeway data, they would not need to be retested on FRESIM. However, the

fuzzy logic controller would still require this predictive input. By using background simulation in which a coprocessor ran the simulation one step in advance, FRESIM could obtain predicted occupancy. This way, the neural network would not have to be incorporated into the controller, yet the controller would still be proactive rather than reactive. A limitation of FRESIM is that it cannot directly model HOV lanes at the present time, but this capability is currently being added to FRESIM.

FRESIM Study Site. Run control information, freeway geometry, calibration data, and other simulation data would have to be specified by a FRESIM input file. Calibrating the freeway model would involve specifying entry volumes, turn movements, speeds, driver aggressiveness, and other factors. Care would have to be taken to ensure that simulation behavior emulated actual freeway conditions. For a detailed description of FRESIM, see the user guide (FHWA, 1994).

The study site chosen is the northbound section of I-5 between NE Northgate Way and NE 175 Street (Figure 22). This study site is suitable because it contains three consecutive metered ramps and recurrent congestion in the afternoon. Free flow data are available before September 1994, when ramp metering for the NE 145 St. and NE 175 St. on-ramps began. Actual ramp demand obtained from free flow data rather than metered ramp volume would be helpful in calibrating the freeway model. The freeway geometry is described in terms of links and nodes, as required by FRESIM (Figure 22). For further information regarding how to input the data set, see the FRESIM User Guide (Federal Highway Administration, 1994). For debugging purposes, a minimal study site containing only the Northgate section would be used initially while incorporating the fuzzy controller into FRESIM.

The NE 145 and NE 175 on-ramps both have an HOV bypass around the ramp metering. There is also an HOV mainline lane throughout the freeway section. If FRESIM's HOV capability becomes available in time, the user will specify the



LEGEND

RM	Ramp Metered node
◇	HOV lane
#	Distance from upstream node in feet
- - -	Loop Detector
XX _#	Variables measured by the loop detector to control metered node # subscripted
○	Node location
⊕	Node number

Figure 22. FRESIM Diagram of Study Site: Part A

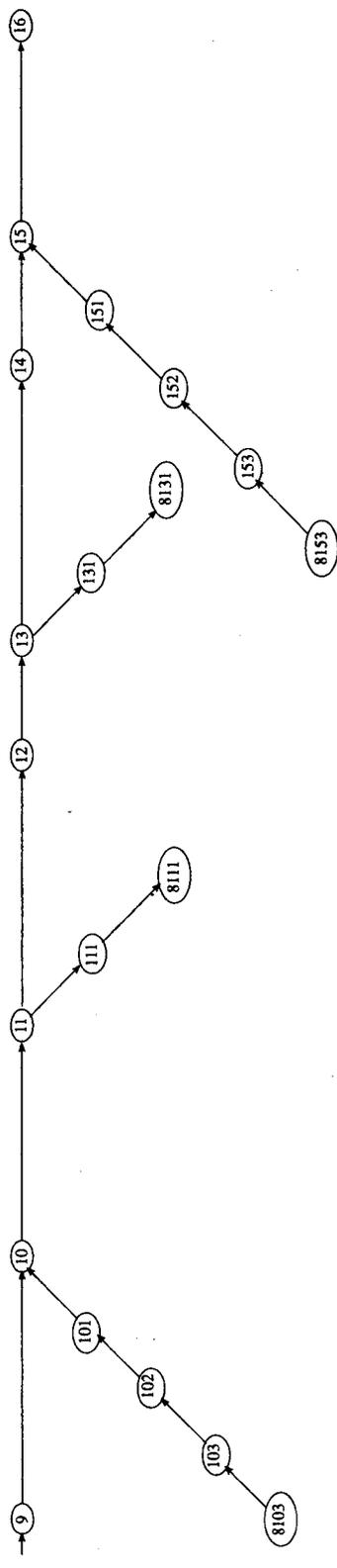
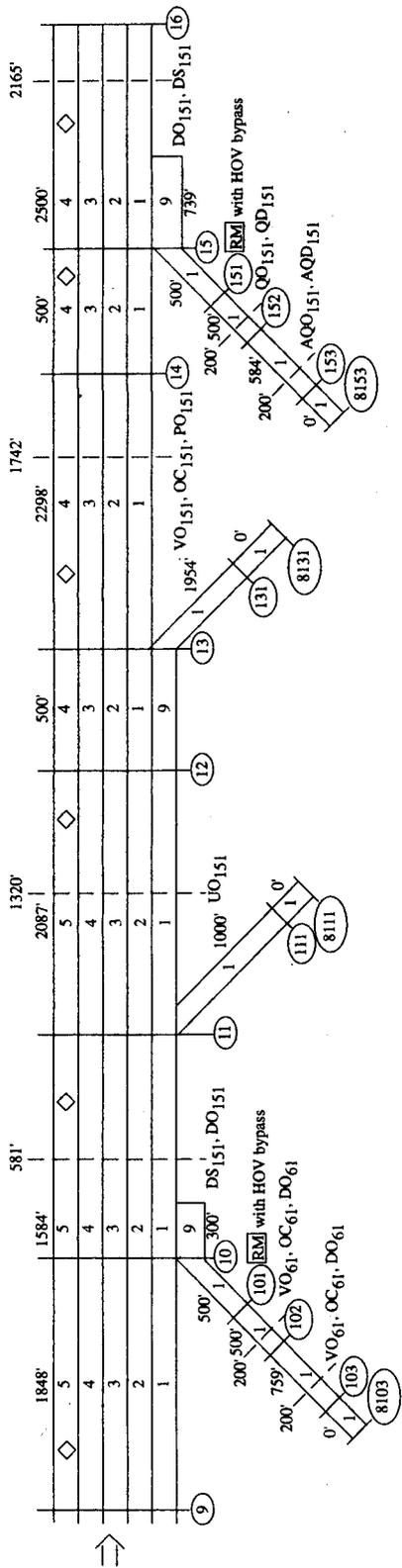


Figure 22. FRESIM Diagram of Study Site: Part B

percentage of the vehicles in each link that are HOV. These vehicles will use the HOV bypass and HOV lanes.

If the HOV capability is not available in time, FRESIM will need to be tricked into modeling the HOV lanes and bypass. The HOV lane for this study site typically has about 80 percent less volume than the other mainline lanes during peak hours. Because FRESIM averages the data across all lanes, the mainline lanes volumes would drop when averaged with the HOV lane. One way to avoid this discrepancy without making source code modifications would be to pad all entry volumes just enough to account for the disproportionate HOV volume. Attempts at directly modeling the HOV bypass geometry were unsuccessful in simulation. Alternatively, a technique to account for the HOV bypass would be to add an additional, unmetered on-ramp for HOV vehicles. Another way would be to add the HOV vehicles into the metered vehicles.

On-Line Testing Procedure

After the algorithm had been tested with a model, the next step would on-line testing. No model can perfectly replicate actual freeway behavior, so the algorithm would need fine-tuning on-line. The success of the fuzzy logic controller will hinge on the developer's abilities and the extent of on-line tuning. On-line testing is recommended in four stages: predictor testing, metering rates generated but not used, metering rates sent to a simulation field rack, and then metering sent to actual ramps.

The neural network prediction testing would not impact the freeway whatsoever, so it could be tested without using a model. The predictor can operate independently of the fuzzy logic ramp metering algorithm, and it should be tested before the complete algorithm is tested. Predictor implementation considerations are discussed in Section II.

Before actually sending metering rates to ramps, the new algorithm should be debugged by generating metering rates while the bottleneck is in use. The generated metering rates would have no impact on the freeway, but they could be compared to the

bottleneck metering rates to make sure the new algorithm was reasonable. The dummy data generator might be useful in collecting data from specified stations.

The next step could partially determine the effect of the generated metering rates by sending them to a hardware simulated field rack. Although this testing would also have no impact on the freeway, it would have an impact on the simulated loop detector inputs of the field rack. By varying the demand of the field rack, this testing would further tune the algorithm and affirm that the generated rates had a beneficial effect.

Finally, the algorithm would be ready for true on-line testing. Without doubt, an override to switch back to the bottleneck algorithm at any time should be available. Like the neural network predictor, the new algorithm should contain a flag to indicate when it is performing poorly, signaling automatic switch back to the old algorithm. Testing should start with one ramp to make sure the algorithm performed reasonably before it was applied to all ramps. The new algorithm would have to be calibrated so that it behaved similarly to the bottleneck algorithm and then gradually changed to improve performance. This technique would have a less noticeable effect on drivers, as well as ensure safety. Since traffic conditions are not uniform from day to day, the algorithm should be tested over a reasonable time span to determine its effect. Alternating week-long operation of the bottleneck algorithm with the fuzzy logic algorithm would reduce nonuniformity caused by seasonal variations. By making one change at a time, the tester could determine which modifications were beneficial, as well as verify that an improvement was due to the algorithm rather than lower demand. Although this testing procedure would be time consuming, the results should be meaningful.

Overall, both model and on-line testing would require extensive time and effort. Model calibration and integration with the controller would require at least a few months. At least one skilled programmer would be necessary for full-scale, on-line implementation. Algorithm testing and fine-tuning might take a year. However, any

ramp metering algorithm must go through this testing and tuning process to maximize and verify its efficiency.

FUZZY RAMP METERING ALGORITHM CONCLUSIONS

The 1-minute ANN prediction will be one of the inputs to a fuzzy logic ramp metering algorithm. The fuzzy logic ramp metering algorithm will determine the metering rates on the basis of both predicted and actual data. This research laid the groundwork for the fuzzy logic ramp metering concepts and algorithm.

Fuzzy logic control is well-suited to the ramp metering application for several reasons. It requires a mathematical model of the system, and it can utilize imprecise or incomplete information. These traits are important, given that the freeway is difficult to accurately model and that loop detector data are susceptible to error. The fuzzy logic rules incorporate human expertise, considering all factors simultaneously rather than making a series of adjustments.

The fuzzy logic ramp metering algorithm was designed for flexibility and robustness. For easy algorithm modification and code simplicity, adjustable parameters define the membership classes. A weight for each rule allows that rule to be emphasized or eliminated for tuning purposes. The fuzzy logic algorithm was also designed to overcome the disadvantages of Seattle's current ramp metering algorithm. The parallel rules of the controller promote robustness to faulty loop detector data. Fuzzy logic control can provide smooth transitions rather than threshold activations, as well as prevent queue formation through the use of qualitative queue inputs. Rules based on the premise of low downstream speed and high downstream occupancy provide a better indicator of bottlenecks than downstream storage rate.

Fuzzy logic algorithm testing and tuning is recommended for future research, first with a simulation model. FRESIM was found to be the most appropriate freeway simulation model to test the new ramp metering algorithm. Although modifying the FRESIM source code to incorporate the fuzzy logic ramp metering algorithm could take

a few months by an experienced programmer, it would be worth the effort. Testing on-line is complicated by the fact that traffic demand and weather characteristics vary from day to day. Because simulation testing does not have this non-uniformity, it is easier to compare and tune algorithms in simulation testing than in on-line testing.

Since no model can perfectly replicate actual freeway behavior, the algorithm will need further tuning and testing on-line. On-line testing is recommended in four steps: predictor testing, metering rates generated but not used, metering rates sent to a simulation field rack, and then metering rates sent to actual ramps. Although on-line testing is time consuming, it is necessary to maximize and verify efficiency. Algorithm testing, first through simulation and then on-line, is being continued by the authors through a TransNow 1994-1995 grant. Overall, the fuzzy logic ramp metering algorithm utilizing an ANN traffic data predictor appears quite promising.

ACKNOWLEDGMENTS

I could not have done this research project without the support of many others. Professor Deedee Meldrum has been an excellent mentor as well as a dedicated advisor. She has always been available for assistance. Her friendship has made graduate school a much richer and more enjoyable experience for me. Professor Juris Vagners has provided me with valuable guidance and support throughout my graduate career. Thanks go to Professor Mahoney for his time and encouragement. I feel fortunate to have had such an outstanding committee. Professor J. N. Hwang, Professor Nancy Nihan, Professor Dan Dailey, and Professor Bob Marks have offered helpful suggestions.

The Washington State Department of Transportation sponsored this research. The Washington State Transportation Center and Washington State Department of Transportation employees have been especially supportive. Larry Senn and Pete Briglia of WSDOT have been tremendously considerate and helpful from the conception of this research project and throughout it. Thanks go to Amy O'Brien for her editing of this report. I appreciate Les Jacobson's strong enthusiasm for this project. Thanks also go to Mark Morse, Amity Trowbridge, Dina Palas, Mike Lewey, Brian Goble, and Abel Wong for providing me with information and data from the Traffic Management Center.

The Federal Highway Administration sponsored the necessary code modifications to FRESIM to test our fuzzy logic ramp metering algorithm. I am grateful for the FRESIM programming support from Hassan Halati, Steven Chien, Jifeng Wu, and Yenlin Li at the Vigen/IDI Corporation.

REFERENCES

- R. Bretherton, "SCOOT Urban Traffic Control System-Philosophy and Evaluation," *IFAC Symposium on Control, Communications in Transportation*, 1989.
- E. Chang and K. Huarng, "Incident Detection Using Advanced Technologies," *Transportation Research Record #1399*, National Research Council, Washington, D.C., 1993.
- S. Clark, M. Dougherty, and H. Kirby, "The Use of Neural Networks and Time Series Models for Short Term Traffic Forecasting: A Comparative Study," Proceedings of PTRC 21st Summer Annual Meeting, Manchester, 1993.
- D. Dailey, "An Optimal Recursive Estimator for Detecting Traffic Anomalies using Real Time Inductance Loop Data," Transportation Research Board, National Research Council, Washington, D.C., 1993.
- M. Dougherty and H. Kirby, "Using Neural Networks to Forecast Measurement Parameters of Motorway Traffic Data," presented at the PacRim Conference, Seattle, 1993.
- Federal Highway Administration, *FRESIM User Guide*, Version 4.5, Turner-Fairbank Highway Research Center, McLean, VA, April 1994.
- A. Gupta, "Controller Design Using Fuzzy Logic (RT/Fuzzy)," Integrated Systems, Inc., 1991.
- H. Hassan and J. Torres, "FRESIM Simulation Model Enhancement and Integration," *FRESIM User Manual*, FHWA Contract DTFH60-85-C-00094, Sept. 1990.
- G. Havinoviski, "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.
- K. Henry and O. Mehryar, "Six-Year FLOW Evaluation," Washington State Department of Transportation, District 1, Jan. 1989.
- J. Hua and A. Faghri, "Dynamic Traffic Pattern Classification Using Artificial Neural Networks," *Transportation Research Board #1399*, National Research Council, Washington, D.C., 1993.
- D. Hush and B. Horne, "Progress in Supervised Neural Networks: What's New Since Lippman?" *IEEE Spectrum Signal Processing Magazine*, Jan. 1993.
- J. N. Hwang, H. Li, M. Maechler, D. Martin, J. Schimert. "Projection Pursuit Learning Network for Regression," *Engineering Applic. Artif. Intell.*, Pergamon Press Ltd., 1992.
- M. Iwasaki, "Empirical Analysis of Congested Traffic Flow Characteristics and Free Speed Affected by Geometric Factors on an Intercity Expressway," *Transportation Research Record 1320*, National Research Council, Washington D.C., 1991, p. 244.

- L. Jacobson, K. Henry, and O. Mehyar, "Real-Time Metering Algorithm for Centralized Control," *Transportation Research Board* #1232, Washington State Department of Transportation, 1988.
- B. Kosko, *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*, Prentice-Hall, New Englewood Cliffs, New Jersey, pp. 318 and 386, 1992.
- S. Kung, *Digital Neural Networks*, Prentice Hall, Englewood Cliff, New Jersey, 1993.
- Y. Lin and T. Lee, "Modeling for Fuzzy Logic Control of Deformable Manipulators," *Proceedings of the American Controls Conference*, San Francisco, June 1993.
- Masher, Ross, Wong, Tuan, Zeidler, and Petracek, *Guidelines for Design and Operation of Ramp Control Systems*, *Stanford Research Institute*, Menlo Park, CA, 1975, p. II_18-II_24.
- A. May, "Washington Workshop: A Basic Course in FREQ10," prepared for Washington State Department of Transportation, Institute of Transportation Studies, University of California, Berkley, Oct. 1993.
- Mead, Fisher, Jones, Bisset, and Lee, "Application of Adaptive and Neural Network Computation Techniques to Traffic Volume and Classification Monitoring," preprint from Transportation Research Board, National Research Council, Washington, D.C., 1994.
- N. Nihan, "Freeway Congestion Prediction," draft technical report, Washington Transportation Center, National Technical Information Service, March 1995.
- N. Nihan and D. Berg, "Predictive Algorithm Improvements for a Real-Time Ramp Control System," Washington State Transportation Center, National Technical Information Service, Sept. 1991.
- N. Nihan and J. Zhu, "Short-Term Forecasts of Freeway Traffic Volumes and Lane Occupancies," Phase I, Vol. IV, Washington State Transportation Center, National Technical Information Service, Nov. 1992.
- J. Robinson and M. Doctor, "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.
- Rumelhart, McClellan, and the PDP Research Group, *Parallel Distributed Processing*, Vol. 1, MIT, pp. 354-362, 1986.
- S. Washburn, "Development of a Predictive Freeway Congestion Algorithm Using Statistical Pattern Recognition Techniques," Master's Thesis, Department of Civil Engineering, University of Washington, 1993.
- C. Wei and P. Schonfeld, "An Artificial Neural Network Approach for Estimating Multiperiod Travel Times in Transportation Networks," presented at Annual Meeting of the Transportation Research Board, National Research Council, Washington, D.C., 1993.

- D. Wicks and E. Lieberman. "Development and Testing of INTRAS, a Microscopic Freeway Model," Vol. 1, *Program Design, Parameters Calibration, and Freeway Dynamics Component Development*. Final Report, FHWA/RD/80/106. FHWA, U.S. Department of Transportation, Oct. 1980.
- L. Wiederholt, P. Okenieff, and J. Wang, "Incident Detection and Artificial Neural Networks," Large Urban Systems, Proceedings of the Advance Traffic Management Center, St. Petersburg, Florida, Oct. 1993.
- Williams and Zisper, "A Learning Algorithm for Continually Running Fully Recurrent Neural Network," *Neural Computation*, Vol 1, MIT, pp. 270-280, 1989.
- S. Yasunobu and S. Miyamoto, "Automatic Train Operation by Predictive Fuzzy Control," *Industrial Applications of Fuzzy Control*, editor M. Sugeno, pp. 1-18, North-Holland, Amsterdam, 1985.
- L. Zadeh, "Fuzzy Sets," *Information and Control* 8, pp. 338-353, 1965.
- H. Zhang and S. Ritchie, "Macroscopic Modeling of Freeway Using an Artificial Neural Network," presented at Annual Meeting of the Transportation Research Board, National Research Council, Washington, D.C., 1993.

APPENDIX A
NEURAL NETWORK PREDICTOR CODE

APPENDIX A: NEURAL NETWORK PREDICTOR CODE

/* Predict4.c trains and tests a MLP neural network to predict freeway volume and occupancy 1-minute in advance for southbound I-5 just before NE 205th St. on-ramp. Data is historical loop detector data from Seattle I-5 mainline near NE 205th St. This neural network has 4*D inputs which are the past D occupancy and volume values for stations s61 and s60. The neural network has two outputs which are the predicted volume and occupancy for s61 at the next sample. */

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#define D 10          /* D=past number of delays used as inputs to NN */
#define S 4000       /* S=number of training iterations */
#define H 28         /* H=number of hidden neurons */
#define MAXR 200
#define MAXIN 20
#define MAXH 50
#define RAND_MAX 0x7fffffff

/* tdat3 returns the training and testing input/teacher pairs and their length */
void tdat3(float (*x1)[MAXIN], float (*t1)[2], int *n1, float (*x2)[MAXIN],
          float (*t2)[2], int *n2)
{
    /* each of the 4 data files below contains 1-minute volume and occupancy
       between 6:30 and 9:00 a.m. for a particular station */
    FILE *s6128;
    FILE *s6101;
    FILE *s6028;
    FILE *s6001;
    FILE *input;
    FILE *result;
    char bogus[9];
    int k, i, j;
    int IN=D*4;      /* # of inputs to ANN=4 if 2 stations used for input data */
    float m1=12.94; /* mean of occupancy output in training/testing sets */
    float m2=94.3;  /* mean of volume output in training/testing sets */
    float s1=1.99;  /* standard deviation of volume in data sets */
    float s2=12.0;  /* standard deviation of occupancy in data sets */
    float s60_28_1[MAXR], s60_01_1[MAXR], s61_28_1[MAXR], s61_01_1[MAXR];
    float s60_28_2[MAXR], s60_01_2[MAXR], s61_28_2[MAXR], s61_01_2[MAXR];

    (*n1)=0;
    if ((s6128=fopen("s61-28.txt", "r"))== NULL) {
        printf("Can't open input file s61-28.txt\n");
        exit(0);
    }
    while(fscanf(s6128, "%s %f %f %s", &bogus, &s61_28_1[*n1], &s61_28_2[*n1],
                &bogus) != EOF)
        (*n1)++;
    fclose(s6128);
```

```

(*n2)=0;
if ((s6101=fopen("s61-01.txt", "r")) == NULL) {
    printf("Can't open input file s61-01.txt\n");
    exit(0);
}
while(fscanf(s6101, "%s %f %f %s", &bogus, &s61_01_1[*n2], &s61_01_2[*n2],
    &bogus) != EOF)
    (*n2)++;
fclose(s6101);

(*n1)=0;
if ((s6028=fopen("s60-28.txt", "r")) == NULL) {
    printf("Can't open input file s60-28.txt\n");
    exit(0);
}
while(fscanf(s6028, "%s %f %f %s", &bogus, &s60_28_1[*n1], &s60_28_2[*n1],
    &bogus) != EOF)
    (*n1)++;
fclose(s6028);

(*n2)=0;
if ((s6001=fopen("s60-01.txt", "r")) == NULL) {
    printf("Can't open input file s60-01.txt\n");
    exit(0);
}
while(fscanf(s6001, "%s %f %f %s", bogus, &s60_01_1[*n2], &s60_01_2[*n2],
    bogus) != EOF)
    (*n2)++;
fclose(s6001);

/* calc. mean */
for (i=0; i<(*n1); i++) {
    m1=m1+s61_28_1[i]+s60_28_1[i];
    m2=m2+s61_28_2[i]+s60_28_2[i];
}
for (i=0; i<(*n2); i++) {
    m1=m1+s61_01_1[i]+s60_01_1[i];
    m2=m2+s61_01_2[i]+s60_01_2[i];
}
m1=m1/(float) (2*( *n1));
m2=m2/(float) (2*( *n1));
*/

if ((result=fopen("result6.m", "w"))== NULL) {
    printf("Can't open input file result6.m\n");
    exit(0);
}
printf("m1= %5.4f\n", m1);
printf("m2= %5.4f\n", m2);
fprintf(result, "m1= %10.6f;\n", m1);
fprintf(result, "m2= %10.6f;\n", m2);

/* calc. std. dev. */
for (i=0; i<(*n1); i++) {
    s1=s1+(float) (pow((double) (s61_28_1[i]-m1),2.0)+

```

```

                pow((double) (s60_28_1[i]-m1),2.0));
s2=s2+(float) (pow((double) (s61_28_2[i]-m2),2.0)+
                pow((double) (s60_28_2[i]-m2),2.0));
    }
    for (i=0; i<(*n2); i++) {
        s1=s1+(float) (pow((double) (s61_01_1[i]-m1),2.0)+
                    pow((double) (s60_01_1[i]-m1),2.0));
        s2=s2+(float) (pow((double) (s61_01_2[i]-m2),2.0)+
                    pow((double) (s60_01_2[i]-m2),2.0));
    }
s1=(float) sqrt((double) (s1/(float) (2*( *n1)+2*( *n2))));
s2=(float) sqrt((double) (s2/(float) (2*( *n1)+2*( *n2))));
*/

printf("s1= %5.4f;\n", s1);
printf("s2= %5.4f;\n", s2);
fprintf(result, "s1= %10.6f;\n", s1);
fprintf(result, "s2= %10.6f;\n", s2);
fclose(result);

/* normalize data using mean and std */
for (i=0; i<(*n1); i++) {
    s61_28_1[i]=(s61_28_1[i]-m1)/(4*s1)+0.5;
    s61_28_2[i]=(s61_28_2[i]-m2)/(4*s2)+0.5;
    s60_28_1[i]=(s60_28_1[i]-m1)/(4*s1)+0.5;
    s60_28_2[i]=(s60_28_2[i]-m2)/(4*s2)+0.5;
}
for (i=0; i<(*n2); i++) {
    s61_01_1[i]=(s61_01_1[i]-m1)/(4*s1)+0.5;
    s61_01_2[i]=(s61_01_2[i]-m2)/(4*s2)+0.5;
    s60_01_1[i]=(s60_01_1[i]-m1)/(4*s1)+0.5;
    s60_01_2[i]=(s60_01_2[i]-m2)/(4*s2)+0.5;
}

/* set up teacher for training and testing */
k=0;
for (i=D; i<(*n1); i++) {
    t1[k][0]=s61_28_2[i];
    t1[k][1]=s61_28_1[i];
    k++;
}
k=0;
for (i=D; i<(*n2); i++) {
    t2[k][0]=s61_01_2[i];
    t2[k][1]=s61_01_1[i];
    k++;
}

/* set up training and testing input */
for (i=1; i<(D+1); i++) {
    j=D-i;
    for (k=0; k<(( *n1)-D); k++) {
        x1[k][2*i-2]=s61_28_2[j];
        x1[k][2*i-1]=s61_28_1[j];
        x1[k][2*i+D*2-2]=s60_28_2[j];
        x1[k][2*i+D*2-1]=s60_28_1[j];
    }
}

```

```

        j++;
    }
    j=D-i;
    for (k=0; k<((n2)-D); k++) {
        x2[k][2*i-2]=s61_01_2[j];
        x2[k][2*i-1]=s61_01_1[j];
        x2[k][2*i+D*2-2]=s60_01_2[j];
        x2[k][2*i+D*2-1]=s60_01_1[j];
        j++;
    }
}
*n1=(n1)-D;
*n2=(n2)-D;
if ((input=fopen("input.m", "w")) == NULL) {
    printf("Output file input.m cannot be opened\n");
    exit(0);
}
fprintf(input, "x1=[");
for (k=0; k<6; k++)
    for (i=0; i<IN; i=i+5)
        fprintf(input, "%7.4f, %7.4f, %7.4f, %7.4f, %7.4f,\n",
            x1[k][i], x1[k][i+1], x1[k][i+2], x1[k][i+3], x1[k][i+4]);
fprintf(input, "];\n");
fclose(input);
}

/* sigmoid activation function */
float sig(float n)
{
    float g;
    g=(float) (1/ (1+exp( (double) (-n) )) );
    return g;
}

/* teach network */
void train3(float (*x1)[MAXIN], float (*t1)[2], int *n1, float *theta1,
    float *theta2, float (*w1)[MAXIN], float (*w2)[MAXH])
{
    int IN=D*4; /* IN=number of inputs to ANN */
    float LR=2.0; /* LR=learning rate */
    int k, j, m, sweep, pair, century;
    float MSE[S/100+1][4]; /* holds sweep, testing mean squared error, LR */
    float error; /* used to calculated mean squared error */
    float a[H], out[2], d1[H], d2[2], output[(n1)][2], sum[2];
    FILE *result;

    /* save LR, H, S, MSE in m-file result6.m */
    if ((result=fopen("result6.m", "a")) == NULL) {
        printf("Output file result6.m cannot be opened\n");
        exit(0);
    }
    fprintf(result, "LR= %5.2f;\n", LR);
    fprintf(result, "H= %d;\n", H);
    fprintf(result, "S= %d;\n", S);
}

```

```

fprintf(result, "D= %d;\n", D);
fprintf(result, "MSE=[");

/* initialize theta1, theta2, w1, w2 to random value between -.5 and .5 */
for (k=0; k<H; k++) {
    /* theta1[k]=(((float) rand())/((float) RAND_MAX))-0.5; */
    for (m=0; m<(IN); m++)
        /* w1[k][m]=(((float) rand())/((float) RAND_MAX))-0.5; */
        for (m=0; m<2; m++)
            /* w2[m][k]=(((float) rand())/((float) RAND_MAX))-0.5; */
}
for (k=0; k<2; k++)
    /* theta2[k]=(((float) rand())/((float) RAND_MAX))-0.5; */

for (sweep=0; sweep<(S); sweep++) {
    for (pair=0; pair<*n1; pair++) {
        if (sweep==1000)
            LR=0.9;

        /* compute output of layer 1 */
        for (k=0; k<H; k++)
            /* init. a=zeros(H,1) */
            a[k]=0.0;
        for (k=0; k<H; k++) {
            for (j=0; j<(IN); j++){
                a[k]=a[k]+w1[k][j]*x1[pair][j];
            }
            a[k]=sig(a[k]+theta1[k]);
        }

        /*compute output of layer 2 */
        for (k=0; k<2; k++)
            out[k]=0.0;
        for (k=0; k<2; k++) {
            for (j=0; j<H; j++)
                out[k]=out[k]+w2[k][j]*a[j];
            out[k]=sig(out[k]+theta2[k]);
        }
        for (k=0; k<2; k++)
            output[pair][k]=out[k];
        /*compute error of output layer */
        for (k=0; k<2; k++)
            d2[k]=(out[k]*(1-out[k]))*(t1[pair][k]-out[k]);
        /* compute error of hidden layer */
        for (k=0; k<H; k++)
            d1[k]=(a[k]*(1-a[k]))*(d2[0]*w2[0][k]+d2[1]*w2[1][k]);

        /* update weights from hidden layer to output layer */
        for (k=0; k<2; k++) {
            for (m=0; m<H; m++) {
                w2[k][m]=w2[k][m]+LR*d2[k]*a[m];
                /* printf("w2[k][m]=%7.4f\n", w2[k][m]); */
            }
            theta2[k]=theta2[k]+LR*d2[k];
            /* printf("theta2[k]=%7.4f\n", theta2[k]); */
        }
    }
}

```

```

/* update weights from input to hidden layer */
for (k=0; k<H; k++) {
    for (m=0; m<IN; m++) {
        w1[k][m]=w1[k][m]+LR*d1[k]*x1[pair][m];
/*         printf("w1[k][m]=%7.4f\n", w1[k][m]); */
    }
    theta1[k]=theta1[k]+LR*d1[k];
/*     printf("theta1[k]=%7.4f\n", theta1[k]); */
}
}
/* save results every 100th sweep */
if (((sweep+1) % 100)==0 || (sweep==0)) {

    century=(int) ((sweep+1)/100);
    printf("Century= %d\n", century);
    printf("Sweep= %d\n", sweep);
    MSE[century][0]=sweep;
    MSE[century][3]=LR;
    for (m=0; m<2; m++)
        sum[m]=0.0; /* init. sum=zeros(2,1) */
    for (k=0; k<(*n1); k++) {
        for (m=0; m<2; m++) {
            error=t1[k][m]-output[k][m];
            sum[m]=sum[m]+(float) pow((double) error,
                (double) 2.0);
        }
    }
    if (sweep==99)
        printf("sum=%7.4f, %7.4f\n", sum[0], sum[1]);
    MSE[century][1]=(float) sqrt((double) sum[0]);
    MSE[century][2]=(float) sqrt((double) sum[1]);
    fprintf(result, "%7.0f, %7.3f, %7.3f\n",
        MSE[century][0], MSE[century][1], MSE[century][2],
        MSE[century][3]);
    printf("MSE[1], MSE[2]= %7.3f, %7.3f\n", MSE[century][1],
        MSE[century][2], MSE[century][3]);
}
}
fprintf(result, ";\n");
fclose(result);
}

/* test network */
void test3(float (*x1)[MAXIN], float (*t1)[2], int *n1, float (*x2)[MAXIN],
    float (*t2)[2], int *n2, float (*w1)[MAXIN],
    float (*w2)[MAXH], float *theta1, float *theta2)
{
    int IN=D*4;
    int pair, k, j;
    float a[H], out[2], output1[*n1][2], output2[*n2][2], error1[*n1][2];
    float sum1[2], sum2[2], MSE1[2], MSE2[2], error2[*n2][2];
    FILE *result;
    for (k=0; k<2; k++) /* init. sum1 */
        sum1[k]=0.0;
}

```

```

for (k=0; k<2; k++)
    sum2[k]=0.0;    /* init. sum2 */

for (pair=0; pair<((n1)+(n2)); pair++) {
    /* compute output of layer 1 */
    for (k=0; k<H; k++)    /* init. a=zeros(H,1) */
        a[k]=0.0;
    for (k=0; k<H; k++) {
        for (j=0; j<IN; j++) {
            if (pair<(n1))
                a[k]=a[k]+w1[k][j]*x1[pair][j];
            else
                a[k]=a[k]+w1[k][j]*x2[pair-(n1)][j];
        }
        a[k]=sig(a[k]+theta1[k]);
    }
    /* compute output of layer 2 */
    for (k=0; k<2; k++)
        out[k]=0.0;    /* init. out=zeros(2,1) */
    for (k=0; k<2; k++) {
        for (j=0; j<H; j++)
            out[k]=out[k]+w2[k][j]*a[j];
        out[k]=sig(out[k]+theta2[k]);
    }
    if (pair<(n1))
        for (k=0; k<2; k++) {
            output1[pair][k]=out[k];
            error1[pair][k]=t1[pair][k]-output1[pair][k];
            sum1[k]=sum1[k]+(float) pow((double) error1[pair][k],
                (double) 2.0);
        }
    else
        for (k=0; k<2; k++) {
            output2[pair-(n1)][k]=out[k];
            error2[pair-(n1)][k]=t2[pair-(n1)][k]-
                output2[pair-(n1)][k];
            sum2[k]=sum2[k]+(float) pow((double)
                error2[pair-(n1)][k],
                (double) 2.0);
        }
    }
printf("sum1=%7.4f, %7.4f\n", sum1[0], sum1[1]);
for (k=0; k<2; k++) {
    MSE1[k]=(float) sqrt((double) sum1[k]);
    MSE2[k]=(float) sqrt((double) sum2[k]);
}
/* save t1, output1, MSE1, error1, t2, output2, MSE2, error2 in m-file */
if ((result=fopen("result6.m", "a")) == NULL) {
    printf("Output file result6.m cannot be opened\n");
    exit(0);
}
fprintf(result, "MSE1= [%7.4f, %7.4f];\n", MSE1[0], MSE1[1]);
fprintf(result, "MSE2=[%7.4f, %7.4f];\n", MSE2[0], MSE2[1]);
fprintf(result, "t1=[");    /* t1 */

```

```

for (k=0; k<(*n1); k++)
    fprintf(result, "%7.4f, %7.4f;\n", t1[k][0], t1[k][1]);
fprintf(result, "];\n");
fprintf(result, "output1=[");          /* output1 */
for (k=0; k<(*n1); k++)
    fprintf(result, "%7.4f, %7.4f;\n", output1[k][0], output1[k][1]);
fprintf(result, "];\n");
fprintf(result, "t2=[");              /* t2 */
for (k=0; k<(*n2); k++)
    fprintf(result, "%7.4f, %7.4f;\n", t2[k][0], t2[k][1]);
fprintf(result, "];\n");
fprintf(result, "output2=[");        /* output2 */
for (k=0; k<(*n2); k++)
    fprintf(result, "%7.4f, %7.4f;\n", output2[k][0], output2[k][1]);
fprintf(result, "];\n");
fclose(result);
}

/* calls set-up, training, and testing functions */
void main()
{
    /*      n1=length training data set
            n2=length testing data set */
    x1=inputs to ANN for training set
    x2=inputs to ANN for testing set
    t1=actual 1-minute output for training set
    t2=actual 1-minute output for testing set
    w1=weights to hidden layer from input layer
    w2=weights to output layer from hidden layer
    theta1=offset for hidden layer neurons
    theta2=offset for output layer neurons */

    int n1, n2;
    float x1[MAXR][MAXIN], t1[MAXR][2], x2[MAXR][MAXIN], t2[MAXR][2];
    float w1[MAXH][MAXIN], w2[2][MAXH], theta1[MAXH], theta2[2];

    tdat3(x1, t1, &n1, x2, t2, &n2);

    train3(x1, t1, &n1, theta1, theta2, w1, w2);

    test3(x1, t1, &n1, x2, t2, &n2, w1, w2, theta1, theta2);
}

```

APPENDIX B
ANN LONG TERM PREDICTIONS

APPENDIX B: ANN LONG TERM PREDICTIONS

This appendix encompasses results for 2- and 5-minute predictions. The discussion of why various techniques were unsuccessful and how to remedy them should aid in future long-term prediction attempts.

TWO-MINUTE PREDICTIONS

The first method tried used 1-minute data with the same inputs to the MLP as in Figure 9, but with the training output being the data sample 2 minutes in advance rather than 1. This ANN configuration forecasted data 2 minutes in advance. Despite numerous trials with different architectures and learning techniques, the 2-minute predictors did not test well on new data. Some of the parameters that were varied included the number of hidden layers, the length of the tapped delay line (defined as the number of past samples used as inputs), the learning rate, and the number of sweeps. For the case of a tapped delay line with 10 taps and a 29 neuron hidden layer, the ANN learned well (Figure B.1); however, the testing data were *not* predicted well with this method (Figure B.2).

Because a vehicle may travel over 2 miles during a 2-minute forecasted period, a successful predictor may require input data from the next four adjacent upstream stations (with an average spacing of a half mile between loop detectors). Therefore, the next attempt at predicting 2 minutes in advance used an MLP architecture similar to Figure 9, except it included the input data from the next two adjacent stations and the station to be predicted, for a total of three input stations rather than two. Although this ANN learned the training set, it could not generalize to the testing set. The ANN output was frequently stuck at 1 on the testing set. Network saturation may have been the cause for output stuck at 1. Another problem with this ANN was that the greater

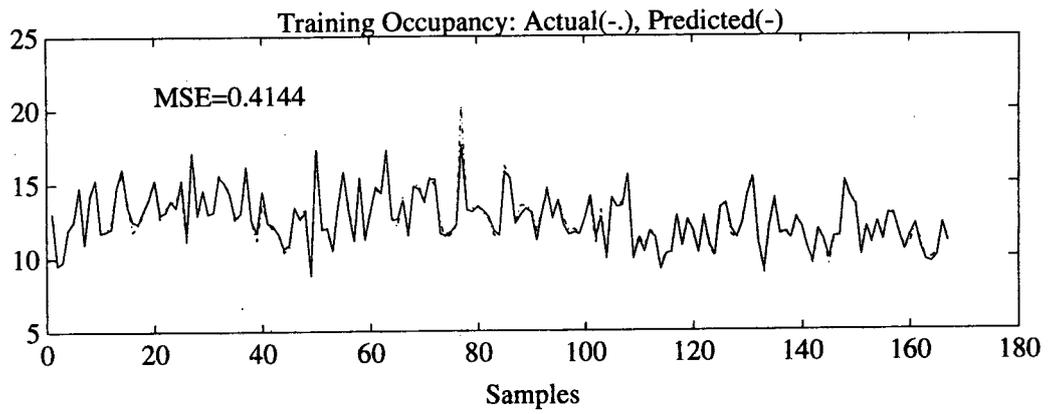
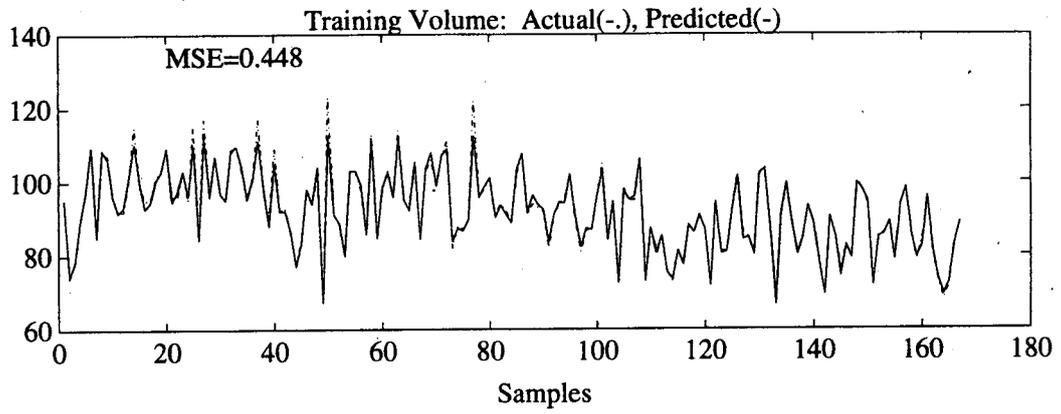


Figure B.1. Training Set for 2-minute Predictor

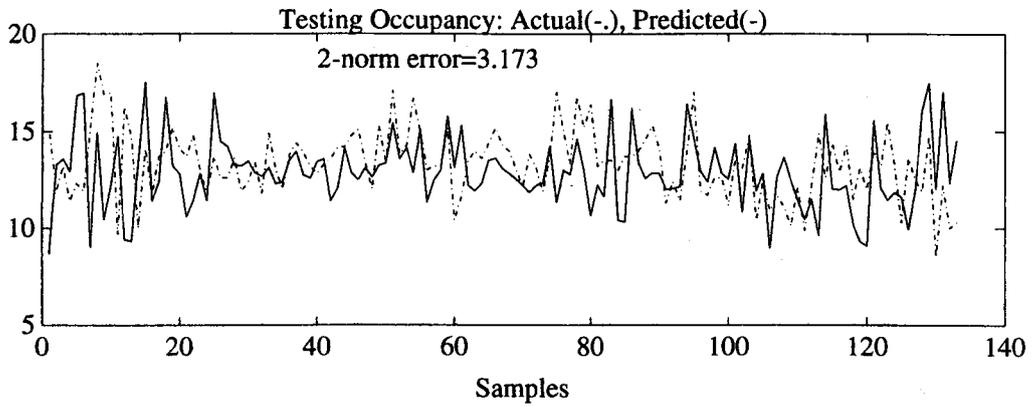
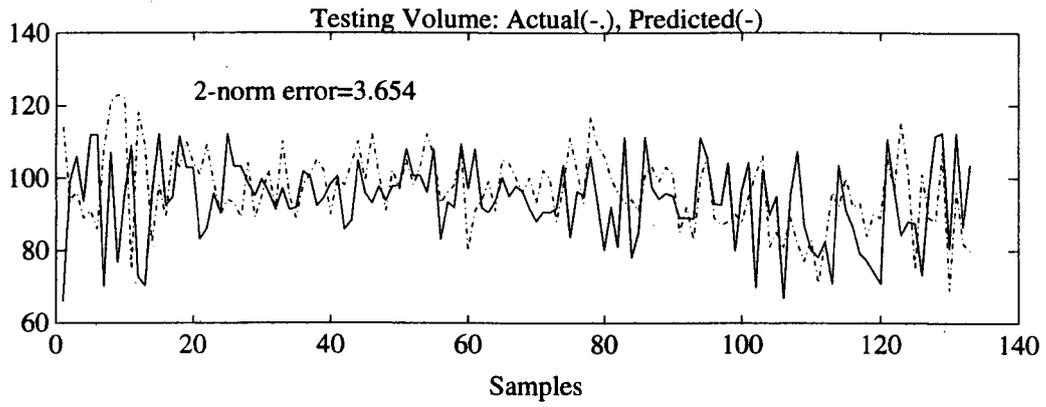


Figure B.2. Testing Set for 2-minute Predictor

complexity made training more cumbersome. Network saturation seems to be more likely with a larger ANN. In addition, a technique may be necessary to keep the ANN minimal. Because the generalization abilities of the three-station input MLP were worse than the two-station input MLP, 2-minute prediction attempts using additional upstream station inputs were not pursued further.

FIVE-MINUTE PREDICTIONS

The 5-minute prediction methods used data sampled every 5 minutes. As with 1-minute data, the volume was an accumulation, and the occupancy was an average over the sampled period. The 5-minute data were smoother than the 1-minute data, with more noticeable general trends. However, each day's data looked different from other days, making generalization challenging. In addition, the greater dynamic range of the 5-minute data required more scaling to fit inside the 0 to 1 range. The dynamic range of the 5-minute data made both training and generalization more difficult. The 5-minute predictors used the same architecture as the 1-minute predictor shown in Figure 9, with a tapped delay line of past samples for the input, and the next sample as the output. Because the ANN used 5-minute data instead of 1-minute data, it predicted 5 minutes in advance.

The ANN trained over several days instead of one in order to have enough training examples. For 1-minute data, a one day 6:00 to 9:30 AM period provided 180 training examples. For 5-minute data, 4 days during 6:00 to 10:00 AM provided 192 training examples. Thus, the training set for such an ANN should be a minimum of 4 days to provide an adequate number of training examples. To create a training set over several days, each day had to be given initial start up conditions. The length of the tapped delay line was the number of initial conditions each day required, which further reduced the number of training examples.

Figures B.3 and B.4 show a 6-day training and a 6-day testing set, respectively. Notice the high occupancy around 150 samples on the training set and around 70, 120, and 260 samples on the testing set. These high occupancies may represent either bottleneck congestion or data errors. The causes of these irregular data points were not certain, but they most likely indicated bottleneck conditions. One way to find out what events irregular points represented would be to watch the traffic on closed-circuit television and later compare remarks to plots of the data. This procedure would be time consuming, as several days of morning data would be needed. Another approach would be to compare loop detector data with that from the AUTOSCOPE (developed by the University of Minnesota Center for Transportation Studies and manufactured by Econolite). The AUTOSCOPE visually counts vehicles, so it could indicate whether the irregular points were caused by loop detector errors.

Figures B.5 and B.6 show the testing and training data for an MLP with two input stations, training over four days, and testing over two days. This example had a tapped delay line of length 4, a 25-neuron hidden layer, and 8000 training iterations with an adjustable learning rate. Using a momentum term, defined as an additional weight change in the direction of the previous weight change, degraded generalization, so no momentum was used in this example. The MLP had more difficulty learning the 5-minute training data than it did learning the 1-minute data, possibly because of the greater dynamic range. The ANN was trained until the testing MSE reached a minimum, but it could learn the training data better at the price of less generalization. The generalization to new data shown in Figure B.6 was poor.

The ANN may have had trouble generalizing because the daily pattern varied considerably from one day to the next. If this assumption is correct, the ANN may have generalized better if it had been given more training examples. The next effort used six days for both training and testing. The MLP used a tapped delay line of length

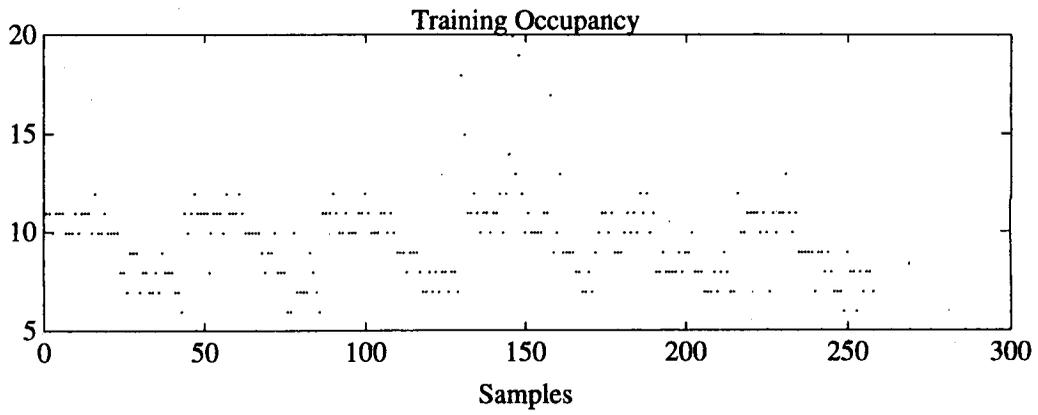
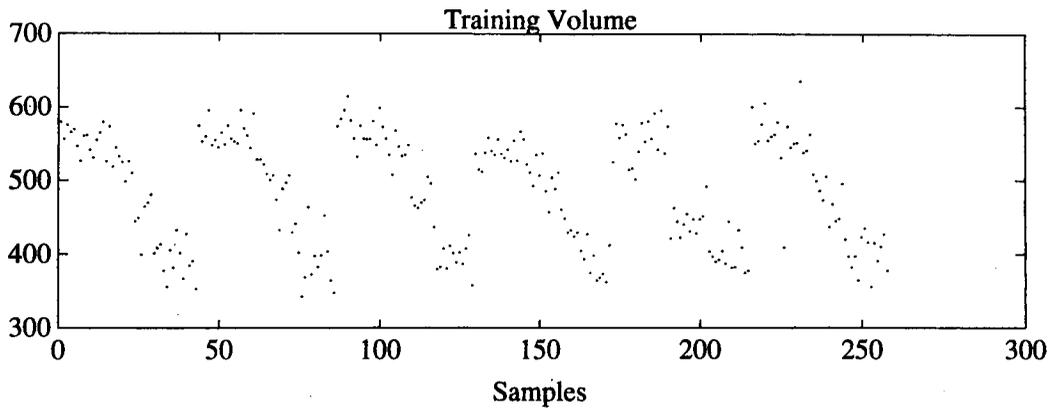


Figure B.3. Six Days of 5-Minute Training Data

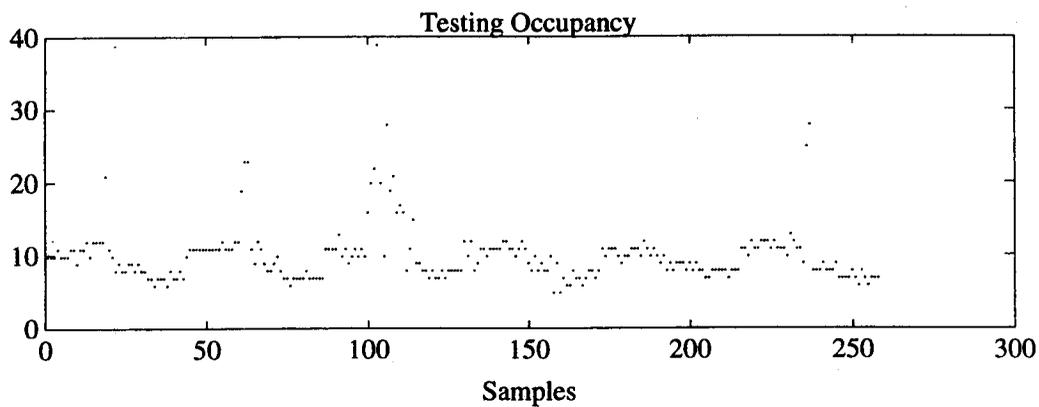
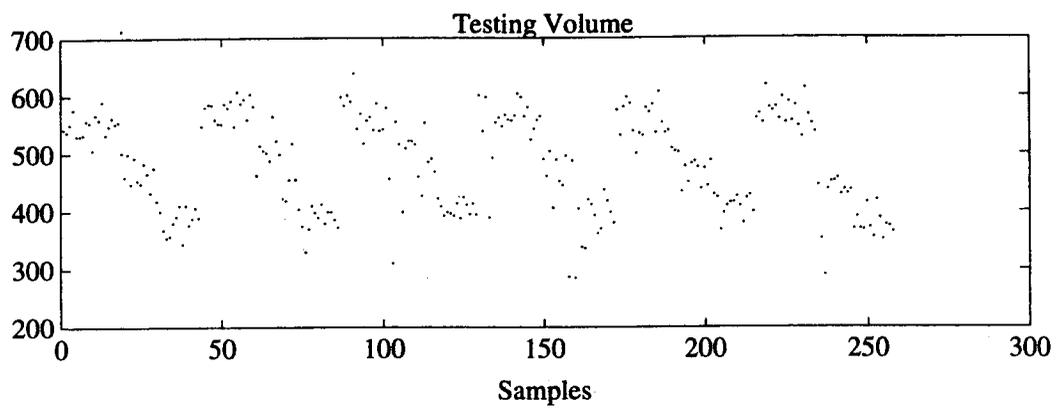


Figure B.4. Six Days of 5-Minute Testing Data

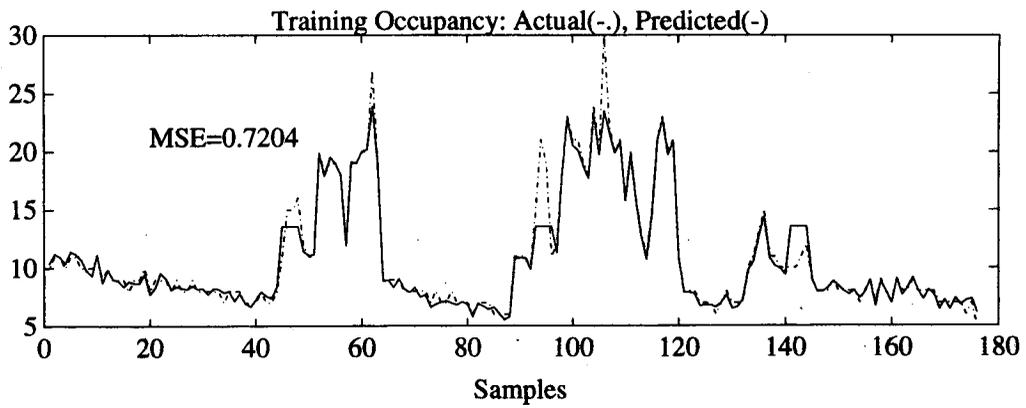
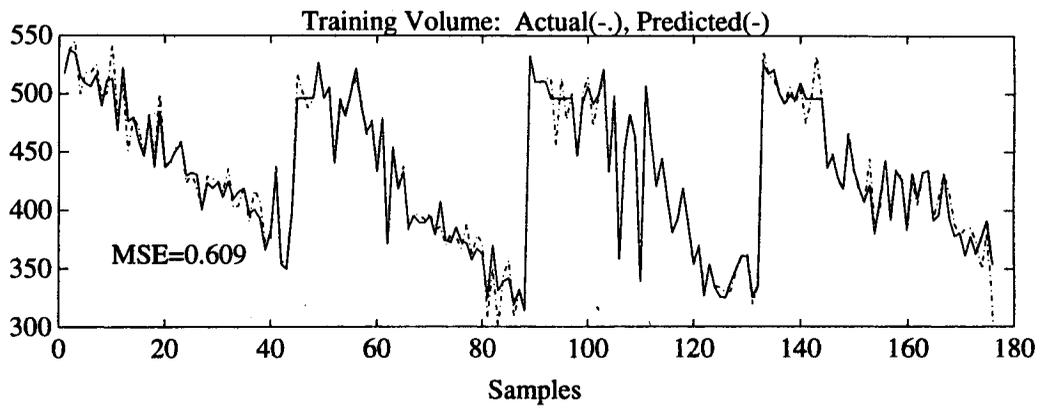


Figure B.5. Four-Day Training Set for 5-minute Predictor

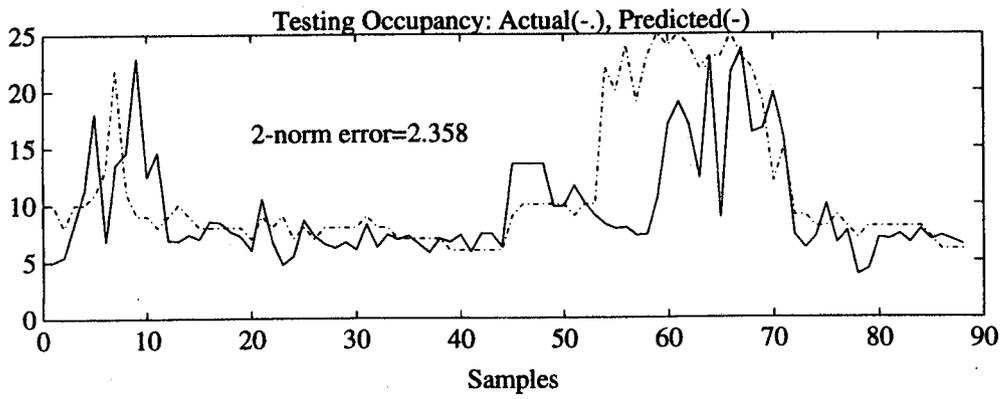
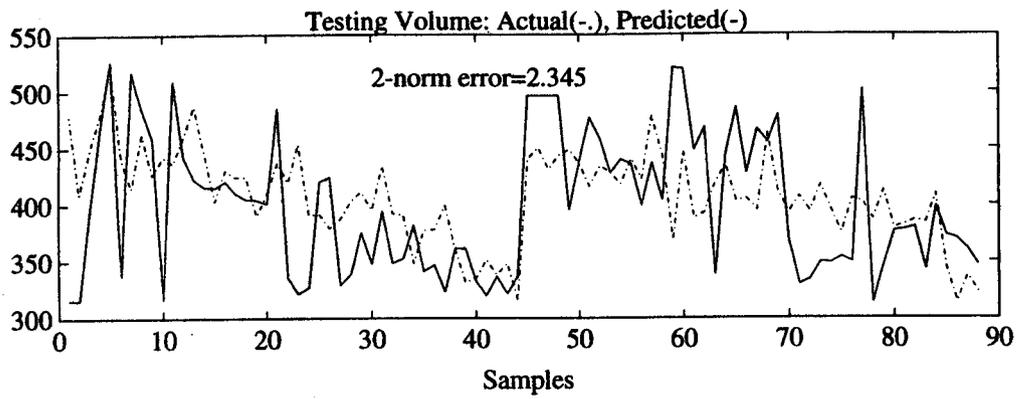


Figure B.6. Two Day Testing Set for 5-minute Predictor

5, a 25-neuron hidden layer, 8000 training iterations, and an adjustable learning rate. For this case, a momentum rate of 0.3 speeded learning without affecting generalization ability. The MLP mimicked the training data (Figure B.7), but the testing output was often stuck at 1.0 (Figure B.8). When the training error has an offset rather than hovering around zero, this indicates that the MLP is not truly learning the training data. Since the training error was usually above zero for this example, learning was poor. The mean training error was positive, but the mean testing error is negative. To overcome the training and dynamic range difficulties, another learning technique or data preprocessing scheme may be necessary.

Network saturation may have contributed to the training difficulty with the 5-minute data. Saturation occurs when a weight freezes at a value because *the neuron* has an output of 0 or 1. The factor

$$output * (1-output)$$

in the error gradient calculation sets the gradient to zero. To avoid this problem, an offset of 0.1 was added to this factor. This attempt failed because the sigmoid offset degraded the training performance for the 5-minute predictor. The next attempt added a random offset between 0 and 1 to this factor whenever the neuron output was within .001 of 0 and 1. This random offset degraded performance and decreased learning speed, so attempts using offsets to avoid saturation were not pursued further.

Because a vehicle can travel over 5 miles in 5 minutes, including further upstream data stations as inputs seems logical. However, the complexity of the neural network with 10 input stations would make training more cumbersome. A neural network of this size would have around 6000 input layer weights. Also, the random inputs over the 5 mile distance that a vehicle can travel during the forecasted period would reduce the prediction accuracy. Examination of four adjacent stations indicated that peaks occur simultaneously in 5-minute data. Consequently, nearby upstream

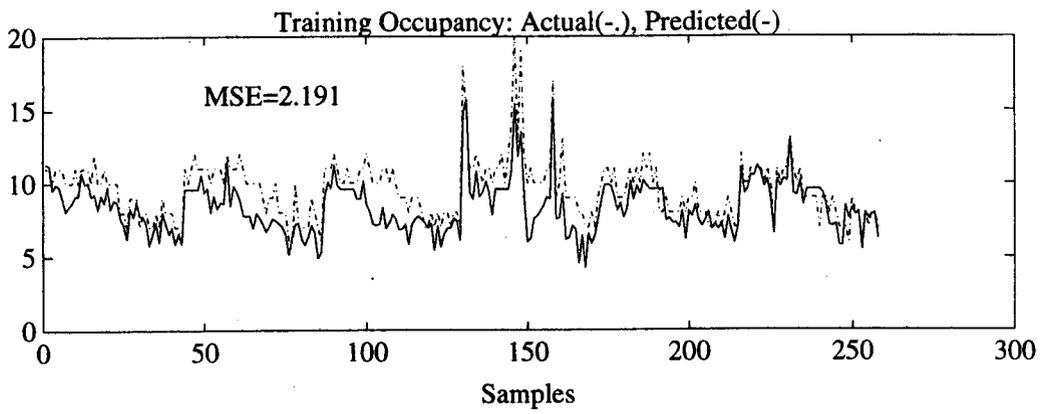
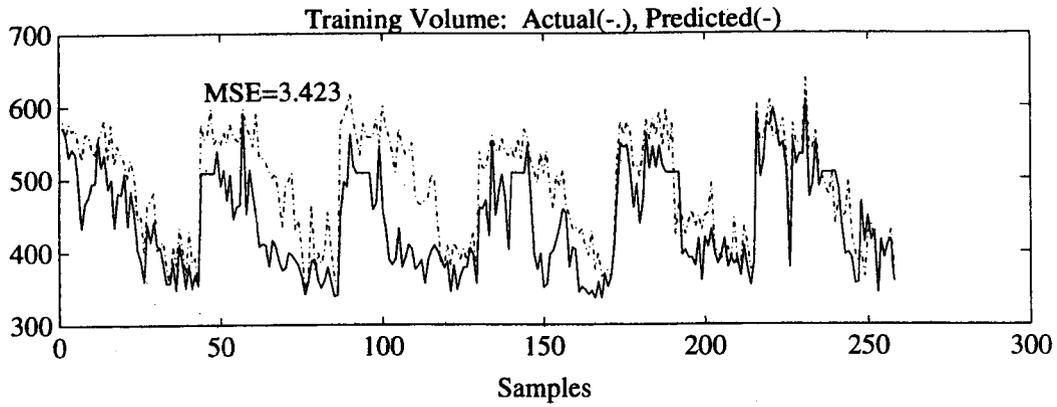


Figure B.7. Six-Day Training Set for 5-minute Predictor

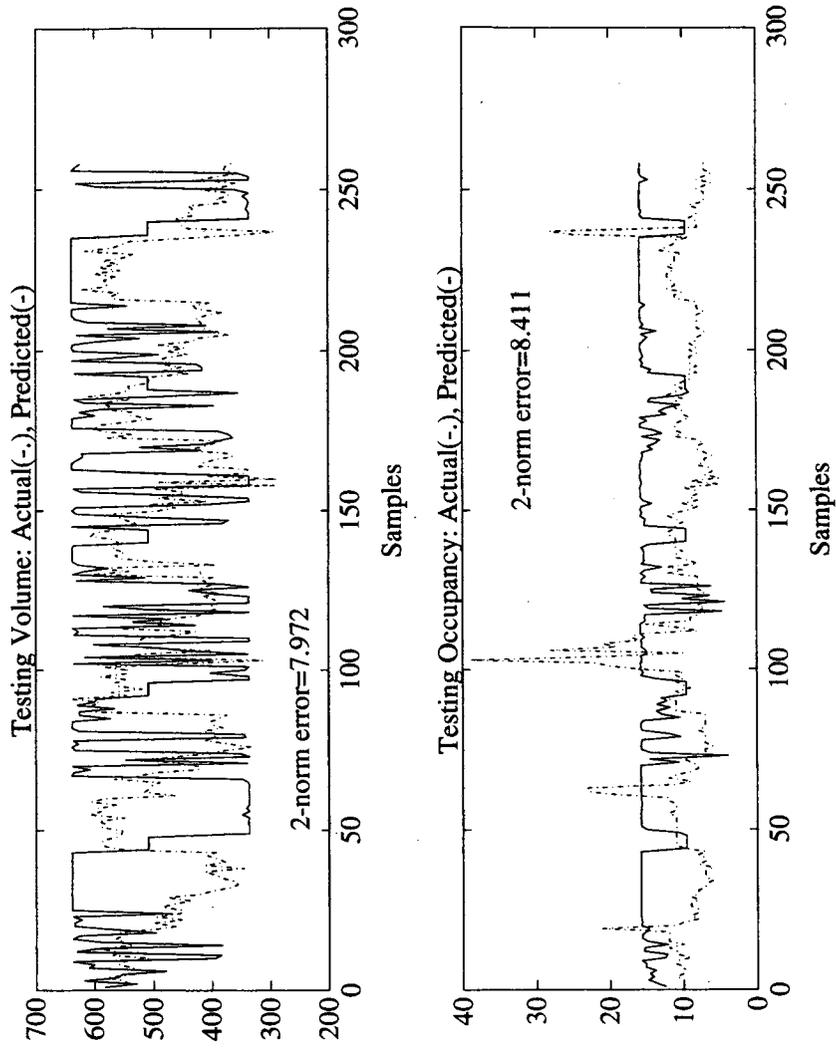


Figure B.8. Six-Day Testing Set for 5-minute Predictor

stations do not provide as much insight with 5-minute data as they do with 1-minute data. Nevertheless, one would hope for a better prediction with an additional upstream input station. Attempts using additional upstream input stations had the same problems as the previous example, such as network saturation and traversing the dynamic range.

FUTURE RESEARCH RECOMMENDATIONS

Although the long term prediction results were discouraging, predictions beyond 1 minute are probably feasible with a different learning technique and architecture. To avoid training difficulty, a second order learning algorithm such as the Newton, Conjugate Gradient, or Quasi-Newton method (Kung, 1993) might speed learning. Although more computationally intensive, these methods use the second derivative as well as the derivative of the error function to minimize the mean squared error.

Smoothing the data before training the ANN would make prediction an easier task. However, vehicles naturally travel in platoons, so many of the rapid data fluctuations would indicate actual traffic gaps rather than a noisy signal. Although some of the information would be lost during the smoothing process, a smoothed long-term prediction would be more useful than no prediction at all.

A technique to minimize the network becomes more important for larger neural networks. Two techniques that minimize the neural network are pruning and growing. Pruning eliminates inconsequential weights of a functional network. The fact that a weight is small does not necessarily mean that the output is insensitive to that weight, which may be the reason pruning did not help the previously mentioned ANNs. Growing, on the other hand, adds one neuron at a time until adding more neurons no longer improves performance. Projection Pursuit Learning (Hwang, Maechler, Martin, and Schimert, 1992) is a promising growing technique that learns a polynomial

activation function for each neuron in addition to learning the optimal number of neurons. Minimizing the ANN systematically should improve generalization abilities.

A third concern with predicting temporal signals is that the prediction might require a dynamic tapped delay line length (the number of past samples used as inputs). For instance, sometimes the predictor would need the past ten samples, while other times the predictor would only need the past five samples. Providing too many past samples might garble the prediction, while inadequate information would hinder the prediction. A dynamic length tapped delay line is difficult to program for an MLP, so a different architecture would be more suitable for temporal problems. Two recurrent ANN architectures that address this problem are Real-Time Recurrent Learning (RTRL) and Back Propagation Through Time (BPTT). The RTRL feeds back the hidden layer outputs as fully connected internal inputs (Williams and Zisper, 1989). The external inputs include only the past sample, so the network must internally store any additional past information that it needs. The BPTT is based on the concept that any recurrent neural network behaves in a manner identical to a feed forward network unfolded a layer for each time step (Rumelhart, McClelland, and the PDP Research Group, 1986).

APPENDIX C
FUZZY LOGIC CONTROLLER CODE

APPENDIX C: FUZZY LOGIC CONTROLLER CODE

```

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

#define M 10          /* number of fuzzification parameters */
#define N 12          /* number of inputs to controller */
#define R 27         /* number of control rules */
#define C 5           /* number of fuzzy classes */

/* Calculates a set of fuzzy inputs for each crisp input */
void fuzzify(float (*finput)[C], float *input, float (*parameter)[M]) {
    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;

    for (i=0; i<N; i++) {          /* ith input */
        LL=parameter[i][0];
        HL=parameter[i][1];
        if (HL-LL<0.001)
            printf("Warning: Division by small number, HL-LL=%7.4f\n",
HL-LL);
        crisp=input[i]/(HL-LL)-LL/(HL-LL);          /* normalize to 0-1 range */
        for (k=0; k<C; k++) {          /* kth class */
            base=parameter[i][5+k];
            if (k==0) {          /* VS 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 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];
                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;
    }
}

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

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

/* Rules finds the consequent of each rule, weights it according
to the importance of each rule, and aggregates the
contributions toward each fuzzy class */
void rules(float *fmr, float (*finput)[C], float *w) {
    FILE *control;
    int i;
    /* rule is the consequent for each rule */
    float rule[R]={ 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]};

    /* print rule outcomes to a file for tuning purposes */
    if ((control=fopen("control.txt", "a"))==NULL) {
        printf("Output file control.txt cannot be opened\n");
        exit(0);
    }
    for (i=0; i<R; i++) {
        if (rule[i]>0.0)
            fprintf(control, "rule[%d]=%.5.3f ", i, rule[i]);
        if (i%7==0)
            fprintf(control, "\n");
    }
    fprintf(control, "\n");

    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]+w[23]*rule[23];
    fmr[4]=w[4]*rule[4]+w[6]*rule[6]+w[11]*rule[11]

```

```

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

/* print aggregate consequents to a file for tuning purposes */
fprintf(control, "fmr={%6.3f, %6.3f, %6.3f, %6.3f, %6.3f}\n",
fmr[0], fmr[1], fmr[2], fmr[3], fmr[4]);
fclose(control);
}

/* Defuzz calculates a crisp metering rate using the discrete fuzzy centroid
when given the accumulated rule consequents for each class */
float defuzz(float *fmr, float (*parameter)[M]) {
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][0]; /* low limit for metering rate */
float HL=parameter[N][1]; /* high limit for metering rate */
int i;

/* Find areas and centroids of each fuzzy class of MR */
for (i=0; i<C; i++) {
base=parameter[N][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][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=%7.4f\n");
printf("May have insufficient rules firing\n");
if (HL-LL<0.01)
printf("Warning: division by small number\n");
printf("HL-LL=%7.4f\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=%7.4f\n",
output);
        return output;
    }

/* flc.c is a fuzzy logic controller that determines a metering rate based
on several inputs:
    parameters define memberships classes for each input (user specified)
    weights indicate the importance of each rule (user specified)
    inputs are the crisp inputs to the controller (from sensors) */
float flc(float (*parameters)[M], float *weights, float *inputs) {
class */
    float finputs[N][C];          /* fuzzified inputs */
    float fmr[C];                 /* accumulated consequent for each MR

    float MR;                     /* metering rate */

    fuzzify(finputs, inputs, parameters);
    rules(fmr, finputs, weights);
    MR=defuzz(fmr, parameters);    /* vehicles/20 secs */
    printf("MR=%7.4f\n", MR);
    return 20.0/MR;                /* return headway in seconds */
}

```

