Technical Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
SWUTC/98/465510-1		

4. Title and Subtitle	5. Report Date
Methodology for Traffic Signal Timing in Oversaturated Arterial Networks	October 1997
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Gye-Hyeong Ahn and Randy B. Machemehl	Research Report 465510-1

9. Performing Organization Name and Address	10. Work Unit No. (TRAIS)
Center for Transportation Research University of Texas at Austin 3208 Red River, Suite 200 Austin, Texas 78705-2650	11. Contract or Grant No. 0079

12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered
Southwest Region University Transportation Center Texas Transportation Institute The Texas A&M University System College Station, Texas 77843-3135	14. Sponsoring Agency Code

15. Supplementary Notes

Supported by a grant from the Office of the Governor of the State of Texas, Energy Office

16. Abstract

A traffic simulation model was developed to provide a methodology for traffic signal timing in oversaturated urban arterial networks. Two control objectives of traffic signal timing in oversaturated conditions were taken into consideration. One was to maximize the number of vehicles processed in an arterial network, which has an oversaturated traffic demand at the entry of the arterial street and moderate traffic demands at the entry of cross streets. The other was to prevent queue spillback or to minimize the occurrence of queue spillback if inevitable.

Signal timing offset was the dominant factor affecting system performance; although, link length was also important. When link length is short, the optimum offset is approximately zero, regardless of the cycle length. As link length increases beyond the minimum 200 feet tested, for highest efficiency, downstream intersection greens should begin before upstream intersection greens. This relationship, opposite to conventional progression greens, moves downstream queues before incoming platoon arrival.

Cross street traffic operations can have significant effects upon arterial performance and system efficiency. System efficiency rapidly deteriorates when any cross street green becomes too short for the link length. Therefore, a "practical" minimum green interval for cross streets is necessary to accommodate upstream cross street through traffic and turning vehicles from the arterial. When the cross street green is shorter than the minimum, even with the best offset combination, queue spillbacks occur on the cross streets and system efficiency deteriorates.

17. Key Words	18. Distribution Statement
Input, Output, Queue Length, Queue Spillback, Oversaturated, Undersaturated, Travel Time Offset, Cycle Length, Network Crossing Time, Link Length, Green Split	No Restrictions. This document is available to the public through NTIS: National Technical Information Service 5285 Port Royal Road Springfield, Virginia 22161

19. Security Classif.(of this report)	20. Security Classif.(of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	210	

Acknowledgment

This publication was developed as part of the University Transportation Centers Program which is funded 50% in oil overcharge funds from the Stripper Well settlement as provided by the Texas State Energy Conservation Office and approved by the U.S. Department of Energy. Mention of trade names or commercial products does not constitute endorsement or recommendation for use.

Disclaimer

The contents of this report reflect the views of the authors who are responsible for the facts and accuracy of the data presented. This document is disseminated under the sponsorship of the Department of Transportation, University Transportation Centers Program, in the interest of information exchange. The U.S. Government assumes no liability for the contents or use thereof.

Abstract

A traffic simulation model was developed to provide a methodology for traffic signal timing in oversaturated urban arterial networks. Two control objectives of traffic signal timing in oversaturated conditions were taken into consideration. One was to maximize the number of vehicles processed in an arterial network, which has an oversaturated traffic demand at the entry of the arterial street and moderate traffic demands at the entry of cross streets. The other was to prevent queue spillback or to minimize the occurrence of queue spillback if inevitable.

Signal timing offset was the dominant factor affecting system performance; although, link length was also important. When link length is short, the optimum offset is approximately zero, regardless of the cycle length. As link length increases beyond the minimum 200 feet tested, for highest efficiency, downstream intersection greens should begin before upstream intersection greens. This relationship, opposite to conventional progression greens, moves downstream queues before incoming platoon arrival.

Cross street traffic operations can have significant effects upon arterial performance and system efficiency. System efficiency rapidly deteriorates when any cross street green becomes too short for the link length. Therefore, a "practical" minimum green interval for cross streets is necessary to accommodate upstream cross street through traffic and turning vehicles from the arterial. When the cross street green is shorter than the minimum, even with the best offset combination, queue spillbacks occur on the cross streets and system efficiency deteriorates.

Executive Summary

If the traffic demand attempting to enter a signalized street network greatly exceeds the network capacity, it is described as oversaturated. Conventional, progression based network signal timing tends to encourage formation of longer and longer progressing platoons until they overfill links (spaces between intersections) producing spill backs into intersections. Grid lock follows spill back as cross street traffic finds paths blocked by vehicles composing spill backs. Recognizing the problems of network oversaturation, this study developed a signal timing methodology specifically designed to prevent platoon or queue spill back, and network gridlock. In order to properly examine oversaturated network performance and test signal timing concepts, a new computer simulation model was developed. This model was designed to be an efficient, authentic tool for oversaturated network evaluation.

The signal timing methodology suggests best signal timing parameters including offsets and cycle lengths based upon network travel time. More explicitly, the primary measure of effectiveness was time for a selected number of vehicles to traverse the network. The resulting timing methodology, opposite to conventional progressive timing, moves downstream queues before incoming platoon arrival.

Table of Contents

List of Figures vii

List of Tables x

CHAPTER 1 INTRODUCTION 1

1.1 STATEMENT OF PROBLEM 1

1.2 RESEARCH OBJECTIVES 3

1.3 LAYOUT OF THE REPORT 4

CHAPTER 2 BACKGROUND AND LITERATURE REVIEW 5

2.1 INTRODUCTION 5

2.2 TRAFFIC PERFORMANCE STATES 5

2.3 THEORETICAL APPROACHES 6

2.4 PRACTICAL GUIDELINES 8

2.5 TRAFFIC SIMULATION MODELS 8

2.6 LIMITATIONS AND PROBLEMS OF PREVIOUS STUDIES 11

2.7 SUMMARY 11

CHAPTER 3 CONCEPTUAL APPROACHES TO PROBLEM 13

3.1 INTRODUCTION 13

3.2 BASIC DEFINITIONS 13

3.3 METHODS FOR RESTRICTING QUEUE LENGTH 14

3.4 APPROACHES TO THE PROBLEM. 15

3.5 DEPARTURE HEADWAY AND STARTING RESPONSE OF TRAFFIC AT SIGNALIZED INTERSECTIONS 17

3.6 GREEN TIME AND AVERAGE TIME BETWEEN SUCCESSIVE VEHICLE STARTS 20

3.7 SUMMARY 21

CHAPTER 4 DEVELOPING A TRAFFIC SIMULATION MODEL 23

4.1 INTRODUCTION 23

4.2 THE FUNDAMENTALS OF TRAFFIC SIMULATION 23

4.2.1 Random Number Generation 23

4.2.2 Scanning and Updating 24

4.2.3 Bookkeeping 24

4.2.4 Assigning Driver/Vehicle Characteristics 24

4.2.5 The Entry Process 25

4.2.6 Warm-Up Time 25
4.3 TRAFFIC SIMULATION MODEL 26
4.3.1 Basic Assumptions and Logic of the Model 26
4.3.2 Structure of the Traffic Simulation Model 29
4.3.3 Subroutines of the Traffic Simulation Model 31
4.3.4 Description of an Example Simulation Run 35
4.3.5 Input 38
4.3.6 Validation and Calibration 40
4.4 SUMMARY 41

CHAPTER 5 DESIGN AND ANALYSIS OF SIMULATION EXPERIMENTS 43
5.1 INTRODUCTION 43
5.2 FORMULATION OF THE EXPERIMENTAL DESIGN 43
5.2.1 Objectives of Experiments 43
5.2.2 Factors and Responses 44
5.2.3 Fractional Factorial Designs and Factor-Screening Strategies 45
5.3 INTRODUCING VARIABILITY TO PARAMETERS 47
5.3.1 Departure Headway 47
5.3.2 Vehicle Space in the Queue 48
5.3.3 Vehicle Speed 49
5.4 ONE-WAY ARTERIAL OPERATION 61
5.4.1 Experimentation 62
5.4.2 Experimentation with Variability 71
5.4.3 Examples of Simulation Runs 74
5.4.4 Offset and Link Length 79
5.4.5 Green Split and Cycle Length 83
5.4.6 Formulation of Relationships 94
5.5 TWO-WAY ARTERIAL OPERATION 97
5.5.1 Experimentation with Two-Phase Operation 97
5.5.2 Offset and Link Length (Two-Phase Operation) 102
5.5.3 Experimentation with Three-Phase Operation 105
5.5.4 Offset and Link Length (Three-Phase Operation) 113
5.6 SUMMARY 116

CHAPTER 6 CONCLUSIONS 119
6.1 SUMMARY AND CONCLUSIONS 119
6.2 RESEARCH CONTRIBUTIONS 122

6.3 RECOMMENDATIONS 123

Appendix A 125

Appendix B 159

References 195

List of Figures

Figure 1.1. An idealized signal progression 2

Figure 2.1. Traffic Performance States 6

Figure 2.2. Simplified Flowchart for the TRANSYT-7F Model 10

Figure 4.1. Use of warm-up intersections to generate a platooned arrival process 25

Figure 4.2. Average overall speed 27

Figure 4.3. Conceptual flow of the simulation program execution 30

Figure 5.1. A vehicle space 48

Figure 5.2. Vehicle speed profiles 51

Figure 5.3. Field study site configuration 52

Figure 5.4. Relationship between Type 1 speed and downstream clear space (data points: 1061) 55

Figure 5.5. Downstream clear space and queue spillback 56

Figure 5.6. Relationship between Type 2 speed and distance for different queue positions 60

Figure 5.7. Relationship between Type 2 speed and travel distance (data points: 612) 61

Figure 5.8. Arterial configuration for a one-way operation 62

Figure 5.9. Network crossing time (sec) [one-way, L = 200 ft, C = 60 sec] 65

Figure 5.10. Network crossing time (sec) [one-way, L = 200 ft, C = 75 sec] 65

Figure 5.11. Network crossing time (sec) [one-way, L = 200 ft, C = 90 sec] 66

Figure 5.12. Network crossing time (sec) [one-way, L = 200 ft, C = 105 sec] 66

Figure 5.13. Network crossing time (sec) [one-way, L = 600 ft, C = 60 sec] 69

Figure 5.14. Network crossing time (sec) [one-way, L = 600 ft, C = 75 sec] 69

Figure 5.15. Network crossing time (sec) [one-way, L = 600 ft, C = 90 sec] 70

Figure 5.16. Network crossing time (sec) [one-way, L = 600 ft, C = 105 sec] 70

Figure 5.17. Offset 2 vs. network crossing time (one-way, L = 200 ft, offset 3 = 0) 80

Figure 5.18. Offset 3 vs. network crossing time (one-way, L = 200 ft, offset 2 = 0) 81

Figure 5.19. Offset 2 vs. network crossing time (one-way, L = 600 ft, offset 3 = 0) 81

Figure 5.20. Offset 3 vs. network crossing time (one-way, L = 600 ft, offset 2 = 0) 82

Figure 5.21. The effect of green split on the network crossing time (L = 200 ft, cross street V = 500 vphpl, with best offsets) 86

Figure 5.22. The effect of green split on the network crossing time (L = 200 ft, cross street V = 1800 vphpl, with best offsets) 86

Figure 5.23. The effect of green split on the network crossing time (L = 200 ft, cross street V = 500 vphpl, with worst offsets) 87

Figure 5.24. The effect of green split on the network crossing time (L = 200 ft, cross street V = 1800 vphpl, with worst offsets) 87

Figure 5.25. The effect of cycle length on the network crossing time (L = 200 ft, with best offsets) 88

Figure 5.26. The effect of cycle length on the network crossing time (L = 200 ft, with worst offsets) 89

Figure 5.27. The effect of green split on the network crossing time (L = 600 ft, cross street V = 500 vphpl, with best offsets) 91

Figure 5.28. The effect of green split on the network crossing time (L = 600 ft, cross street V = 1800 vphpl, with best offsets) 91

Figure 5.29. The effect of green split on the network crossing time (L = 600 ft, cross street V = 500 vphpl, with worst offsets) 92

Figure 5.30. The effect of green split on the network crossing time (L = 600 ft, cross street V = 1800 vphpl, with worst offsets) 92

Figure 5.31. The effect of cycle length on the network crossing time (L = 600 ft, with best offsets) 93

Figure 5.32. The effect of cycle length on the network crossing time (L = 600 ft, with worst offsets) 93
Figure 5.33. Relationship between simlation time and OFFD (L = 200 ft) 96
Figure 5.34. Relationship between simlation time and OFFD (L = 600 ft) 97
Figure 5.35. Arterial two-way operational configuration 98
Figure 5.36. Offset 2 vs. network crossing time (two-way, two-phase, L = 200 ft, offset 3 = 0) 102
Figure 5.37. Offset 3 vs. network crossing time (two-way, two-phase, L = 200 ft, offset 2 = 0) 103
Figure 5.38. Offset 2 vs. network crossing time (two-way, two-phase, L = 600 ft, offset 3 = 0) 103
Figure 5.39. Offset 3 vs. network crossing time (two-way, two-phase, L = 600 ft, offset 2 = 0) 104
Figure 5.40. Network crossing time (sec) [two-way, three-phase, L = 200 ft, C = 60 sec] 108
Figure 5.41. Network crossing time (sec) [two-way, three-phase, L = 200 ft, C = 75 sec] 108
Figure 5.42. Network crossing time (sec) [two-way, three-phase, L = 200 ft, C = 90 sec] 109
Figure 5.43. Network crossing time (sec) [two-way, three-phase, L = 200 ft, C = 105 sec] 109
Figure 5.44. Network crossing time (sec) [two-way, three-phase, L = 600 ft, C = 60 sec] 111
Figure 5.45. Network crossing time (sec) [two-way, three-phase, L = 600 ft, C = 75 sec] 111
Figure 5.46. Network crossing time (sec) [two-way, three-phase, L = 600 ft, C = 90 sec] 112
Figure 5.47. Network crossing time (sec) [two-way, three-phase, L = 600 ft, C = 105 sec] 112
Figure 5.48. Offset 2 vs. network crossing time (two-way, three-phase, L = 200 ft, offset 3 = 0) 113
Figure 5.49. Offset 3 vs. network crossing time (two-way, three-phase, L = 200 ft, offset 2 = 0) 114
Figure 5.50. Offset 2 vs. network crossing time (two-way, three-phase, L = 600 ft, offset 3 = 0) 114
Figure 5.51. Offset 3 vs. network crossing time (two-way, three-phase, L = 600 ft, offset 2 = 0) 115

List of Tables

TABLE 2.1. TRAFFIC SIMULATION MODELS 9

TABLE 3.1. COMPARISON OF VARIOUS RESEARCH RESULTS OF DEPARTURE HEADWAYS 18

TABLE 4.1. DESCRIPTION OF SUBROUTINES 31

TABLE 4.2. EXECUTION STEPS OF THE SUBROUTINE ARRQLA 33

TABLE 4.3. EXECUTION STEPS OF THE SUBROUTINE CROSST 33

TABLE 4.4. EXECUTION STEPS OF THE SUBROUTINE DEPQLS 34

TABLE 4.5. EXECUTION STEPS OF THE SUBROUTINES LEFTL AND RIGHTL 35

TABLE 4.6. OUTPUT EXAMPLE OF A SIMULATION RUN 37

TABLE 4.7. SIMULATION INPUT DATA 38

TABLE 5.1. DESIGN MATRIX BY QUALITATIVE FACTORS 45

TABLE 5.2. DESIGN MATRIX BY QUANTITATIVE FACTORS 45

TABLE 5.3. NUMBER OF SIMULATION RUNS (BY A FACTORIAL DESIGN) 46

TABLE 5.4. REVISED DESIGN MATRIX 46

TABLE 5.5. RESULTS OF FIELD STUDY FOR TYPE 1 AVERAGE OVERALL SPEED 54

TABLE 5.6. RESULTS OF FIELD STUDY FOR TYPE 2 AVERAGE OVERALL SPEED 59

TABLE 5.7. NETWORK CROSSING TIME (SEC) [ONE-WAY OPERATION, L = 200 FT] 64

TABLE 5.8. NETWORK CROSSING TIME (SEC) [ONE-WAY OPERATION, L = 600 FT] 68

TABLE 5.9. NETWORK CROSSING TIME (SEC) [ONE-WAY OPERATION, WITH VARIABILITY, L = 200 FT] 72

TABLE 5.10. NETWORK CROSSING TIME (SEC) [ONE-WAY OPERATION, WITH VARIABILITY, L = 600 FT] 73

TABLE 5.11. NETWORK CROSSING TIME (SEC) [FOR 1200 VEHICLES, L = 200 FT] 75

TABLE 5.12. NUMBER OF VEHICLES SIMULATED ON EACH LANE (FOR 2000 TOTAL VEHICLES) 75

TABLE 5.13. NUMBER OF VEHICLES SIMULATED ON EACH ARTERIAL LANE PER CYCLE (FOR 2000 TOTAL VEHICLES) 76
TABLE 5.14. NETWORK CROSSING TIME (SEC) [FOR 1200 VEHICLES, L = 600 FT] 77
TABLE 5.15. NUMBER OF WARM-UP VEHICLES 78
TABLE 5.16. NUMBER OF VEHICLES SIMULATED ON EACH LANE (FIXED OFF3) 78
TABLE 5.17. NUMBER OF VEHICLES SIMULATED ON EACH ARTERIAL LANE PER CYCLE 79
TABLE 5.18. DESIRABLE OFFSET RANGES (ONE-WAY OPERATION) 82
TABLE 5.19. UNDESIRABLE OFFSET RANGES (ONE-WAY OPERATION) 83
TABLE 5.20. THE EFFECT OF GREEN SPLIT ON THE NETWORK CROSSING TIME (L = 200 FT) 85
TABLE 5.21. THE EFFECT OF GREEN SPLIT ON THE NETWORK CROSSING TIME (L = 600 FT) 90
TABLE 5.22. RESULTS OF MULTIPLE REGRESSION ANALYSES 95
TABLE 5.23. NETWORK CROSSING TIME (SEC) [TWO-WAY, TWO-PHASE, L = 200 FT] .. 100
TABLE 5.24. NETWORK CROSSING TIME (SEC) [TWO-WAY, TWO-PHASE, L = 600 FT] .. 101
TABLE 5.25. DESIRABLE OFFSET RANGES (TWO-WAY AND TWO-PHASE OPERATION) 104
TABLE 5.26. UNDESIRABLE OFFSET RANGES (TWO-WAY AND TWO-PHASE OPERATION) 105
TABLE 5.27. NETWORK CROSSING TIME (SEC) [TWO-WAY, THREE-PHASE, L = 200 FT.] 107
TABLE 5.28. NETWORK CROSSING TIME (SEC) [TWO-WAY, THREE-PHASE, L = 600 FT] 110
TABLE 5.29. DESIRABLE OFFSET RANGES (TWO-WAY ARTERIAL AND THREE-PHASE SIGNAL OPERATION) 115
TABLE 5.30. UNDESIRABLE OFFSET RANGES (TWO-WAY ARTERIAL AND THREE-PHASE SIGNAL OPERATION) 116

CHAPTER 1 INTRODUCTION

1.1 STATEMENT OF PROBLEM

A coordinated arterial signal system consists of two or more traffic signals having a fixed time relationship to each other. The relationship among arterial signals may be designed to permit travel without stopping or progression. If the system is operated without coordination, progression is not likely and increased stops, delay, and fuel consumption may result. Each time platoons stop, green time is lost and delay is increased. Stops are generally related to delay, but they are not necessarily related proportionally. Fuel consumption increases with delay and stops. All signals in the system must be coordinated if progression along the arterial is provided.

The simplest signal coordination scheme would provide progression along a one-way arterial or in one direction on a two-way arterial. If all vehicles travel at some design speed, v, the standard scheme for coordination of signals on a one-way arterial is as illustrated in Figure 1.1. All signals operate on the same cycle time and have the same green interval for the progressed direction. The offsets are chosen so that the start of green at each intersection occurs at a time d after that for the adjacent upstream intersection with d = d/v equal to the trip time at the design speed between the two intersections at spacing d. As shown in Figure 1.1, with this signal coordination, any vehicle traveling on the arterial at the design speed which passes the first signal at the start (end) of the green, will pass every other signal at the start (end) of the green.

In a coordinated arterial signal system, platoon flows are common and more closely represent reality. As the platoon travels through the signal system, it forms a progression (green) band which is the time in seconds elapsed between the passing of the first and the last possible vehicle in the platoon moving in accordance with the design speed of a progressive signal system.

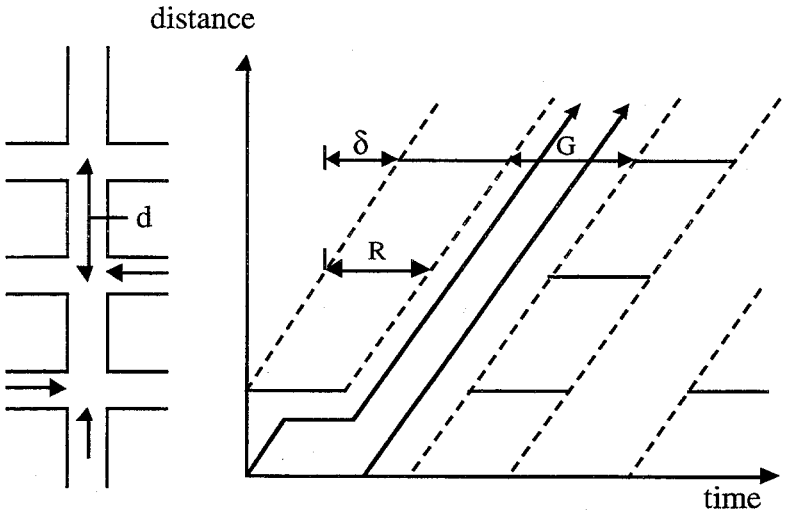


Figure 1.1: An idealized signal progression

Vehicles turning onto the arterial from a cross street add more complexity to the initial condition which only considers the arterial. The turning vehicles form a queue at a signal on the arterial during a red interval. A queue formed in this manner has an effect on the progression band. The incoming platoon on the arterial would have to stop or reduce speed while the waiting queue starts and begins moving. A queue clearance time must be used to clear the queue formed by the cross street traffic. If the average stopped queue for the through traffic is known, then the queue clearance time can be calculated from

$$Q_i = \frac{N_i}{S_i / 3600} + u_i \qquad (1.1)$$

where

Q_i = queue clearance time in seconds;

N_i = average number in queue at start of green;

S_i = saturation flow;

u_i = queue start-up time.

To provide progressive movement on the arterial when stopped queues exist, the queue clearance time Q_i is subtracted from the through traffic green time g_i to determine the resulting progressive through green time G_i:

$$G_i = g_i - Q_i \qquad (1.2)$$

As the arterial demand grows, the platoon length eventually exceeds the progression band and oversaturation occurs. When this heavy demand continues for a long time period the queue length may extend into the upstream intersection. Not only through traffic on the arterial but also cross street traffic may be blocked by the extended queue. Queue spillback to upstream intersections is common in oversaturated arterial networks, especially when intersections are closely spaced. Queue spillback blocks traffic along the arterials, disrupts progressive movements, and forms standing queues in the presence of green signal indications. Recurring queue spillback is the dominating factor influencing traffic operations in arterial networks. A primary objective for developing a control policy is to prevent queue spillback.

Many studies have been performed and applied successfully for the control of undersaturated intersections, but most of them have been ineffective or invalid in oversaturated conditions. There has been relatively limited research on the area of traffic control for oversaturated environments, and most of the research has been too theoretical to be applied in a real system. A number of traffic optimization models such as TRANSYT-7F and PASSER-II can develop optimal signal timing plans for undersaturated arterials, but none of these is applicable for oversaturated conditions.

1.2 RESEARCH OBJECTIVES

In general, the control objective of traffic signal timing in undersaturated environments has been to maximize bandwidth and/or to minimize delay. When demands are extremely high, however, the control strategy should be changed because of the different traffic characteristics of oversaturated operations. Two control objectives of traffic signal timing in oversaturated conditions are taken into consideration. One is to maximize the throughput, or the number of vehicles processed, in the arterial system. The other is to prevent queue spillback or to minimize the probability of queue spillback if inevitable. Queue spillback should be treated with caution because it may have serious arterial system effects.

The main objective of this research is to develop a traffic simulation model to provide a methodology for traffic signal timing in oversaturated urban arterial networks. This model is designed to accommodate two-way arterial systems which have at least three intersections with multi-phase signal operation and one-way cross streets. To simplify the analysis, a basic form of a coordinated arterial network, such as a one-way street, two-phase signals, etc, will be considered first. The results can be extended for more complicated situations.

1.3 LAYOUT OF THE REPORT

This report consists of six chapters, including this introductory chapter. Chapter 2 provides a literature review of theoretical and practical approaches for developing traffic control policies in oversaturated environments. Traffic simulation models are also described. Chapter 3 describes the conceptual approach to the problem, including analysis of departure headway and starting response of traffic at signalized intersections. Chapter 4 describes details of the traffic simulation model which include the code details, logic and assumptions, an example simulation run, input details, and calibration and validation of the model. Chapter 5 discusses the experimental design, simulation experiments, and analysis of experimentation results. This chapter also describes introducing variability to parameters and average overall speed data collections. Chapter 6 discusses the summary and conclusions from the study, research contributions, application of results, and further study needs.

CHAPTER 2 BACKGROUND AND LITERATURE REVIEW

2.1 INTRODUCTION

Traffic control signals for street traffic were first used more than a century ago. It was in the late 1950s that Webster initiated studies on traffic signal timing. Research on traffic signal control has since been performed on various problems but it focused on undersaturated traffic conditions. A limited number of studies have treated the area of traffic control for oversaturated operations. The purpose of this chapter is to give a systematic discussion of the various subjects related to traffic control during oversaturated traffic conditions.

In this chapter, the following topics will be reviewed and discussed. First, definitions of traffic performance states will be given. Second, theoretical approaches to developing a control policy for an oversaturated environment will be discussed, followed by reviews of practical guidelines for traffic control in oversaturated operations. Finally, traffic simulation models which might have application to oversaturated conditions are described.

2.2 TRAFFIC PERFORMANCE STATES

It is not unusual to find the terms congested, saturated, and oversaturated all applied to the same situation, i.e., where one or more vehicles remain to be served at the end of the green signal phase. A more exact and nonredundant set of terms are needed to characterize the various traffic performance conditions that could occur.

Lee, et al. (1975) gave good definitions of traffic performance states as shown in Figure 2.1. Specifically, the definitions have been constructed around a queue formation mechanism and, as such, they relate to the extent and growth of queues. The following definitions are all described in terms of one approach to a signal.

The term *uncongested operations* is characterized as a situation where there is no significant queue formation. *Congested operations* represents the entire range of operations which may be experienced when traffic demand approaches or exceeds, or both, the capacity of the signal. Since this is not a sufficient definition to describe the problem, the realm of congested operations has been divided into two major categories: saturated and oversaturated operations. *Saturated operations* has been further subdivided into stable and unstable ranges. *Stable saturation* exists when a queue has formed and is not growing, and delay effects are local. *Local*

effects in this context implies that traffic performance is only affected at the intersection at which the queue occurs, and that no other intersection's performance is affected by this queue. *Unstable saturation* exists when a queue exists and is growing, and delay effects are still local. *Oversaturated operations* refers to a situation wherein a queue exists, and it has grown to the point where the upstream intersection's performance is adversely affected.

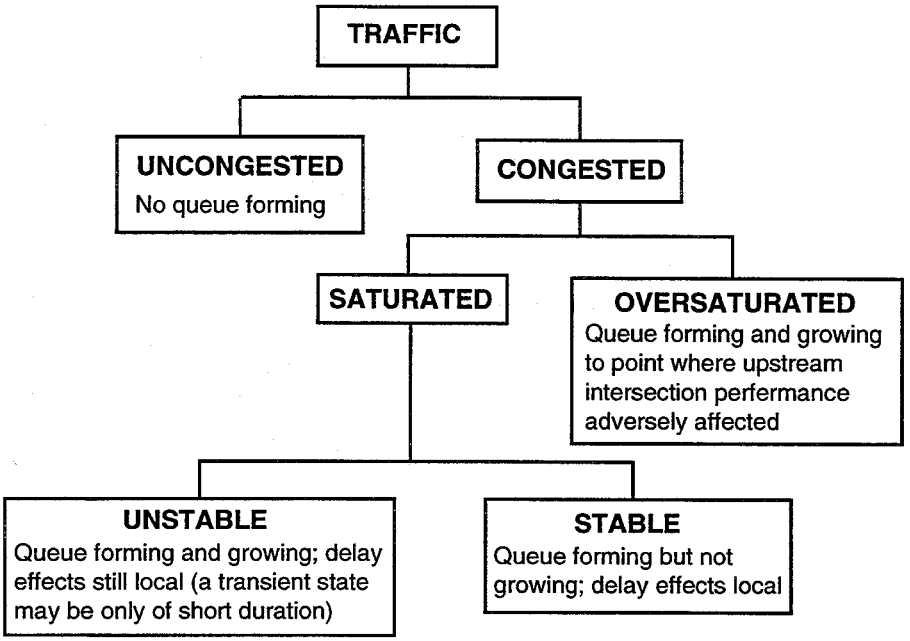


Figure 2.1: Traffic Performance States

2.3 THEORETICAL APPROACHES

Within the past 30 years a number of studies have been performed to develop a traffic control policy for oversaturated operations. For pretimed signals, the control of an oversaturated intersection was first considered by Gazis and Potts (1965) who presented a method for obtaining traffic signal settings that optimize the movement of traffic during the oversaturation period. In another paper, Gazis (1964) extended the control policy to two oversaturated linked intersections with one-way operation. He solved this problem by using the semi-graphical methods employed in a previous isolated intersection paper.

Longley (1968) proposed a control strategy for real time control of an isolated intersection and a network of four intersections. His basic control philosophy was based upon the fact that

traffic signals cannot clear queues in conditions of primary congestion, and their function is thus to maintain the queues to the extent of not blocking a side road in order to delay the onset of secondary congestion. It was not applied in a real system.

Lee, et al. (1975) developed a control policy for oversaturated intersections. It is a queue-actuated signal control in which an approach receives green automatically when the queue on that approach becomes equal to or greater than some predetermined length. A primary objective of this control policy is to delay or eliminate intersection blockage, which is the outgrowth of oversaturation.

Michalopoulos and Stephanopoulos (1977) suggested an optimal control policy for oversaturated intersections. In addition to queue length constraints, travel time between the two intersections and turning movements were taken into account. In another paper (1981) they developed a real time control policy minimizing total intersection delays subject to queue length constraints. Their proposed policy was tested against the pretimed control policy at a high volume intersection and it was found superior. It was only applicable for isolated intersections at which traffic arrivals are not affected by upstream signals.

Shibata and Yamamoto (1984) proposed on-line real-time congestion control for isolated intersections with multi-phase operation. Lieberman (1986) and Rathi (1988) suggested a control scheme for high traffic density networks which is based on a spillback avoidance approach rather than the conventional progressive movement approach. One of the two proposed control policies was designed to limit the spillback of cross street queues by providing optimal offsets and increased green times along the cross streets. The other was queue management control along arterials, a form of internal metering, which was designed to manage queue lengths to reduce the spillback probability. They showed that the optimal relative offsets along arterials are approximately zero (simultaneous green) in the presence of a moderate queue along arterials and offsets providing reverse progression for cross streets are optimal or near optimal for streets with long queues and low discharge rates.

Kim (1990) developed a dynamic optimization model that provides optimal signal control strategies and a queue management scheme for oversaturated intersections. The dynamic model was designed to accommodate two-way arterials whose intersections have multi-phase operations and variation of demand during the control period. The model controls queue length by the efficient and timely changing of signal timing plans as demand changes and its control strategies are also designed to minimize the transitional delay caused by frequent changes of the

timing plans. Rouphail (1991) proposed a dynamic and time-dependent approach for analysis of congested flow at signalized intersections.

2.4 PRACTICAL GUIDELINES

Practical guidelines have been developed to help traffic engineers examine the cause and seriousness of the traffic congestion problem. Pignataro, et al. (1978) proposed a set of guidelines developed for the treatment of traffic congestion on street networks. The guidelines provided both a tutorial and an illustrated reference regarding what techniques to consider and how to consider them systematically. The various treatments and remedies were classified into the three categories, minimal-responsive signal control policies, highly responsive signal policies, and nonsignal control policies, i.e., other treatments in a signalized environment. OECD (1981) provided policy-makers and traffic engineers with an up-to-date assessment of traffic congestion management and highlighted the effects of traffic congestion in terms of energy waste, environmental nuisance, and driver annoyance.

2.5 TRAFFIC SIMULATION MODELS

Traffic simulation can provide traffic engineers with an effective tool for evaluating traffic system management strategies before implementing such strategies in the field. The simulation approach is less costly than empirical studies and can provide results in a fraction of the time needed for field experiments without disrupting traffic operations. Simulation provides a high level of modeling detail and expands the opportunity for developing new and innovative management strategies. Traffic simulation provides a wide range of MOE's that cannot be obtained empirically.

Many algorithms and computer simulation models are available today for analyzing various highway system operating environments. These operating environments contain signalized intersections, arterial networks, freeway corridors, and rural highways. Both microscopic and macroscopic computer simulation models have been developed for each of the operating environments stated above. The models can be categorized as shown in Table 2.1 (May, 1990).

Among the traffic simulation models for arterial networks, only TRANSYT-7F and TRAF-NETSIM have a feature to take into account the effect of queue spillback. The latter is the only microscopic computer simulation model available for arterial networks and it is described later in this section.

TABLE 2.1: TRAFFIC SIMULATION MODELS [1)]

Operating environment	Microscopic	Macroscopic
Signalized intersections	TRAF-NETSIM, TEXAS	CALSIG, CAPCAL, CAPSSI, POSIT, SIDRA, SOAP-84, SIGNAL-85
Arterial networks	TRAF-NETSIM	MAXBAND, SPAN, SSTOP, PASSER-II, PASSER-III, TRANSYT-7F, SIGOP-III
Freeway corridors	INTRAS	CORQ, FREQ, FRECON2, KRONOS
Rural highways	TWOPAS, VTI, TRARR	RURAL

1) Some optimization models are included

The Traffic Network Study Tool (TRANSYT) is one of the most widely used models in the world. It was developed in England with Robertson as the principal author. TRANSYT-7F is the Federal Highway Administration's version of TRANSYT-7 which is one of nine English versions (TRANSYT1 through TRANSYT9). TRANSYT-7F has continuously been enhanced in the United States during the1980s and is the version most widely used.

TRANSYT-7F is a macroscopic and deterministic single-time period simulation and optimization model. The simulation submodel calculates the performance of an arterial network for a specified signal timing. The optimization submodel is a hill-climbing optimization process which determines the near-optimum signal timing plan. A simplified flowchart for the TRANSYT-7F model is depicted in Figure 2.2.

A number of enhancements were made to the modeling capabilities of TRANSYT-7F (Release 6). One of them includes an expansion of the optimization objective function to optionally include excess queue backup (spillback) and/or operating cost. The queue backup length is the maximum extension of the queue upstream on the link during one cycle. TRANSYT-7F reports the maximum back of queue (queue length) and queue capacity values for each link in the network. By comparing the two values, the user can easily identify where spillover may occur. Although TRANSYT-7F considers the excess maximum back of queue in the optimization

process where an objective function includes the excess maximum back of queue term, the simulation still does not explicitly deal with spillover.

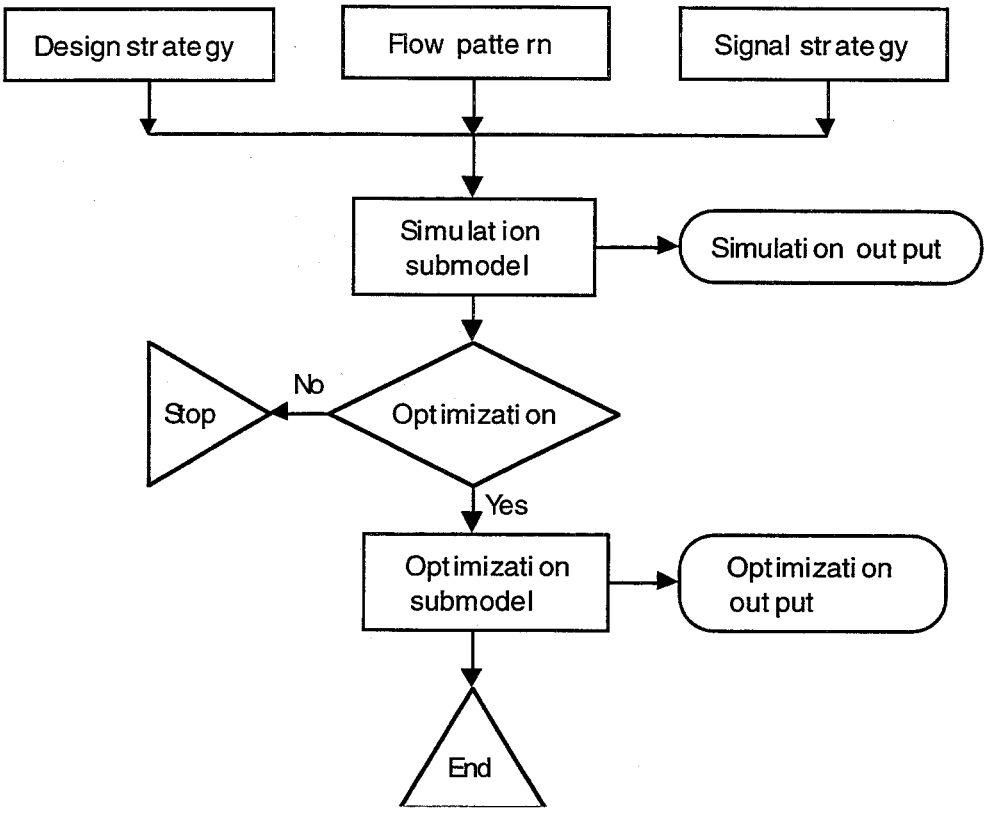


Figure 2.2: Simplified Flowchart for the TRANSYT-7F Model

TRAF-NETSIM is the successor to FHWA's well-known NETSIM traffic simulation program for urban street networks. It is a microscopic and stochastic model based on traffic flow theory and traffic simulation concepts. TRAF-NETSIM simulates individual vehicles as they traverse a network of urban streets. TRAF-NETSIM is a powerful tool for analyzing complex urban network traffic problems. It has a feature to model saturated conditions and intersection overflow. Wong (1990) described how TRAF-NETSIM simulates oversaturated traffic conditions as follows:

> "--- If the receiving lanes are full, a vehicle discharging from the stop line may either wait or join the queue. If it is a left- or right-turning vehicle, it will always join the queue and block the intersection. If it is a through vehicle, the program assigns a probability (user

specified or default) of joining the queue. (The default probability is 1.00 for the first through vehicle, 0.81 for the second, 0.69 for the third, and 0.40 for the fourth.) Vehicles waiting at the stop line will incur delay but will not affect cross-street traffic. Vehicles blocking the intersection will affect cross-street traffic."

Even though TRAF-NETSIM does not have the capability of signal-timing optimization, it is applicable to oversaturated traffic environments where TRANSYT-7F only has limited capability.

2.6 LIMITATIONS AND PROBLEMS OF PREVIOUS STUDIES

The control of two oversaturated intersections was first considered by Gazis (1964) where it was assumed that travel time as well as queueing storage between the two intersections was negligible, and turning movements were ignored. In other words, the two intersections were treated as one. In the early 1970's some attempts were made to remove these deficiencies, but there was little improvement. For real time control, some algorithms have been proposed, (Longley, 1968; Gordon, 1969; Lee et al., 1975) but it should be noted that none has been applied in a real system due to their complex computational requirements and the extensive instrumentation required for their implementation.

Considering travel time between the two intersections, queue length constraints and turning movements, Michalopoulos (1977b) dealt with the problems stated previously, but his approach was limited to a one way street and simple phase operation. His real time control policy (1981) was tested and validated but it was only applied to isolated intersections. Kim (1990) solved some questions by accommodating two-way street intersections with multi-phase operations and variable demand during the control period in his dynamic model. The stochastic nature of traffic demand within each time slice, however, was not explicitly considered in the formulation of the model. It is necessary to validate his model in a real system because artificial traffic data were used to test the model.

2.7 SUMMARY

For both pretimed and real time controls, theoretical and practical approaches for developing traffic control policies in oversaturated environments were reviewed. Traffic simulation models which are applicable to oversaturated conditions, like TRANSYT-7F and TRAF-NETSIM, were described. Finally, limitations and problems of previous studies, which are partly because of

oversimplified assumptions used and partly because of it's lack of applicability in a real system, were discussed.

CHAPTER 3 CONCEPTUAL APPROACHES TO PROBLEM

3.1 INTRODUCTION

This chapter describes basic definitions, queue length management schemes, approaches to the problem, and conceptual analysis of oversaturated traffic conditions. The departure headway and average time between successive vehicle starts are important parameters in the study underlying the traffic simulation model. The literature review and selected data of departure headway and average time between successive vehicle starts are described.

3.2 BASIC DEFINITIONS

Consider a simplified isolated intersection with one-way single-lane approaches and no turning movements. At this stage, assume lost-time and start-up delays are not taken into consideration. Each intersection has conflicting movements that cannot be accommodated simultaneously. Movements into and out of the intersection can be described in terms of input and output. Input is defined as the number of vehicles arriving at an intersection approach during a certain time period and output is defined as the number of vehicles discharged from the stop line (arrivals and departures of vehicles are assumed to be deterministic). Vehicles discharged from the stop line are assumed to have uniform headways under oversaturated conditions, and hence, for that approach, output is proportional to the green interval. For the same reason input is proportional to the green interval for the corresponding upstream intersection approach. Oversaturation occurs when input exceeds output. Input and output can be represented by

$$\text{Output}_{i,j} = \frac{(g_{i,j} / C_j)T}{h}$$

$$\text{Input}_{i,j} = \frac{(g_{i,j-1} / C_{j-1})T}{h} \qquad (3.1)$$

where

$g_{i,j}$ = green interval for approach i at intersection j

$g_{i,j-1}$ = green interval for approach i at upstream intersection j-1

C_j = cycle length of intersection j

C_{j-1} = cycle length of upstream intersection j-1

T = analysis period

h = headway.

Therefore, when ($g_{i,j-1}/C_{j-1}$) is greater than ($g_{i,j}/C_j$), oversaturation occurs and queues form. Queue length (Q_L) can be derived in terms of input and output as follows:

$$Q_L = \text{Input} - \text{Output}$$

When $g_{i,j-1}/C_{j-1}$ is greater than $g_{i,j}/C_j$, then Q_L is non-zero and increases with time T; otherwise, Q_L equals zero:

$$Q_L = \frac{(g_{i,j-1} / C_{j-1} - g_{i,j} / C_j)T}{h} \qquad (3.2)$$

One objective for treating oversaturated conditions is limiting the queue length so as not to block the upstream intersection, or in other words, to prevent queue spillback. When input exceeds output for a long time period, the queue length may extend into the upstream intersection. Queue length must be restricted to the maximum allowable length, which is usually equal to the queue storage length (S_L). Queue spillback occurs when the queue length is greater than the queue storage length. One question which arises here, is how to prevent the queue length from exceeding the queue storage length. From equation (3.2) the possible way of reducing queue length is to decrease the value of ($g_{i,j-1}/C_{j-1}$) or to increase the value of ($g_{i,j}/C_j$) or to do both.

3.3 METHODS FOR RESTRICTING QUEUE LENGTH

The queue length can be reduced by either increasing the output or decreasing the input. Each of the procedures is described below.

1. If the output is increased by increasing ($g_{i,j}/C_j$) ratio, then either $g_{i,j}$ should be increased or C_j should be decreased. Increasing $g_{i,j}$ leads to an increase in the input on the same approach at the downstream intersection and a decrease in the green time for the cross street. Decreasing C_j has the same effects on the cross-street and the downstream intersection. If progression along the arterial is provided, the common cycle length for the arterial should be decreased, otherwise only C_j can be decreased.

2. If the input is decreased by decreasing the ($g_{i,j-1}/C_{j-1}$) ratio, then either $g_{i,j-1}$ should be decreased or C_{j-1} should be increased. Decreasing $g_{i,j-1}$ leads to a decrease in the input to the corresponding approach at the downstream intersection and an increase in the green time for the cross street. Increasing C_{j-1} has the same effects on the cross-street and the downstream intersection. If progression along the arterial is provided, the common cycle length for the arterial should be increased, otherwise only C_{j-1} can be increased.

The advantage of the first method is that more vehicles can be accommodated along the arterial during a given time period. This method gives priority to the arterial over the cross streets. But this option leads to a decrease in the green time on the cross street, and it may deteriorate cross-street traffic conditions. On the contrary, the second method can provide the cross street with more green time, which improves cross-street traffic conditions. Thus, this method gives priority to the cross streets over the arterial and can accommodate fewer vehicles along the arterial. The second method seems to be better than the first one, based on the following arguments; (1) In the first method, increasing the output by increasing ($g_{i,j}/C_j$) ratio may worsen the traffic performance of the downstream intersection; (2) The second method improves cross street traffic conditions.

3.4 APPROACHES TO THE PROBLEM.

In the previous section the different methods that can be used to limit queue length to the storage length on the arterial have been discussed. In this section, some basic concepts used in developing a strategy for dealing with oversaturated conditions will be discussed.

If a link in a network is saturated, or in other words, $Q_L = S_L$ on the link, then, assuming uniform driver behavior, the green interval required to clear the vehicles on this link is given by Greenshield's queue departure model as follows.

$$\begin{aligned} g &= d + h(n) \\ &= 4 + 2n \end{aligned} \qquad (3.3)$$

where

g = green interval

d = starting delay

h = headway

n = number of vehicles to be cleared

(= $Q_L = S_L$, in case of single-lane)

For example, when the link length is 400 ft and assuming the vehicle length plus average vehicle spacing equals 20 ft, a maximum of 20 vehicles can be stored in that link. The green interval required to clear 20 vehicles is 44 seconds by equation (3.3).

If two adjacent links in a network are saturated, one can consider two extreme cases for dealing with this condition depending upon the way vehicles are cleared from the upstream intersection.

Case 1. The first vehicle at the upstream intersection stop line is not allowed to move forward until all the vehicles stored in the downstream link are cleared. This method is illustrated with examples for uniform and nonuniform block spacing.

If the uniform block spacing is 400 ft, then the number of vehicles in each of the adjacent links is 20. The green interval required to clear each of the links is 44 seconds. Since, in this case, all vehicles in the downstream link are cleared before the vehicles in the upstream link are allowed to move, the offset for the upstream intersection should be 44 seconds with reference to the downstream intersection.

If the block spacing is nonuniform, and is equal to 200 ft and 400 ft for the downstream link and the upstream link respectively, then the number of vehicles on the downstream link is equal to 10 and that on the upstream link is equal to 20. Hence, the green interval required to clear the vehicles on the downstream link is 24 seconds. For the same reason mentioned above, the offset for the upstream intersection should be 24 seconds with reference to the downstream intersection. Two possible green intervals can be considered for the upstream link, a 24-second interval and a 44-second interval. If a 24-second interval is used, only 10 out of the 20 vehicles are cleared from the upstream link. If a 44-second interval is used for the upstream link, the number of vehicles entering the downstream link exceeds its storage length and may block the intersection. Here, one can see that the green interval at the upstream intersection is dependent upon that at the downstream intersection.

From the above example one notes that this method leads to wasted storage space which results in unnecessary arterial delay.

Case 2. The first vehicle at the stop line of the upstream intersection is allowed to follow the last vehicle in the downstream link as soon as the last vehicle starts moving forward. If this method is used for the first example with uniform block spacing, which has already been described in Case1, the offset for the upstream intersection should be less than 44 seconds with reference to the downstream intersection. How much less than 44 seconds depends upon the time the last vehicle, at the downstream, starts moving. This method yields a smaller offset, better storage space utilization, and hence lesser delays along the arterial in comparison with Case1. More about this method will be discussed later in Section 3.5.

From the above two cases, one can observe that the storage space limitation governs the green interval, and the method used to clear the queues governs the offsets, which in turn affects performance measures such as delay and average travel time.

Consider right turns during red signal indications that usually affect intersection outputs. Right-turn traffic from the cross street onto the arterial during the cross street red interval leads to

increased input to the arterial. Right-turn traffic from the arterial into the cross street during the red arterial interval yields an arterial output increase. For oversaturated conditions in which the link storage space is constantly filled, potential right turners have no space into which they can move. Therefore, the only option for right turners is to move into the intersection waiting for a link storage space. Therefore, right-turn traffic on red intervals in this type of oversaturated condition can be ignored.

Conceptual analysis of oversaturated traffic conditions

On an intersection approach, if input exceeds output, a queue forms and grows. If input continues to exceed output, the entire link, which is the approach storage area, will be filled and one more vehicle joins the queue and blocks the intersection. This is called queue spillback and occurs in oversaturated traffic conditions. This queue spillback will block the same direction traffic flow until it is cleared. If it remains after its own green signal duration, it also will block cross street traffic flow.

Oversaturated arterial networks composed mainly of passenger cars, have several common traffic flow characteristics. First, drivers do not have enough freedom to move as they wish. Thus, they have very limited opportunities to pass or change lanes. Second, variance of parameters (departure headway and average overall speed, etc.) are relatively small. For example, very long departure headways or high vehicle speeds are very unlikely.

Queue spillback causing long delay (especially, when it is not cleared before its own green signal duration ends) is more problematic, because it has an adverse effect not only on upstream intersections but also cross streets. As discussed before, one control objective of traffic signal timing in oversaturated conditions is to prevent queue spillback or to minimize the probability of queue spillback in the arterial network if inevitable. Therefore, a queue spillback avoidance approach should be a key strategy in oversaturated traffic conditions.

3.5 DEPARTURE HEADWAY AND STARTING RESPONSE OF TRAFFIC AT SIGNALIZED INTERSECTIONS

The departure headway is the time between successive vehicles departing a stop line of a signalized intersection after the signal turns green. The values of several important parameters regarding signalized intersection operation - such as saturation flow rate, starting delay, and lost time - are often derivatives of departure headway measurements. Many researches and investigations on the departure headway have been performed.

Greenshields et al. (1947) studied traffic flow behavior at intersections in New York City and New Haven, Connecticut. It was one of the earliest efforts to quantify vehicle flow characteristics on approaches to intersections. According to their data, an average of 3.8 seconds was necessary for a queue of 6 or more stopped vehicles to begin moving after the traffic signal turned green. The successive mean headways for the following vehicles entering an intersection from a stopped queue are shown in Table 3.1.

TABLE 3.1: COMPARISON OF VARIOUS RESEARCH RESULTS OF DEPARTURE HEADWAYS

n	Greenshields ('47)	Gerlough & Wagner ('67)	Carstens ('71)	Wilkinson & King ('76)
1	3.8	3.85	2.64	2.61
2	6.9	6.66	5.13	5.61
3	9.6	9.17	7.62	8.13
4	12.0	11.64	9.91	10.50
5	14.2	14.01	12.20	12.71
6	16.3	16.37	14.49	14.85
7	18.4	18.77	16.78	16.99
8	20.5	21.08	19.07	19.13
9	22.6	23.32	21.36	21.27
10	24.7	25.66	23.65	23.41
Screen line	intersection line (front of car)	n/a	stop line (front wheel)	stop line (front or rear wheel)
Relationship	3.7+2.1n (n≥5)	2.44+2.3n (n≥5)	0.75+2.29n (n≥3)	2.01+2.14n (n≥6)
n	Lee & Chen ('86) (1)	Lu ('84) (left-turn)	Moussavi & Tarawneh ('90)	Efstathiadis ('92)
1	3.80	2.43	2.90	2.04
2	6.36	5.05	4.94	4.50
3	8.71	7.15	7.04	6.62
4	10.93	9.24	9.08	8.62
5	13.09	11.04	10.95	10.55
6	15.12	12.84	12.86	12.45
7	17.09	14.64	14.61	14.30
8	19.03	16.44	16.36	16.12
9	20.97	18.24	18.11	17.91
10	22.75	20.04	19.86	19.70
Screen line	stop line (rear bumper)	intersection line (front of car)	n/a	front of first car (front of car)
Relationship	2.29+2.09n 1) (n≥2)	2.04+1.8n (n≥5)		1.34+1.82n (n≥4)

1) Regression analysis was applied to get a relationship. (R = 0.999)

George and Heroy (1966) conducted an interesting study to determine the time required for each vehicle in a line of stopped vehicles to begin its forward motion after the beginning of the green signal at a signalized intersection. In this study, the time lag from the beginning of a green period to the start of forward motion of vehicles for each position from an intersection stop line

was measured. The relationship between vehicle position and average time of starting from a stopped position approximated a straight line. The average time between successive vehicle starts was approximately 1.4 seconds.

Gerlough and Wagner (1967) completed a study on queue discharging behavior using field data collected from the Los Angeles metropolitan area. Departure headways collected from the field are also shown in Table 3.1.

Carstens (1971) studied starting delays and headways with manual counts, stop watches, and time-lapse photographs in Ames, Iowa. King and Wilkinson (1976) used a manual input method to study the relative effectiveness of various signal configurations and lens sizes in dissipating queues in Brookline, Massachusetts; San Francisco; Sacramento; and Huntington, New York. Lee and Chen (1986) measured departure headways by using video camera equipment in a small city, Lawrence, Kansas. Lu (1984) used a time recorder and stop watches to collect left-turn departure headways at an unprotected and a protected signalized intersection in Austin, Texas.

Moussavi and Tarawneh (1990) collected departure headways by using a laptop microcomputer in six cities in the state of Nebraska. According to their data analysis, the departure headways for different queue positions in large cities are smaller than those for smaller cities. The results of the Chi-Square test indicated that the departure headways for each queue position follow the normal distribution.

One of the most recent studies on departure headways was made by Efstathiadis (1992). He measured departure headways using a storage stop watch device for a number of signalized intersection approaches in Austin, Texas. The results of a normality test showed that departure headways were, in most cases, normally distributed. The mean value of headway generally decreased from front to rear of the queue. All the average headway data obtained from the above studies are summarized in Table 3.1.

Table 3.1 shows some interesting trends in the average departure headway as time has progressed since the late 1940s. The table shows that the average departure headway increased until the early 1970's, and thereafter, it indicates a steady decrease in the average departure headway until now. This might be a direct result of a decrease in average car size, improved vehicle design, improved traffic flow control strategies, an improvement in the intersection geometry, and improved pavements. Furthermore, more aggressive driver behavior and the advent of automatic transmissions might have contributed to more rapid response to the green signal by today's driver. For example, the recent studies, by Efstathiadis (1992), Moussavi and

Tarawneh (1990), and Lu (1984), showed the similar results that departure headways measured in all their studies were lower than in almost all other earlier studies.

Most past studies were based on limited data points and date back 10 to over 40 years. Since traffic and vehicle characteristics have changed considerably over time, for current applications, results of those studies might not be appropriate.

For reasons discussed above, average departure headways for different queue positions which are presented in Efstathiadis' study were used for this study. His study is the most recent one and the values are assumed to represent current traffic, vehicle, and driver characteristics.

Starting delay is closely related to the selection of the screen line where departure headways are measured. As the distance between the screen line and the first vehicle in the queue increases, starting delay increases.

3.6 GREEN TIME AND AVERAGE TIME BETWEEN SUCCESSIVE VEHICLE STARTS

GREEN TIME

Efstathiadis (1992) developed an equation which relates the time required for a number of stopped vehicles at a signalized intersection to pass the reference line with start-up lost time, L, and time headway, H, between vehicles. The equation is as follows:

$$G = L + H * n \quad \text{for } n \geq a$$

$$= 1.34 + 1.82 * n \quad \text{for } n \geq 4 \qquad (3.4)$$

where

G = time needed for n vehicles (front end) in a single-line stopped queue to cross a designated reference line at a signalized intersection after the signal indication changes to green

L = start-up lost time

H = average time headway between successive vehicles

n = number of vehicles that cross the reference line.

a = number of vehicles that contribute to the start-up lost time

The overall average start-up lost time of 1.34 seconds can be attributed to the first four vehicles and the average headway, H, after the fourth vehicle was 1.82 seconds. Equation (3.4) was used to calculate the green time required to clear queued vehicles.

Average time between successive vehicle starts

According to the George and Heroy study (1966), the average time between successive vehicle starts was approximately 1.4 seconds. However, a decrease in average car size, more aggressive driver behavior, and the advent of automatic transmissions have contributed to more rapid response in starting from a stopped position. Therefore, the average time between successive vehicle starts is modified to 1.1 seconds to represent current traffic conditions, appropriately. This value is used to calculate the time a vehicle stored in a queue starts moving forward after the beginning of the green signal.

3.7 SUMMARY

Basic terminology such as input, output, queue length, and queue spillback has been defined in this chapter. Methods for restricting queue length, approaches to the problem, and conceptual analysis of oversaturated conditions were discussed. As important parameters underlying the traffic simulation model, the departure headway and average time between successive vehicle starts were described. The next chapter will describe the development of the traffic simulation model for traffic signal timing in oversaturated conditions.

CHAPTER 4 DEVELOPING A TRAFFIC SIMULATION MODEL

4.1 INTRODUCTION

As discussed in Chapter 2, the only traffic simulation models for arterial networks, which have a feature to take into account the effect of queue spillback are TRANSYT-7F and TRAF-NETSIM. However, since the former does not explicitly deal with queue spillback and the latter lacks the capability of signal timing optimization, both models are not suitable for use in this study.

An analytical tool is needed to analyze and predict arterial network traffic performance. The model should be designed to study an oversaturated time period over an arterial including, at least, three intersections. Because of the sophisticated nature of the problem, that will mainly deal with queue spillback in oversaturated traffic conditions, a stochastic microscopic-type model is envisioned.

A number of analytical techniques are expected to be required for this problem, including capacity analysis, traffic flow models, and queueing analysis. The use of a computer simulation model is best to integrate these various techniques into one analytical framework.

This chapter discusses fundamentals of traffic simulation and how these are implemented in the traffic simulation model. The details of the simulation model are described in the following order: basic assumptions and logic of the model, the structure of the traffic simulation model, a brief description of each subroutine and program details of several important subroutines, a description of an example simulation run, the simulation input data, and the calibration and validation of the model.

4.2 THE FUNDAMENTALS OF TRAFFIC SIMULATION

This section discusses some fundamentals of traffic simulation and how they are implemented in this model.

4.2.1 Random Number Generation

In this traffic simulation model, a random number generating routine was applied for introducing "real world" variability to parameters, which will be discussed in the next chapter. To overcome the disadvantage of system-supplied random number generators, an improved random number generator from Press et al. (1986) was used. The improved random number generator

has an indefinite period (for all practical purposes) and should have no sensible sequential correlations.

4.2.2 Scanning and Updating

A digital computer cannot examine simulated vehicles simultaneously or continuously. It is, thus, necessary to define a scanning procedure by which each vehicle unit is examined. The two basic procedures that might be adopted are referred to as *time-based* and *event-based* scanning. In a time-based system, the simulated traffic is scanned and updated after each predetermined increment of simulated time (typically 1 second). An event-based system scans to determine the time of the next event, and this becomes the time increment for the simulation updating. Due to the nature of oversaturated traffic operations, the event-based scanning procedure was chosen as more efficient but was supplemented, for some sub-procedures, by time-based scanning.

4.2.3 Bookkeeping

There are two different systems for the structuring of storage arrays in traffic simulation bookkeeping. One system divides each traffic lane into a finite number of spaces, and allocates computational array elements to each space. The position of a vehicle is then represented by the location within the array, and position updating consists of transferring the information indicating each vehicle to the appropriate array elements. The other system uses vehicle-based arrays, which was chosen for this simulation model. Vehicle position, as well as other vehicle specific information, is stored in array elements permanently allocated to each vehicle. There are various options within each of these systems, and some features may be combined. As the scanning and information retrieval procedures will depend on the storage method, the bookkeeping system must be considered as an integral simulation logic component.

4.2.4 Assigning Driver/Vehicle Characteristics

Each simulated driver and vehicle unit must be assigned a number of parameters which will be used to determine appropriate responses and behavior within the simulation. The employed method of assigning characteristics for this simulation model is to regard all such parameters as being random, and to assign unit characteristics randomly chosen from specified probability distributions describing parameters such as start-up lost time and time spacing of vehicles departing from a signalized intersection.

4.2.5 The Entry Process

A stream of simulated vehicles must be generated with an appropriate arrival process at each end of the simulated arterial section. *Warm-up intersections* are used for generating a platooned arrival process at the stream entries to a simulated arterial. As shown in Figure 4.1, hypothetical intersections, referred to as 'warm-up' intersections, were added at each end of the simulated arterial. Vehicles are generated according to a uniform arrival process at the entry to the warm-up intersection. Infinite storage spaces upstream of each warm-up intersection are assumed.

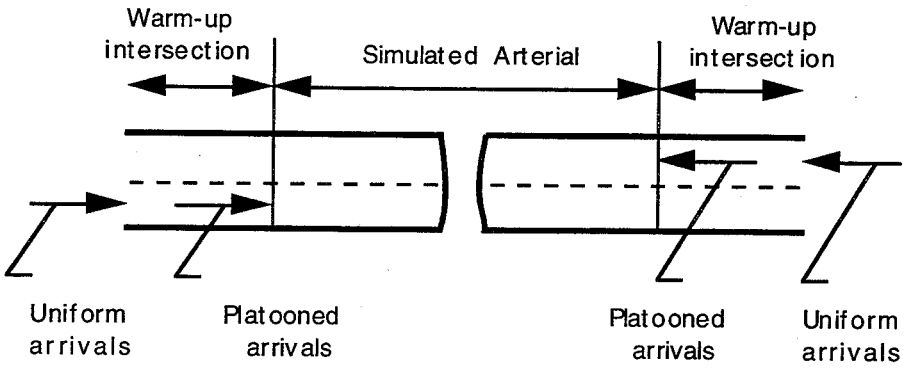


Figure 4.1: Use of warm-up intersections to generate a platooned arrival process

4.2.6 Warm-Up Time

Since the simulation model assumes that the initial state of an arterial system is empty, a certain amount of time is required to fill the arterial system after simulation run start. Considering a three intersection saturated arterial network, movements into and out of intersections, can be described in terms of input and output. When input is equal to output, no queue spillback occurs. Turning movements into an arterial occuring during the arterial red time can easily upset this relationship and therefore, are key factors that cause queue spillback.

A "leftover" vehicle is defined as one that cannot move forward into the downstream intersection during the first green interval after its initial stop. The number of "leftover" vehicles in a link is a function of the number of turn-in vehicles. Queue spillback occurs when the number of vehicles, which include "leftover" vehicles and arriving vehicles, exceed the link capacity. Assuming no variability in parameters, this kind of queue spillback will occur continuously throughout the simulation after the first queue spillback. Therefore, the warm-up period l can be chosen by

$$l = (\text{Dist} / X) / \min(\text{nlt}, \text{nrt}) \qquad (4.1)$$

where

l = warm-up period (number of cycles)

Dist = link length (ft)

X = vehicle space headway (ft/veh)

nlt = number of (protected) left-turn vehicles per cycle

nrt = number of right-turn vehicles per cycle

This warm-up period is approximately equal to the time required for a link to be filled with "leftover" vehicles.

Example: Dist = 600 ft. X = 20 ft/veh

nlt = 6 veh/cyclenrt = 3 veh/cycle

l = (600/20) / min (6, 3) = 10 signal cycles

4.3 TRAFFIC SIMULATION MODEL

This section describes in detail how the traffic simulation model operates. Basic assumptions and logic used in processing vehicles are discussed. Most subroutines are briefly described and program details of several important subroutines are presented. A detailed description of an example simulation run is provided.

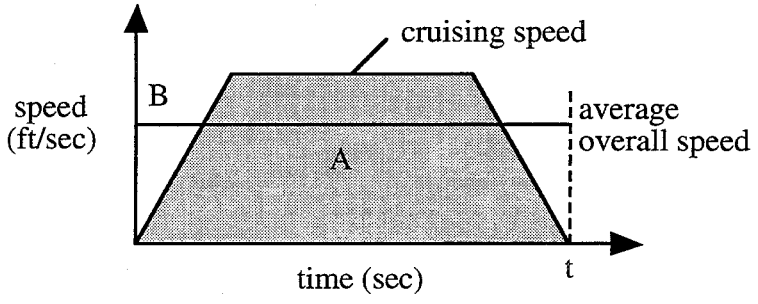
4.3.1 Basic Assumptions and Logic of the Model

This subsection discusses basic assumptions and logic used in processing vehicles from their point of arrival through the intersection. Listed below are basic assumptions and logic not previously discussed in this study:

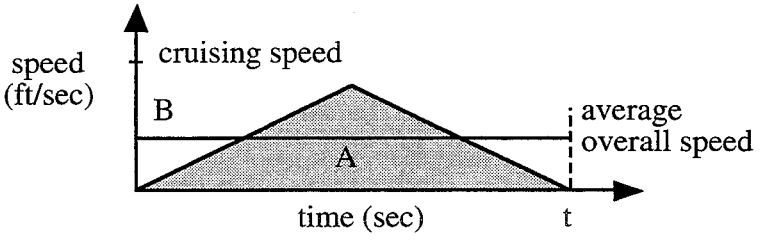
- This model is designed specifically for oversaturated arterial traffic conditions. Traffic demand at the entry of the arterial street is greater than the capacity. The demand is given in terms of arriving headways at the first (warm-up) intersection. The downstream intersection traffic demand is controlled by the upstream intersection green time. The default value for the arterial arriving headway is 2.0 seconds per vehicle, that is equivalent to 1800 vehicles per hour per lane (vphpl). If the saturation flow is 1800 vphpl, the entry of the arterial street will always be oversaturated unless it has all available green time.
- Within the cases specifically simulated, traffic demand at the entry of cross streets is selected so that the degree of cross street saturation can be determined by the green ratio. The default value for the cross street arriving headway is 7.2 seconds per vehicle, that is equivalent to 500 vphpl. Given this volume, if cross streets have less than 27.7 % (=

500/1800) available green time, the entries of cross streets will be oversaturated, otherwise, they will be undersaturated.

• To simulate vehicle movements in the traffic simulation model, appropriate acceleration and deceleration rates are necessary. If the appropriate empirical data for those rates are available, to apply the data to each vehicle would be the best way of simulating vehicle movements. However, since such empirical data are usually unavailable, uniform or linear acceleration and deceleration rates are often assumed. As shown in Figure 4.2, if the area A (shaded) is equal to the area below line B, distances traveled by a vehicle accelerating, cruising (if necessary), and decelerating for time t and traveling at the average overall speed for time t will be equivalent. Thus, applying appropriate average overall speed obtained from the field may produce more accurate results in calculating the travel time than applying assumed uniform or linear acceleration and deceleration rates to each vehicle. Furthermore, applying appropriate average overall speed enables the model to simulate vehicle movements faster, which is important for a network simulation model. Therefore, the average overall speed concept was used in this study instead of applying to each vehicle uniform or linear acceleration and deceleration rates.



a) case 1: vehicles reach a cruising speed before decelerating



b) case 2: vehicles never reach a cruising speed before decelerating

Figure 4.2: Average overall speed

• Two of the most important events to be tracked through the simulation process are the arrival and departure times of each vehicle simulated. Each vehicle's movement in the model is determined mainly by departure headway and average overall speed. The departure time is largely determined by departure headway. Each vehicle moves at its own average overall speed in the link. The arrival time is largely determined by departure headway and average overall speed. Therefore, for this simulation model, data for departure headway and average overall speeds are essential.

• Since event-based scanning is adopted, it is not necessary to identify what happens between the two consecutive events. Once a vehicle departs the intersection stop line (reference line), the next event to be identified is its arrival time in the downstream link. Thus, for example, it is not necessary to identify acceleration and deceleration rates or cruising speed of the vehicle at a point between the departure and arrival times as long as average overall speed is applied correctly. A vehicle can depart the reference line and move at its own average overall speed during the green interval whenever the downstream clear space is available. If a queue spillback occurs in the downstream link, upstream vehicles must wait for downstream clear space(s) before advancing.

• When a vehicle departs the reference line, an average overall speed based on the downstream clear space is assigned to it. Then, it moves into the downstream intersection at the average overall speed. Therefore, each vehicle simulated has its own space and time headway relationships with the leading and following vehicles when they are moving, which indicates that a platoon dispersion model is employed in an implicit way. Since event-based scanning is adopted, however, it is not necessary to identify this phenomenon as an event.

• Overtaking (passing) is not allowed due to oversaturated conditions in which no space is available.

• Lane changing is not allowed in the model due to oversaturated conditions.

• Right-turn (left-turn) vehicles enter the rightmost (leftmost) lane.

• There are no bus stops or pedestrians.

• There are passenger vehicles only, that is, no heavy vehicles.

• A yellow clearance interval is included in each green interval.

• There are no grades.

The following logic and assumptions are previously explained and summarized here:

• Discharge from the head of a queue is based on results of Efstathiadis' study (1992).

• Based on the George and Heroy study (1966), average times of successive vehicle starts are determined as follows;

$$sut(n) = 1.1 * n \quad \text{for } n \geq 4 \tag{4.2}$$

where

sut (n) = time required for the nth vehicle in the queue to start moving after the signal turns green

n = number of vehicles in the queue.

- Given a green signal and a filled downstream link, one storage space in the intersection is always provided to the advancing traffic stream. A queue spillback occurs when upstream vehicles with a green signal, move into the intersection joining the stopped queue. If a queue spillback occurs, the vehicle caught in an intersection blocks the intersection until it is cleared. Queue spillback is decribed in detail in the next chapter.
- A simulation begins after the warm-up time, which is set to a specified number of traffic signal cycles.
- The duration of a simulation job is specified by a number of vehicles to be passed through the simulated network.

4.3.2 Structure of the Traffic Simulation Model

The traffic simulation model consists of a MAIN routine and many subroutines. The MAIN routine includes three major subroutines; RIGHTL, LEFTL, and CROSST. Two subroutines, RIGHTL and LEFTL, are called to process the arterial traffic. For two-way operation, they are used twice in the MAIN routine. For each cross street the subroutine named CROSST is called to process cross street traffic. The conceptual flow of the simulation program execution is shown in Figure 4.3.

The MAIN program reads the simulation input data which includes the arterial geometry, traffic signal timing, traffic demand, and simulation period. With the initialization of the array and the necessary calculation to transform the input data, the simulation begins at time zero. At the start, the arterial system is assumed to have empty links with no vehicles. The RIGHTL subroutine is called to generate vehicles and process them in the right arterial lane. When any green of the arterial ends in the RIGHTL subroutine, the LEFTL subroutine is called to generate and process vehicles in the left arterial lane.

The first CROSST subroutine is called to generate and process vehicles in each lane of the first of three cross streets after arterial green ends during LEFTL subroutine execution. In the CROSST subroutine vehicles in the left lane are processed first followed by right lane vehicles. For the second and third cross streets, the second and third CROSST subroutines are executed in the same way as the first. The MAIN routine is executed until the specified number of vehicles

is simulated (including arterial and cross street vehicles after the warm-up time ends). The warm-up time is set to a specified number of traffic signal cycles.

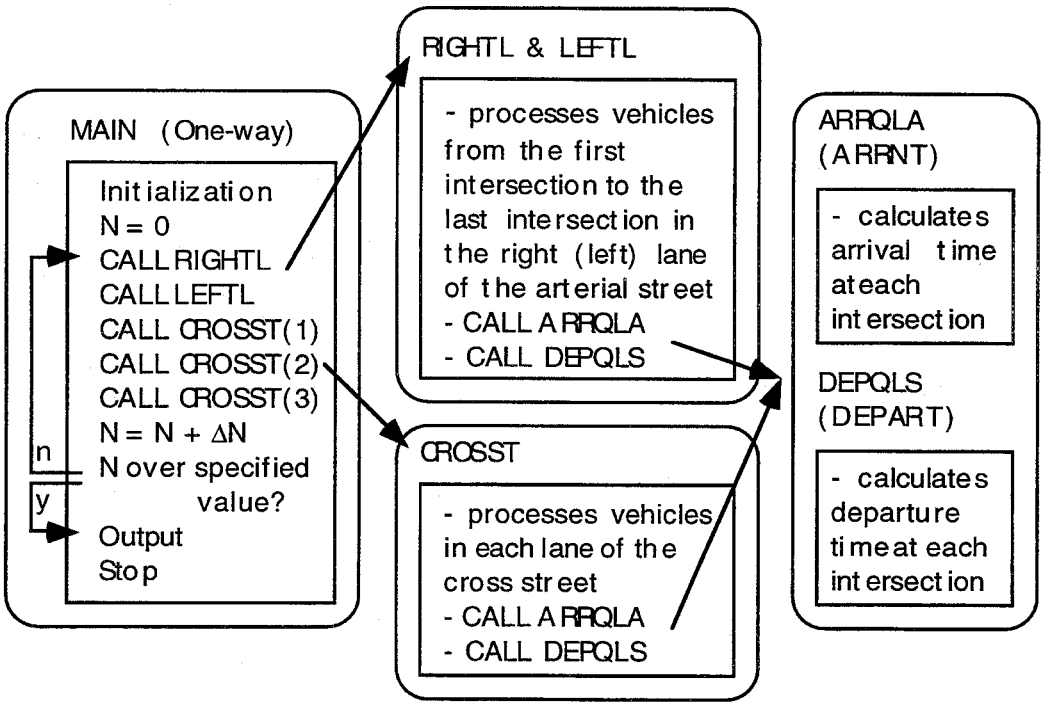


Figure 4.3: Conceptual flow of the simulation program execution

Throughout the simulation, intersection arrivals and departures are major events. An arrival time at an intersection is defined differently depending on queue presence. If there is no queue in the link, the time a vehicle reaches the intersection stop line (reference line) is the arrival time, otherwise it is the time of queue joining. A departure time is the time a vehicle passes the intersection stop line. If there is no queue in the link, the intersection arrival and departure times are equal.

The three major vehicle processing subroutines call the arrival and departure subroutines at each intersection. The arrival subroutines (ARRQLA and ARRNT) calculate the arrival time of each simulated vehicle at each intersection. ARRQLA deals with the case of vehicles turning on to the arterial from cross streets. In the absence of cross street vehicles turning on to the arterial, the subroutine named ARRNT is used. The departure subroutines (DEPQLS and DEPART) calculate departure times of each vehicle at each intersection. DEPQLS includes the option to delay vehicle departure when a queue spillback occurs in the downstream link. When the downstream link is full, only one vehicle from the upstream intersection stop line is allowed to join

the queued vehicles in the downstream link, creating a queue spillback. When this occurs the following vehicles must wait until the vehicle caught in the intersection starts moving forward. If this queue spillback is not cleared within the current green, it will block cross street movements until it is cleared. The departure headway, which is used for departure times of queued vehicles, is based on the Efstathiadis' study (1992).

4.3.3 Subroutines of the Traffic Simulation Model

The program was envisioned as including a main executive program calling other subroutines, which would do the actual work. A brief description of each subroutine is listed in Table 4.1. As each subroutine completes its task, control returns to the main program, which in turn calls the next subroutine.

TABLE 4.1: DESCRIPTION OF SUBROUTINES

SUBROUTINE NAME	DESCRIPTION
ARRNT	calculates arrival time of a vehicle when there are no turn-in movements.
ARRQLA	calculates arrival time of a vehicle when there are turn-in movements.
CNNQS	counts numbers of queue spillbacks.
CROSST	processes vehicles in the cross street.
CSPEED	calculates average overall speed
CTIME	calculates cross street traffic signal timing.
DEPART	calculates vehicle departure times.
DEPQLS	calculates vehicle departure times and includes an option that delays departure when a queue spillback occurs.
DPHDWY	stores departure headway and the time queued vehicles start moving forward when the signal turns green.
FINDEV	calculates when the (m-(iveh1+1))th vehicle in the downstream link starts moving forward (iveh1 = link capacity in number of vehicles).
INPUT	stores data in appropriate format and locations for later use by other subroutines.
INIT	, INIT1, and INIT2 initialize all computational arrays.

Table 4.1 continued on next page

Table 4.1 continued from previous page

SUBROUTINE NAME	DESCRIPTION
LEFTL	processes vehicles in the left arterial lane.
NEXTTI	finds the next available turn-in vehicle.
NORDEV	generates normal random deviates with a mean m and a standard deviation s.
NORMAL	generates normally distributed deviates with zero mean and unit variance.
NOQUE	counts the number of queued vehicles.
NOVSIM	counts the number of vehicles simulated.
NOVWUP	counts the number of warm-up vehicles.
PIKSRT	sorts n values in ascending order.
PRN	prints arrival and departure times at each intersection.
PRNQS	calculates and prints number of queue spillbacks caused by through traffic.
PRNQST	calculates and prints number of queue spillbacks caused by turning-in vehicles.
PRNOP1	prints summarized outputs (number of queue spillbacks per cycle).
PRNOP2	prints summarized outputs (number of vehicles simulated).
RANNUM	generates uniform random deviates between 0.0 and 1.0.
RANSPD	generates random average overall speeds.
RIGHTL	processes vehicles in the right arterial lane.
SIGADJ	adjusts traffic signal timing obtained by SIGNAL.
SIGNAL	calculates arterial traffic signal timing.
UPDATE	updates signal timing.

Execution steps of several major subroutines, which perform the important program work, are summarized in Tables 4.2 - 4.5.

TABLE 4.2: EXECUTION STEPS OF THE SUBROUTINE ARRQLA

Steps	Description
Step 1	Count the number of queued vehicles in the downstream link and calculate the time the last vehicle in the queue starts moving.
Step 2	Calculate or generate (in case with variability) the average overall speed and the distance to be traveled based on the results of step 1.
Step 3	Calculate arrival time of the mth vehicle (arr(m)) based on the departure time of the vehicle from the upstream intersection (depp(m)), the average overall speed (speed), and distance to be traveled (dist): arr(m) = depp(m) + dist / speed
Step 4	During the cross street green, calculate arrival time of the nth turn-in vehicle from cross street (arrr(n)) by applying step 3.
Step 5	If arr(m-1) < arrr(n) < arr(m), then: arr(m+1) = arr(m) arr(m) = arrr(n)
Step 6	If a queue spillback occurs in the downstream link before a vehicle turns onto the arterial from the cross street, then delay processing the turn-in movements until the vehicle caught in the intersection starts moving.

TABLE 4.3: EXECUTION STEPS OF THE SUBROUTINE CROSST

Steps	Description
Step 1	If the left lane routine is not called (if klt = 0), then go to step 10.
Step 2	Update the number of vehicles simulated.
Step 3	If the second intersection green is updated, then go to step 7.
Step 4	Generate vehicles at the first/warm-up intersection with the specified arrival rate.
Step 5	Calculate the departure time from the first intersection.
Step 6	Calculate the arrival and departure times at the second intersection. If dep2 exceeds the end of green, then update signal timing and go to step 10.
Step 7	Calculate the arrival and departure times at the third intersection.

Table 4.3 continued on next page

Table 4.3 continued from previous page

Steps	Description
Step 8	Print the arrival and departure times at each intersection.
Step 9	Go to step 2.
Step 10	If the right lane routine is not called (if krt = 0), then exit.
Step 11	Update the number of vehicles simulated.
Step 12	If the second intersection green is updated, then go to step 16.
Step 13	Generate vehicles at the first/warm-up intersection with the specified arrival rate.
Step 14	Calculate the departure time from the first intersection.
Step 15	Calculate the arrival and departure times at the second intersection. If dep2 exceeds the end of green, then update signal timing and exit.
Step 16	Calculate the arrival and departure times at the third intersection.
Step 17	Print the arrival and departure times at each intersection.
Step 18	Go to step 11.

TABLE 4.4: EXECUTION STEPS OF THE SUBROUTINE DEPQLS

Steps	Description
Step 1	Update k (kth vehicle in the queue when the signal turns green).
Step 2	Count the number of queued vehicles in the downstream link .
Step 3	Calculate departure time (dep) based on the k and departure headway (z) and beginning of the green (ig): dep(m) = z(k) + ig
Step 4	If a queue spillback occurs in the downstream link, then delay the departure of the first vehicle in the upstream link until the vehicle caught in the intersection starts moving forward. Calculate the departure delay and add it to the departure times of the following vehicles until the green ends: dep(m) = z(k) + ig + delay
Step 5	If dep(m) exceeds the end of green, then update signal timing and delay departures of vehicles until the next available green begins.
Step 6	If there is no queue in the downstream link, then dep(m) = arr(m).

TABLE 4.5: EXECUTION STEPS OF THE SUBROUTINES LEFTL AND RIGHTL

Steps	Description
Step 1	Examine which green interval is updated. If the first intersection green is updated, then go to step 5. If the second intersection green is updated, then go to step 6. If the third intersection green is updated, then go to step 7.
Step 2	Update the number of vehicles simulated.
Step 3	Generate vehicles at the first/warm-up intersection with the specified arrival rate.
Step 4	Calculate the departure time from the first intersection (dep1). If dep1 exceeds the end of green, then update the first intersection signal timing and exit.
Step 5	Calculate the arrival and departure times (arr2 and dep2) at the second intersection. If dep2 exceeds the end of green, then update the second intersection signal timing and exit.
Step 6	Calculate the arrival and departure times (arr3 and dep3) at the third intersection. If dep3 exceeds the end of green, then update the third intersection signal timing and exit.
Step 7	Print the arrival and departure times at each intersection.
Step 8	Go to step 1.

4.3.4 Description of an Example Simulation Run

For a better understanding of the model, a detailed description of an example simulation run is provided. The example selected includes a one-way two-lane arterial street and three one-way two-lane cross streets with a short link length (200 feet) and a short cycle length (60 seconds). The arterial configuration is shown in Figure 5.8. For offsets 2 and 3, values of zero and 45 seconds are used and a common 60 second cycle length with 40 and 20 seconds arterial and cross street green interval, respectively, are used. Two vehicles per cycle turn from the arterial to cross streets. One vehicle per cycle turns from cross streets to arterial. Ten vehicles can be stored in a 200 foot long link and up to twenty-one vehicles can be released during a 40 second arterial green interval. Up to ten vehicles can be released during a 20 second cross street green interval. The widths of a cross street and arterial are 40 and 60 feet, respectively. Since

both arterial lanes are similar in vehicle processing activities, except turning movements, (in and out), the right lane of the arterial street is described.

Table 4.6 shows a simulation run output example. The table includes a partial output of the arterial street right lane (first 30 vehicles). Vehicles with zero values for AT (arrival time) and DT (departure time) at the third intersection turn on to the second cross street at the second intersection. The vehicles with zero values for AT and DT at the first intersection, turn onto the arterial street from the first intersection. Since the first intersection is used as a warm-up intersection, by definition, no queue spillback occurs there.

A simulation begins with an arterial street red interval (cross streets green) at t = 0. The green intervals for the first and second intersections begin at t = 20 (20 seconds after the beginning of the simulation). The green interval for the third intersection begins at t = 5 [20 (beginning of the green for the first intersection) + 45 (offset 3) = 65 - 60 (cycle length) = 5]. The first vehicle in the queue (m = 1) departs the first intersection at t = 22.0 (green begins at t = 20) and arrives at the second intersection at t = 29.8. The travel distance is 240 feet (link length 200 feet plus cross street width 40 feet), and the average overall speed is 30.9 ft./sec; therefore, it takes 7.8 seconds for the first vehicle to arrive at the second intersection. The first vehicle passes the second intersection without stopping because it arrives at the intersection during the green interval and there is no queue. Fourteen vehicles ($2 \leq m \leq 15$) also pass the second intersection without stopping for the same reason. The fifth vehicle arrives at the third intersection after the third intersection signal turns red. Ten vehicles ($6 \leq m \leq 15$) arrive at the third intersection during the red interval. Therefore, a queue spillback occurs when the fifteenth vehicle (m = 15) joins the queue developed at the third intersection. The sixteenth vehicle has to wait at the stop line of the second intersection until the fifteenth vehicle, caught in the second intersection, moves forward at t = 65 + 1.1 * 11 = 77.1 (the 2nd green of the 3rd intersection begins at t = 65). However, the traffic signal of the second intersection turns red (at t = 60) before the fifteenth vehicle starts moving forward. The fifteenth vehicle blocks the right traffic lane of the second cross street during almost the entire cross street green [17.1 ($60 < t < 77.1$) seconds out of a 20 second interval ($60 < t < 80$)]. The remaining five vehicles ($17 \leq m \leq 21$) arrive at the second intersection during the red interval. The first vehicle that passed the second intersection without stopping arrives at the third intersection at t = 37.6 and passes the third intersection without stopping. Three vehicles ($2 \leq m \leq 4$) also pass the intersection without stopping and two of them turn on to the second cross street.

TABLE 4.6: OUTPUT EXAMPLE OF A SIMULATION RUN

S[1]	turn		1st intersection		2nd intersection			3rd intersection		
m	in[2]	out[3]	AT[4]	DT[5]	AT	DT	NQ2[6]	AT	DT	NQ3[7]
1	0	0	2.0	22.0	29.8	29.8	0	37.6	37.6	0
2	0	1	4.0	24.5	32.3	32.3	0	.0	.0	0
3	0	1	6.0	26.6	34.4	34.4	0	42.2	42.2	0
4	0	2	8.0	28.6	36.4	36.4	0	.0	.0	0
5	0	2	10.0	30.4	38.2	38.2	0	46.0	67.0	0
6	0	2	12.0	32.3	40.0	40.0	0	49.8	69.5	1
7	0	2	14.0	34.1	41.8	41.8	0	51.1	71.6	2
8	0	2	16.0	35.9	43.7	43.7	0	52.4	73.6	3
9	0	2	18.0	37.7	45.5	45.5	0	53.6	75.4	4
10	0	2	20.0	39.5	47.3	47.3	0	54.7	77.3	5
11	0	2	22.0	41.4	49.1	49.1	0	55.8	79.1	6
12	0	2	24.0	43.2	50.9	50.9	0	56.8	80.9	7
13	0	2	26.0	45.0	52.8	52.8	0	57.7	82.7	8
14	0	2	28.0	46.8	54.6	54.6	0	58.5	84.5	9
15	0	2	30.0	48.6	56.4	56.4	0	59.2	86.4	10
16	0	2	32.0	50.5	58.2	82.0	0	84.8	88.2	11
17	0	3	34.0	52.3	62.1	84.5	1	.0	.0	2
18	0	3	36.0	54.1	63.4	86.6	2	95.9	95.9	2
19	0	4	38.0	55.9	64.6	88.6	3	.0	.0	0
20	0	4	40.0	57.7	65.8	90.4	4	98.2	98.2	0
21	0	4	42.0	59.6	67.0	92.3	5	100.0	100.0	0
22	1	4	.0	.0	79.4	94.1	6	101.8	101.8	0
23	1	4	44.0	82.0	87.9	95.9	7	103.7	103.7	0
24	1	4	46.0	84.5	97.2	97.7	5	105.5	127.0	0
25	1	4	48.0	86.6	97.7	99.5	1	109.3	129.5	1
26	1	4	50.0	88.6	98.4	101.4	1	110.6	131.6	2
27	1	4	52.0	90.4	99.7	103.2	2	111.9	133.6	3
28	1	4	54.0	92.3	101.5	105.0	2	113.1	135.4	4
29	1	4	56.0	94.1	103.4	106.8	2	114.2	137.3	5
30	1	4	58.0	95.9	105.2	108.6	2	115.3	139.1	6

1) S = total number of vehicles simulated

2) in = number of vehicles turning from cross street to arterial street

3) out = number of vehicles turning from arterial street to cross street

4) AT = arrival time in seconds

5) DT = departure time in seconds

6) NQ2 = number of vehicles stored in the second intersection after the (m-1)th vehicle arrives at the intersection

7) NQ3 = number of vehicles stored in the third intersection after the (m-1)th vehicle arrives at the intersection

4.3.5 Input

The traffic simulation program consists of two separate versions, which permits one-way or two-way street operation. The one-way operation version deals with a two-lane one-way arterial and three two-lane one-way cross streets. The two-way operation version treats a two-lane two-way arterial (total four lanes) and three two-lane one-way cross streets. Since both versions use the same input file, the one-way operation version is chosen for input data descriptions.

The program can be executed in two different modes, a single run and multiple run modes. In the single run mode, the program runs a single case and produces detailed output including arrival and departure times of each vehicle at each intersection. In the multiple run mode, the program runs multiple cases (number of runs is determined by input data) in order to find an optimal case with respect to one of three input parameters, offsets, green split, or cycle length; and produces summarized outputs like network crossing times. In each mode (single or multiple run), the program can be executed either with or without parameter variability. The simulation input data are summarized in Table 4.7.

The first data set read by the MAIN program consists of three binary indicator variables; the first variable **mult** specifies whether a simulation run includes a single case or multiple cases; the second variable **jcyc** specifies whether a simulation run finds an optimal cycle length or not; the last variable **jvar** indicates the type of parameters, uniformity or variability. If **ioff1** is not equal to **ioff2** or **joff1** is not equal to **joff2**, then offset 1 or offset 2 is optimized. If **jcyc** is zero and **incg** is greater than zero, then the green split is optimized. Two or three (if **ipt** is greater than zero) phase operation is available in the two-way operation version. Average overall speed data are included in the arrival subroutines and the RSPEED subroutine.

TABLE 4.7: SIMULATION INPUT DATA

Row	Column	Variable	Default	Definition [Unit]
1	1 - 6	mult	0	number of cases: 0; a single case 1; multiple cases (for optimization)
1	7 - 12	jcyc	0	cycle length optimization: 0; no 1; yes (optimization)

Table 4.7 continued on next page

Table 4.7 continued from previous page

Row	Column	Variable	Default	Definition [Unit]
1	13 - 18	jvar	0	variability of parameters: 0; no (uniformity) 1; yes (variability)
2	1 - 6	nvol	2000	number of vehicles to be simulated
2	7 - 12	ncwu	10	number of cycles for warm-up time
2	13 - 18	tdc	7.2	cross street traffic demand (arriving headway) [seconds]
2	19 - 24	nac	2	number of turning vehicles per cycle from arterial to cross street
2	25 - 30	nca	1	number of turning vehicles per cycle from cross street to arterial
3	1 - 6	dist(1)	200	link length between the 1st and 2nd intersections [feet]
3	7 - 12	dist(2)	200	link length between the 2nd and 3rd intersections [feet]
3	13 - 18	wida	60	width of arterial [feet]
3	19 - 24	widc	40	width of cross streets [feet]
3	25 - 30	avsh	20	average vehicle space headway [feet]
4	1 - 6	icl	60	cycle length
4	7 - 12	lgi	40	arterial green interval [seconds]
4	13 - 18	idxn	0	difference between maximum and minimum arterial greens
4	19 - 24	incg	10	increment of idxn (for optimization)
4	25 - 30	ipt	0	protected left turn phase duration (only for two-way operation) [seconds]
4	31 - 36	jdq	7	minimum delay to determine a queue spillback [seconds]
5	1 - 6	noff(1)	0	offset 1 [seconds]
5	7 - 12	ioff1	0	minimum offset 2 (0, for optimization)
5	13 - 18	ioff2	C	maximum offset 2 (C, for optimization)

Table 4.7 continued on next page

Table 4.7 continued from previous page

Row	Column	Variable	Default	Definition [Unit]
5	19 - 24	joff1	0	minimum offset 3 (0, for optimization)
5	25 - 30	joff2	C	maximum offset 3 (C, for optimization)
5	31 - 36	incre	15	increment of offsets (for optimization)

4.3.6 Validation and Calibration

One means of validating traffic simulation models is comparing simulated and field observed measures of effectiveness. For simulation of a single intersection approach or even an entire intersection, this is feasible. Simulation model output quantities can be appropriately compared to field measured versions of the same quantities and the model can be tuned to reproduce field measured quantities. However, even though field and simulated measures of effectiveness agree, such apparent agreement does not guarantee all simulation model components are properly emulating the real world. Significant positive and negative errors in complementary model components can compensate for each other causing total output quantities to appear correct. Therefore, calibration of model component parts is perhaps more important than checking overall output measures of effectiveness.

Network traffic simulation model validation, in terms of overall output quantities, requires such large field data collection efforts that it is generally not feasible. Even if sufficient quantities of appropriate field network data can be obtained, calibration of components of the simulation system remains essential. Validation of the oversaturated arterial traffic simulation process has been done primarily through calibration of components. Overall model output measures of effectiveness have been checked for reasonableness and have been carefully examined for extreme or boundary condition appropriateness.

As explained in the subsection 5.3.1, the departure headway data of the Efstathiadis study (1992) were used. His data including 5915 field-observed data points were collected from five intersections in Austin, Texas. Most vehicles included in his study were passenger cars including mini vans and small trucks.

The definition and type of average overall speeds are described in subsection 5.3.3. The average overall speed data were also collected from an arterial in Austin, Texas. Type 1 and Type 2 speed data include 1061 and 1146 real-life observations, respectively. All the vehicles measured were passenger cars including mini vans and small trucks.

Another important component of the model is the average time between successive vehicle starts, which is used to calculate the time each vehicle in the queue starts moving. Based

on the George and Heroy study [They suggested 1.4 seconds for this value; 1966], the average time between successive vehicle starts is modified to 1.1 seconds to represent current traffic conditions, appropriately.

To see how well the model represents real life under the conditions tested, a detailed description of an example simulation run is provided in subsection 4.3.4. Two of the most important model incidents are the arrival and departure times of each vehicle simulated. The departure time, defined in subsection 5.3.3, is largely determined by the departure headway. The arrival time, also defined in the subsection 5.3.3, is largely determined by the departure time and average overall speed. Therefore, with the given traffic signal timing plans and arterial configurations, each vehicle's movement is mainly determined by the departure headway and average overall speed. In addition to these two components, the average time between successive vehicle starts is especially used to calculate the time a vehicle, caught in an intersection when a queue spillback occurs, starts moving. Field-observed real-life data were used for all the important model components and most important model incidents are mainly determined by those components. It can reasonably be said that the model represents a real situation.

4.4 SUMMARY

This chapter described how a traffic simulation model for traffic signal timing in oversaturated conditions was developed. Fundamentals of traffic simulation and how they are implemented in this model were discussed. The execution flow and the overall structure of the program were described. A brief description of each subroutine and program details (execution steps) of several important subroutines were summarized. Logic and assumptions used in processing vehicles were discussed. For a better understanding of the model, a partial output of an example simulation run was described. Finally, the simulation input data were explained in detail. With this model, experiment design, simulation experiments, and analyses of simulation experiment results are conducted in the next chapter.

CHAPTER 5 DESIGN AND ANALYSIS OF SIMULATION EXPERIMENTS

5.1 INTRODUCTION

In the previous chapter, a traffic simulation model for analyzing traffic signal timing in oversaturated conditions was described. This chapter discusses experimental design, simulation experiments, and analysis of results obtained through execution of the computer simulation model. To simplify the analysis, a basic form of an arterial network, a one-way street and two-phase signals, is considered first.

5.2 FORMULATION OF THE EXPERIMENTAL DESIGN

All applications of computer simulation models require an experimental design. In simulation, an experimental design provides a way of deciding before the runs are made which particular configurations to simulate so that the desired information can be obtained with the least effort.

In experimental-design terminology, the input parameters and structural assumptions composing a model are called *factors*, and the output performance measures are called *responses*. Factors can be either quantitative or qualitative. Quantitative factors naturally assume numerical values, while qualitative factors typically represent structural assumptions that are not naturally quantified.

For each system design to be simulated, decisions have to be made on such issues as initial conditions for the simulation runs, length of the warm-up period, simulation run duration, and the number of simulation runs.

5.2.1 Objectives of Experiments

The objective of the experimental program was the development of a methodology that would maximize the number of vehicles moved through an arterial network, which has an oversaturated traffic demand at the arterial street entry and a moderate traffic demand at the entry of cross streets, during a given time period. Additionally, the methodology must prevent queue spillback or minimize the occurrence of queue spillback if inevitable.

As discussed in the previous chapter, the traffic demand at the arterial street entry is set to 1800 vehicles per hour per lane (vphpl). If the saturation flow is 1800 vehicles per green hour per lane, the entry of the arterial street will always be oversaturated unless it has continuous signal green. The downstream intersection traffic demands are controlled by upstream

intersection green times. The traffic demand at the entry of cross streets is set to 500 vphpl. Thus, the entry of cross streets could be under or over saturated depending on the green ratio. If cross streets have less than 27.7 % (= 500/1800) available green time, the entries of cross streets will be oversaturated, otherwise, they will be under saturated.

Since the main focus is on the arterial street rather than cross streets, a moderate traffic demand (500 vphpl) for cross streets was chosen. If the same oversaturated traffic demand as in the arterial street were used for cross streets, then all available green time should be given to the direction favoring the largest number of lanes. This simple conclusion would maximize the number of arterial network vehicles processed.

5.2.2 Factors and Responses

As listed below, five input parameters and three structural assumptions were considered. A cycle length is broken into two components as design factors, which are an arterial green interval and a cross street green interval. Thus, neither cycle length nor green ratio is a factor. Offset i is the time difference between the initiation of green at intersection 1 and at intersection i. Link length is the distance between two adjacent intersections excluding intersection width. Since the effect of turning movements on queue spillback likelihood differs with link length, the number of turning vehicles itself is not an appropriate factor. Therefore, the link length percentage that would be occupied by a queue composed of all turning vehicles was used.

Input parameters (Quantitative factors)

- Arterial green interval: g
- Cross street green interval: gc
- Offset
- Link length: L
- Number of turning vehicles: N (as percentage of a link length)

Structural assumptions (Qualitative factors)

- Arterial operation: one-way or two-way
- Number of phases: 2-phase or 3-phase (only for two-way operation)
- Variability: no (uniformity) or yes (variability)

Tables 5.1 and 5.2 show the design matrix by qualitative and quantitative factors respectively.

TABLE 5.1: DESIGN MATRIX BY QUALITATIVE FACTORS

arterial operation	variability	2-phase	3-phase[1)]
one-way	no	I	NA
	yes	I'	NA
two-way	no	II	III
	yes	II'	III'

1) 3-phase operation includes a protected left-turn phase on the arterial and only applies to a two-way arterial.

TABLE 5.2: DESIGN MATRIX BY QUANTITATIVE FACTORS

factors	lower limit	upper limit	increment
g	20 sec.	90 sec.	10 sec.
gc	20 sec.	60 sec.	10 sec.
C	40 sec.	150 sec.	10 sec.
offset	0 sec.	C - 10	10 sec.
N (turning vehicles)	10 %	30 %	10 %
link length	200 ft.	600 ft.	200 ft.

Responses

Network crossing time, which is the time required for a given number of vehicles to move through the simulated network, was used as a response variable by which system efficiencies are compared. To help explain the simulation outputs, three auxiliary performance measures, none of which completely quantifies system efficiency, are used:

- Number of queue spillbacks per cycle (or queue spillback probability)
- Warm-up period (or number of vehicles simulated during this period)
- Number of vehicles simulated by each lane

5.2.3 Fractional Factorial Designs and Factor-Screening Strategies

As shown in Table 5.3, a total number of simulation runs can be calculated by multiplying the number of levels in each cell. The required 29,070 simulation runs make this design almost unmanageable even without considering qualitative factors and a protected left-turn phase length. If, however, these additional factors are considered, the total number of simulation runs would exceed 100,000. A fractional factorial design and a factor screening strategy are used to solve

this problem. A fractional factorial design is constructed by choosing a certain subset of all the possible design points and then running the simulation for only these chosen points. A factor screening strategy is formed by "screening out" some of the factors that are unimportant, fixing these factors at some reasonable value and omitting them from further consideration.

TABLE 5.3: NUMBER OF SIMULATION RUNS (BY A FACTORIAL DESIGN)

factors	lower limit	upper limit	increment	# of levels
g	20 sec.	90 sec.	10 sec.	
gc	20 sec.	60 sec.	10 sec.	3,230*
offset 2	0 sec.	C - 10	10 sec.	
offset 3	0 sec.	C - 10	10 sec.	
N (turning vehicles)	10 %	30 %	10 %	3
link length	200 ft.	600 ft.	200 ft.	3
total				29,070

* $4^2+5^2+2*6^2+2*7^2+3*8^2+3*9^2+4*10^2+4*11^2+4*12^2+3*13^2+2*14^2+15^2 = 3,230$

Table 5.4 shows a revised design matrix by fractional factorial designs and factor-screening strategies. For practical considerations, a minimum 40 seconds arterial green interval, or a minimum 60 seconds cycle length, is used. A cycle length shorter than 60 seconds is not appropriate for a saturated arterial network. As discussed before, the cross street traffic demand is selected so that the degree of cross street saturation can be determined by the green ratio. The default value for the cross street arriving headway is 7.2 seconds per vehicle (that is equivalent to a 500 vehicles per hour demand) and the cross street green interval is fixed at 20 seconds. This cross street green interval duration may cause under or over-saturation depending upon the selected traffic demand.

TABLE 5.4: REVISED DESIGN MATRIX

factors	lower limit	upper limit	increment	# of levels
g	40 sec.	85 sec.	15 sec.	
gc	20 sec.	20 sec.	0	$4^2+5^2+6^2+7^2$
offset 2	0 sec.	C - 15	15 sec.	= 126
offset 3	0 sec.	C - 15	15 sec.	
N (turning vehicles)	10 %	20 %	10 %	2
link length	200 ft.	600 ft.	400 ft.	2
total				504

As indicated in Table 5.4, the required number of simulation runs was further reduced through specifications for N, timing increments and link length. The maximum N value was changed from 30 to 20 percent and increments for arterial green, offset 2, and offset 3 were adjusted from 10 to 15 seconds. The number of link length levels was reduced to 2 by increasing the increment from 200 feet to 400 feet. The revised design matrix 1 shows considerably reduced design points, or 504, which is much more manageable.

5.3 INTRODUCING VARIABILITY TO PARAMETERS

Each simulated vehicle unit should be assigned a number of parameters which will be used to determine appropriate responses and behavior within the simulation. The employed method of introducing variability to the parameters for this simulation model is to regard all such parameters as being random, and to randomly assign each unit a value from a specified probability distribution. Three parameters considered for introducing variability are departure headway, vehicle space in the queue, and vehicle speed.

5.3.1 Departure Headway

The departure headway is the time between successive vehicles departing the stop line of a signalized intersection after the signal turns green. In the recent studies, Efstathiadis (1992) and Moussavi and Tarawneh (1990) found that the departure headways for different queue positions at signalized intersections follow the normal distribution. Therefore, using those study results, the departure headways are assumed normally distributed with a mean 1.82 and a standard deviation 0.43.

$$hdwy(n) = 1.82 \quad \text{for } n \geq 4 \tag{5.4}$$

where

hdwy (n) = departure headway between the (n-1)th and the nth vehicle

hdwy (1) = 2.04, hdwy (2) = 2.46, hdwy (3) = 2.12

To introduce variability to departure headways, the following procedure was adopted. This procedure can be applied for other parameters having the same probability distribution.

Step 1. Random number generator (output: rn(i)) : Generate uniform random deviates between 0.0 and 1.0.

Step 2. Random normal (0, 1) deviate generator (output: vnor(i)) : Generate normally distributed deviates with zero mean and unit variance by transforming uniform deviates to normal deviates.

Step 3. Random normal (m, s^2) deviate generator (output: hdwy(i)) : Generate normal random deviates with a mean **m** and a standard deviation **s** by transforming the output of step 2 according to:

$$hdwy(i) = vnor(i) * s + m \quad (5.5)$$

where

vnor = random normal (0, 1) deviates

s = standard deviation of departure headways

m = mean of departure headways

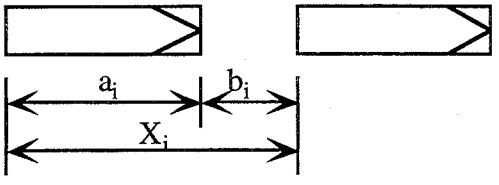
5.3.2 Vehicle Space in the Queue

As shown in Figure 5.1, a vehicle space (X_i) consists of a vehicle length (a_i) and clear space (b_i). Let a vehicle length (a_1,..., a_n) and clear space (b_1,..., b_n) be random variables (independent and identically distributed) which are normally distributed with means m_a and m_b and variances s_a^2 and s_b^2. Then a vehicle space (X_i,..., X_n) is a random variable which is normally distributed with a mean $m_a + m_b$ and variance $s_a^2 + s_b^2$.

$$a_i : N(m_a, s_a^2)$$

$$b_i : N(m_b, s_b^2) \quad (5.6)$$

$$X_i : N(m_a + m_b, s_a^2 + s_b^2) \quad (5.7)$$



a_i = vehicle length

b_i = clear space

X_i = vehicle space = $a_i + b_i$

Figure 5.1: A vehicle space

Let L = link length

n = the maximum integer value which satisfies the following inequality

$$X_1 + X_2 + ... + X_n \le L$$

(n = number of vehicles in a link when it is full)

Y = total vehicle space when a link is full (a linear combination of the X_i's)

$$Y = X_1 + X_2 + ... + X_n$$

Since total vehicle space (Y) is a linear combination of n vehicle spaces (X_i), it is normally distributed with a mean $n * (m_a + m_b)$ and variance $n * (s_a^2 + s_b^2)$.

$$Y : N(n(m_a + m_b), n(s_a^2 + s_b^2)) \quad (5.8)$$

Example:

Let • vehicle length (a): m_a = 14.7 ft, $s_a^2 = 1.35^2$ [Data collected by students in Traffic Engineering class at University of Texas at Austin taught by Dr. Clyde E. Lee; 110 vehicles in various university parking lots; 1994]

e.g.) $P(14.7 - 3 * 1.35 \le a \le 14.7 + 3 * 1.35)$

$= P(10.65' \le a \le 18.75') = P(127.8'' \le a \le 225'') = 99.7\ \%$

• clear space (b): m_b = 5.3 ft, $s_b^2 = 1.0^2$ [Edie (1961) and May (1990)) suggest jam density as approximately 250 vehicles per lane-mile which equals 21.1 ft average vehicle space. A conservative, rounded value of 20 ft is used for the average vehicle space, therefore, m_b is 5.3 ft (or 20 - 14.7).]

• link length (L): L = 200 ft

• n = 10

Then • vehicle space (X_i):

$$\mu_X = m_a + m_b = 14.7 + 5.3 = 20.0 \text{ ft.}$$

$$s_X^2 = s_a^2 + s_b^2 = 1.35^2 + 1.0^2 = 1.68^2$$

• total vehicle space ($Y = X_1 + X_2 + ... + X_{10}$):

$$\mu_Y = n(m_a + m_b) = 10 * 20.0 = 200.0 \text{ ft}$$

$$s_Y^2 = n(s_a^2 + s_b^2) = 10 * 1.68^2 = (5.31)^2$$

$P(m_Y - 2s_Y \le Y \le m_Y + 2s_Y) = 0.954$

$P(189.4 \le Y \le 210.6) = 0.954$

$P(m_Y - 3s_Y \le Y \le m_Y + 3s_Y) = 0.997$

$P(184.1 \le Y \le 215.9) = 0.997$

Approximately 95.4% of the values in any normal population lie within two standard deviations of the mean. It is indeed rare to observe a value from a normal population that is much further than two standard deviations from the mean. Since the length of two standard deviations in the above example is approximately one half of an average vehicle space, introducing vehicle space variability will not make a considerable difference in the value of n (the number of vehicles in a link when it is full). Therefore, vehicle space uniformity can be assumed throughout the simulation without loss of generality.

5.3.3 Vehicle Speed

Since the model is event based, identification of events to be tracked through the simulation process is important. Vehicular arrival and departure times are two such events. Arrival time in the simulation model is defined as the time a vehicle arrives at a link by joining a queue or,

if no queue exists, crossing the intersection reference line. Departure time is defined as the time a vehicle crosses the reference line and enters the intersection. If no queue exists in the link during a green interval, the intersection arrival time is the same as the departure time. The intersection arrival time is determined by the following relationship:

$$arr = depp + dist / speed \quad (5.9)$$

where

arr = arrival time at the downstream link

depp = departure time at the upstream intersection

dist = distance traveled from the reference line of the upstream intersection to the downstream link

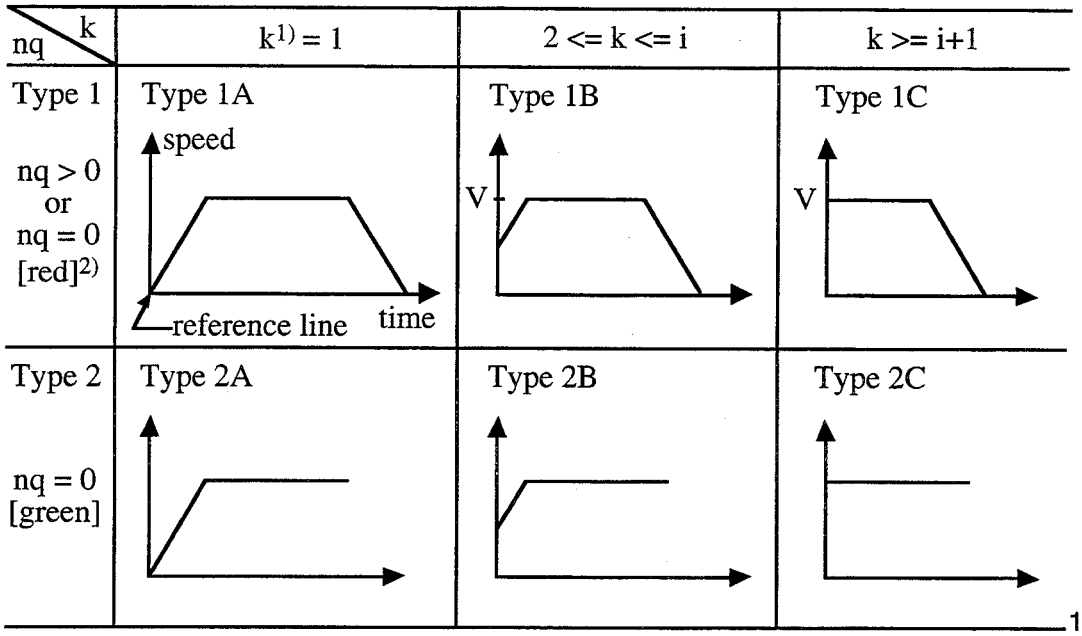
speed = average overall speed

Prior to experimental speed data collection, vehicle speed was hypothesized to depend upon vehicle queue position, and available downstream clear space. For a given downstream link length, available clear space depends upon downstream queue length. Therefore, according to the hypothesis, vehicle speed profiles from departure to arrival time should be determined by vehicle upstream link queue position and downstream link clear space determined by queue presence and length. If k indicates the vehicle queue position in the upstream link (k^{th} vehicle in a queue when a green indication begins) and nq is the number of vehicles queued in the downstream link when the vehicle arrives, six vehicle speed profile types are possible as presented in Figure 5.2. For convenience, constant acceleration and deceleration rates are assumed here.

If there is a queue stopped in the downstream link ($nq > 0$), an advancing vehicle must stop and join the queue. Even without a queue ($nq = 0$), a vehicle arriving during a red interval must stop. In either stopping case deceleration to a stop should be included in the vehicle speed profiles as shown in the Figure 5.2 Type 1 examples. If there is no queue stored in the downstream link during a green interval ($nq = 0$), an advancing vehicle passes the downstream intersection without stopping. In this case, the vehicle speed profile is one of the Type 2 examples shown in Figure 5.2.

If k equals one (first vehicle in the queue), an entire acceleration portion (from speed zero to speed V) is included in the vehicle speed profile (see the first column in Figure 5.2). For the second vehicle through the i^{th} vehicle in a queue ($2 \leq k \leq i$), the vehicle speed when passing the reference line is greater than zero. For the $(i+1)^{th}$ vehicle through the last vehicle in a departing, accelerating queue ($k > i$), the reference line speed is a cruising speed V. In this case, no

acceleration portion is included in the vehicle speed profile as shown in the last column of Figure 5.2.



1) k = index for vehicle position in departing queue

2) [red] : a vehicle arrives at the downstream intersection during a red interval

Figure 5.2: Vehicle speed profiles

A field study was conducted to provide appropriate distributions of average overall speeds. The site was a six-lane arterial in the Central Business District of Austin, Texas. Figure 5.3 shows a test site layout. Two phase operation and a 60 second (90 second at 2nd Street) fixed-time traffic signal cycle were used.

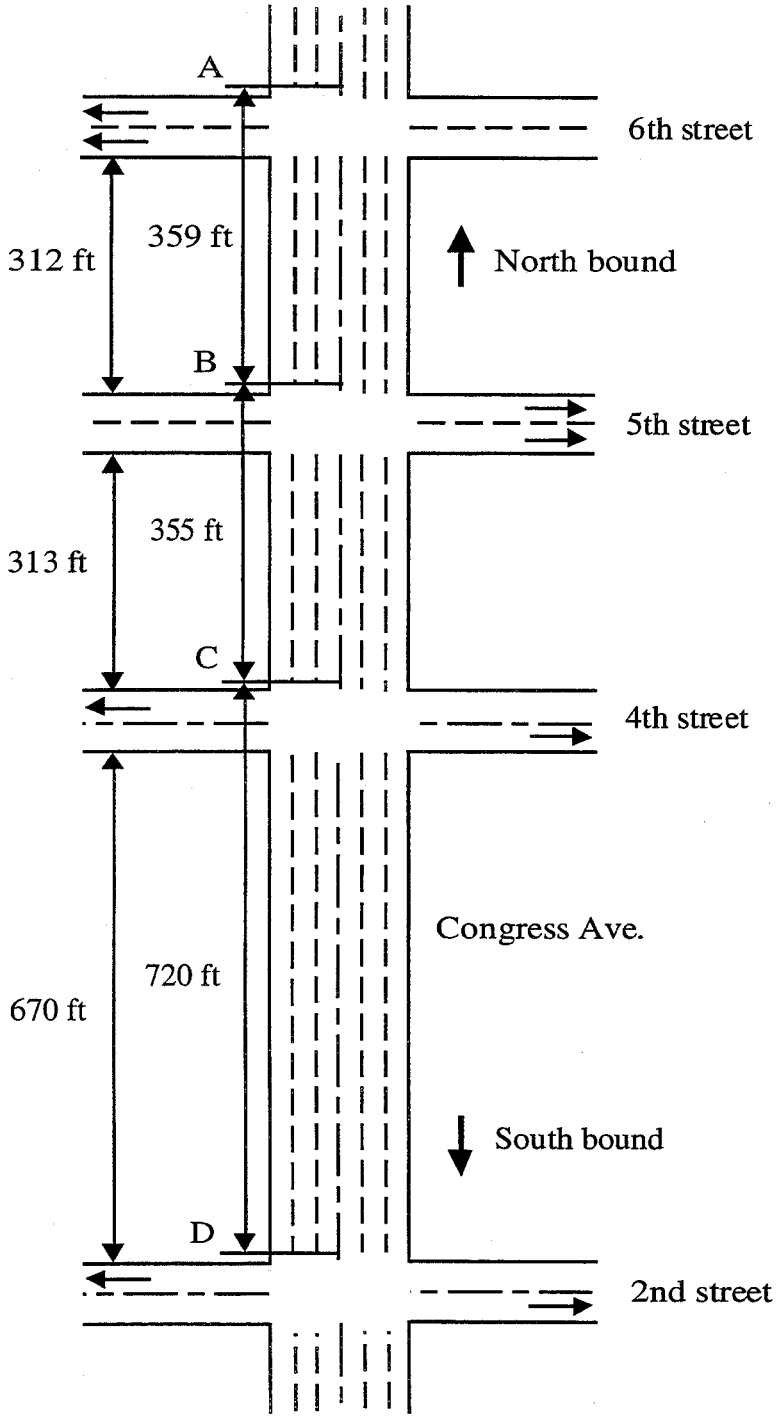


Figure 5.3: Field study site configuration

Type 1 speed

Preliminary observation of southbound traffic indicated most vehicles accelerating past reference line A, at Sixth Street, (see Figure 5.3) stopped in the downstream link due to the Fifth Street signal offset. Therefore, the middle lane of southbound Congress Avenue between Sixth and Fifth Streets was selected for Type 1 speed data collection. The site was videotaped from the roof of a 22-story building located at Congress Avenue and Third Street. Since Type 1 speed data collected from this site provide short downstream clear space (up to 370 ft), the middle lane of southbound Congress Avenue between Fourth and Second Streets was also selected for Type 1 speed data with long downstream clear space (up to 730 ft). Due to different traffic signal cycle lengths at Fourth and Second Streets, both Type 1 speed and Type 2 speed profiles could be observed here. This site was videotaped from the 32nd floor of a building located at Congress Avenue and Sixth Street. During the replay of the videotape, departure times at reference lines A and C and arrival times in the downstream links were measured for each vehicle using a stopwatch which measured to the nearest one-hundredth second. Also, the traveled distance for each vehicle was recorded so that an average overall speed could be calculated. No heavy vehicles or vehicles passing the downstream intersection without stopping were considered.

Results of this field study are summarized in Table 5.5. The data were used to examine the hypothesis that vehicle speeds were functions of downstream clear space and the vehicles own queue position. As indicated in Table 5.5, speeds for different queue positions (k values) are very similar and, in fact, the four queue position means of each data set (Table 5.5-a/b) are within a 99 percent confidence interval of each grand mean. Therefore, these data indicate no statistically significant effect of queue position on vehicle speeds. Available downstream clear space and speed, however, appear to have a statistically significant relationship. This relationship has been captured in regression equation (5.10) which is shown with the plotted data in Figure 5.4. The equation predicts overall average (or space mean) speed using a variable called downstream clear space. The definition of downstream clear space as used here is provided in Figure 5.5. The correlation coefficient for speed versus downstream clear space (1061 data points) is statistically significant beyond the 1 percent confidence level and Student's T test of the slope being different from zero also shows significance beyond the 1 percent level. Therefore, the following relationship including all data points was selected for predicting Type 1 speed as a function of downstream clear space:

$$y = 13.033 + 0.026584\,x \quad (x \leq 730 \text{ ft}, \quad R = 0.894) \tag{5.10}$$

where

y = average overall speed (ft/sec)

x = downstream clear space (ft, see Figure 5.5)

R = correlation coefficient

TABLE 5.5: RESULTS OF FIELD STUDY FOR TYPE 1 AVERAGE OVERALL SPEED

position (k)	no. obs.	mean (ft/sec)	mean (mph)	std. dev.(ft/sec)
a. Type 1 speed data with short downstream clear space [1]				
1	85	17.8	12.1	2.78
2	53	17.5	11.9	2.44
3	38	17.3	11.8	2.66
> 4	412	18.3	12.5	3.48
all	588	18.1	12.3	3.27
b. Type 1 speed data with long downstream clear space [2]				
1	66	29.1	19.8	3.55
2	61	29.4	20.1	3.02
3	52	28.9	19.7	2.75
> 4	294	28.9	19.7	3.70
all	473	29.0	19.7	3.50

1) Data collected from southbound Congress Avenue between 6th and 5th Streets (downstream clear space ≤ 370 ft)

2) Data collected from southbound Congress Avenue between 4th and 2nd Streets (350 ft ≤ downstream clear space ≤ 730 ft)

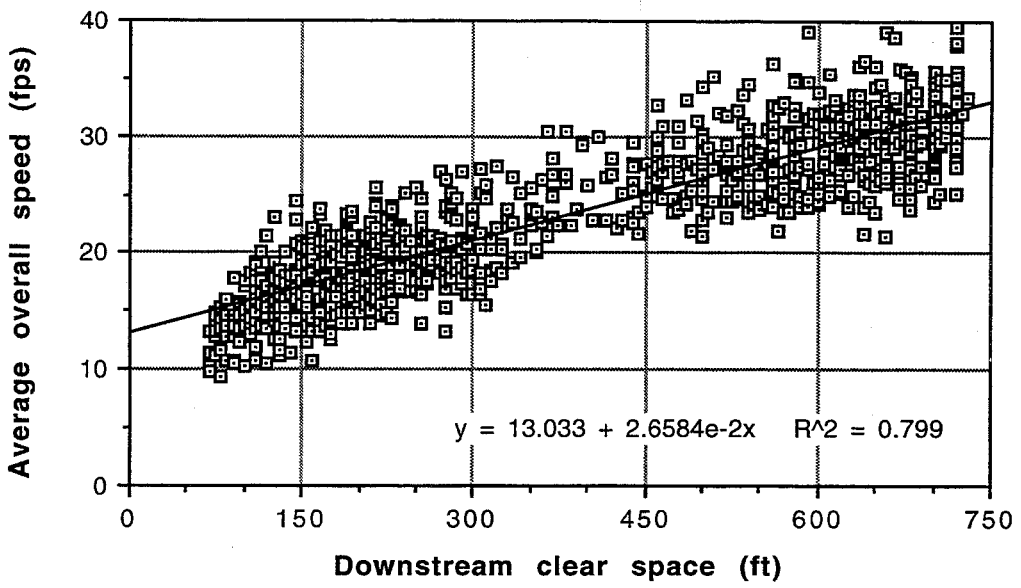


Figure 5.4: Relationship between Type 1 speed and downstream clear space (data points: 1061)

Figure 5.5 illustrates downstream clear space and queue spillback. Downstream clear space is defined as the downstream link length to be traveled by a vehicle departing the reference line in lane i. As shown in equation (5.10), the average overall speed of vehicle A (see Figure 5.5-a) is a function of the downstream clear space. As the available downstream link clear space decreases, the average overall speed decreases.

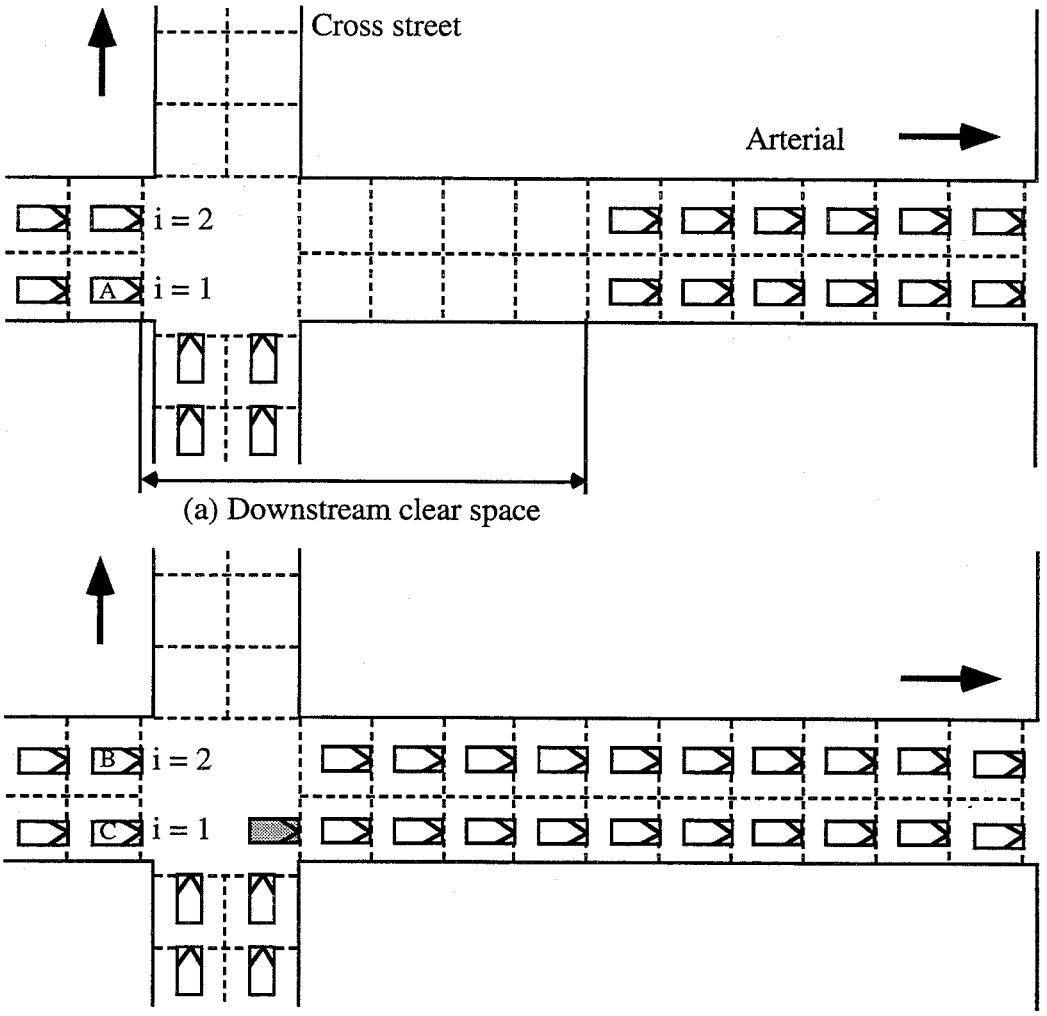


Figure 5.5: Downstream clear space and queue spillback

When the downstream link is full during the upstream link green interval [see arterial lane 2 (i = 2) in Figure 5.5-b], one additional vehicle B is permitted to join the downstream queue causing queue spillback. A non zero speed is assigned by equation (5.10) to any vehicle departing the upstream intersection unless a queue spillback has already occurred in the downstream link [see arterial lane 1 (i = 1) in Figure 5.5-b]. In this case, vehicle C cannot move forward, therefore, no speed is assigned. As soon as space becomes available, vehicle C moves with speed determined by equation (5.10).

When the downstream link (i+1) is filled with stopped vehicles, upstream vehicles (link i) with a green signal, sometimes move into the intersection joining the stopped queue constituting

a queue spillback. Other times, under the same conditions, upstream vehicles wait for downstream clear space(s) before advancing avoiding being trapped in the intersection. This driver behavioral characteristic could not be reliably predicted so a compromise was chosen. Given a green signal and a filled downstream link, one storage space in the intersection is always provided to the advancing traffic stream. This concept is a reasonable behavioral compromise between the extremes of no movement without downstream clear space and always filling the intersection behind the stopped downstream queue.

Therefore, the downstream link length available for queued vehicles includes storage spaces in link i+1 plus one intersection space. If all these storage spaces are full, then one vehicle is stopped in the intersection, and zero speed is assigned to the next vehicle attempting movement into the intersection. If these storage spaces are not full, non zero speeds are assigned to upstream vehicles using equation (5.10). Within the simulation code, the case of link i+1 and the intersection space being filled is handled as a special situation.

Type 2 speed

As shown in Figure 5.2, vehicles with Type 2 speed profiles pass the downstream intersection without stopping. Vehicles with Type 2C speed profile reach the cruise speed V before or when crossing the reference line. For the speed data application purpose, Type 2C speed profile was divided into two categories: 1) Type 2C-1, in which vehicles join the stopped queue and cross the reference line and 2) Type 2C-2, in which vehicles pass both the upstream intersection and the reference line without stopping.

As explained before, observation of Congress Avenue vehicles crossing reference line C (see Figure 5.3) indicated that some of them passed through the downstream intersection without stopping due to the different traffic signal cycle lengths at Fourth and Second Streets. Since this trajectory is the Type 2 speed profile, the middle lane of southbound Congress Avenue between Fourth and Second Streets was selected for Type 2 speed data collection. This site was videotaped from the 32nd floor of a building located at Congress Avenue and Sixth Street. Departure time at reference line C (see Figure 5.3) and crossing times at six points (80, 160, 240, 360, 500, and 640 ft from reference line C) in the downstream link were measured for each vehicle during videotape replay. Only vehicles passing the downstream intersection without stopping (Type 2 speed profile) were considered. Heavy vehicles or vehicles with large deceleration rates were excluded.

Results of this field study are summarized in Table 5.6 and Figure 5.6. These data provided another opportunity to test the hypothesis that overall average speeds from departure to arrival vary with queue position and travel distance [Instead of downstream clear space, travel

distance was used in Type 2 speed case.]. As presented in Table 5.6 and Figure 5.6, when travel distance is short (80 ft or 160 ft), space mean speeds (Types 2A/2B/2C-1) among the four queue positions ($k = 1, 2, 3, \text{or } k \geq 4$) are different and the differences are statistically significant at the 0.01 level. On the other hand, when travel distance is long (500 ft or 640 ft), space mean speeds (Types 2A/2B/2C-1) among the four queue positions ($k = 1, 2, 3, \text{or } k \geq 4$) are not statistically different at the 0.01 level. Therefore, the effect of queue position on vehicle speeds varies with travel distance. When travel distance is short, a statistically significant queue position effect on vehicle speed exists. On the other hand, when travel distance is long, the effect of queue position on vehicle speeds is not statistically significant.

Due to the nature of oversaturated traffic operations, however, Type 2 speed profiles are rarely observed. Even though space mean speeds among the different queue positions are different when travel distance is short (80 ft or 160 ft), minimum travel distance applied for Type 2 speed is 240 ft. Space mean speeds among the different queue positions are not considerably different when travel distance is medium (240 ft or 360 ft). Of course, the differences among space mean speeds for the different queue positions are not statistically significant at a 0.01 level when travel distance is long (500 ft or 640 ft). Therefore, judgment was made to use one grand mean and standard deviation for each travel distance (see series 5 in Figure 5.6 and Table 5.6) instead of using four different statistics for the four queue position groups for each travel distance.

As shown in series 6 of Table 5.6 and Figure 5.6, Type 2C-2 speeds are, regardless of travel distance almost constant. Since Type 2C-2 speed profiles can exist only when vehicles pass both the upstream intersection and the reference line without stopping, the hypothesized relationship between speeds and queue position was not relevant. All six means are within a 99 percent confidence interval of the grand mean 42.8 ft/sec. Therefore, these data indicate no statistically significant effect of travel distance on vehicle speeds. The same procedure applied to the departure headway in subsection 5.3.1 was used to introduce variability to vehicle speeds.

TABLE 5.6: RESULTS OF FIELD STUDY FOR TYPE 2 AVERAGE OVERALL SPEED

k (freq)	distance[1] (ft)	80	160	240	360	500	640
a) Types 2A/2B/2C-1 (Number of observations: 612)							
	mean (fps)	19.6	25.3	28.5	31.7	34.6	36.2
1) k = 1	mean (mph)	13.4	17.3	19.4	21.6	23.6	24.7
(26)	st.dev. (fps)	2.36	2.71	2.77	3.02	3.42	3.62
	mean (fps)	23.6	28.4	30.6	32.9	35.1	36.1
2) k = 2	mean (mph)	16.1	19.3	20.9	22.4	23.9	24.6
(25)	st.dev. (fps)	2.55	2.60	2.61	2.81	3.20	3.18
	mean (fps)	27.0	31.0	32.6	34.3	36.0	36.9
3) k = 3	mean (mph)	18.4	21.1	22.2	23.4	24.5	25.2
(21)	st.dev. (fps)	2.46	2.63	2.90	3.11	3.22	3.46
	mean (fps)	30.2	32.6	33.4	34.3	35.8	36.6
4) k ≥ 4	mean (mph)	20.6	22.2	22.8	23.4	24.4	24.9
(30)	st.dev. (fps)	3.05	2.83	2.93	3.21	3.56	3.47
	mean (fps)	25.2	29.4	31.3	33.3	35.3	36.4
5) k = n	mean (mph)	17.2	20	21.3	22.7	24.1	24.8
(102)	st.dev. (fps)	4.85	3.90	3.38	3.19	3.36	3.40
b) Type 2C-2 (Number of observations: 534)							
	mean (fps)	42.8	43.7	43.0	42.2	42.6	42.4
6) k = n	mean (mph)	29.2	29.8	29.3	28.8	29.0	28.9
(89)	st.dev. (fps)	6.66	6.50	6.34	6.23	6.24	5.91

1) Travel distance from the reference line

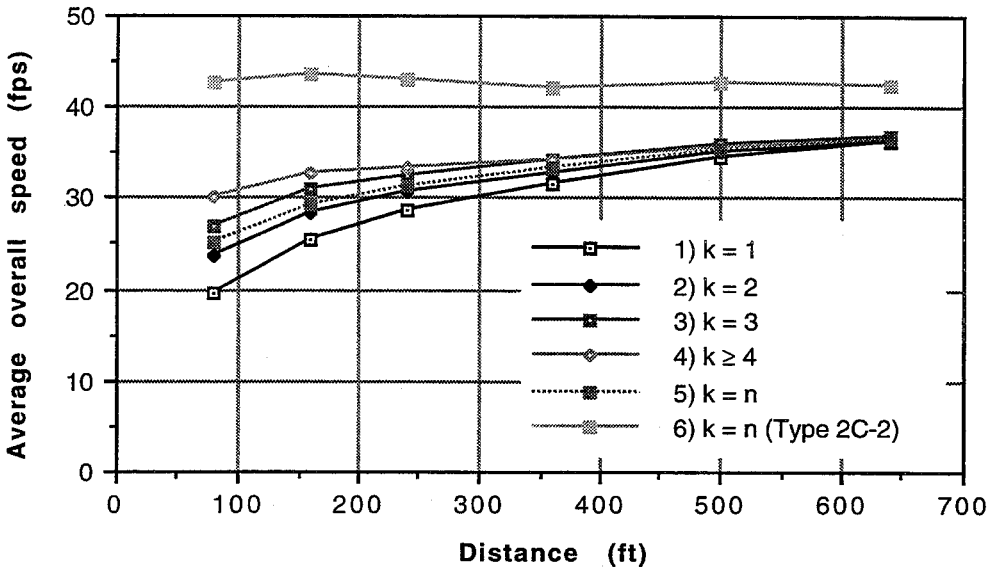


Figure 5.6: Relationship between Type 2 speed and distance for different queue positions

As discussed in Type 2 speed data analysis, travel distance and Type 2 speed (Types 2A/2B/2C-1) appear to have a statistically significant relationship. As an alternative, an attempt was made to draw a general relationship between Type 2 speed (Types 2A/2B/2C-1) and travel distance from the Type 2 speed data based upon the assumption that Type 2 speed trajectories of vehicles in different queue positions are similar. Travel distances of Type 2 speed data were adjusted for the second through the last vehicle in a queue ($k \geq 2$) by adding the distance from the reference line to each vehicle stopped position to each vehicle travel distance. Therefore, unlike Type 2 speed data in Table 5.6 and Figure 5.6, in this relationship travel distance indicates the distance not from the reference line but from the stopped queue position. This relationship has been captured in regression equation (5.11) which is shown with the plotted data in Figure 5.7. Therefore, the following relationship including all data points was selected for predicting Type 2 speed (Types 2A/2B/2C-1) as a function of travel distance:

$$y = -6.1525 + 15.268 * LOG(x) \quad (x \leq 760 \text{ ft}, \ R = 0.785) \quad (5.11)$$

where

y = average overall speed (fps)

x = travel distance (ft)

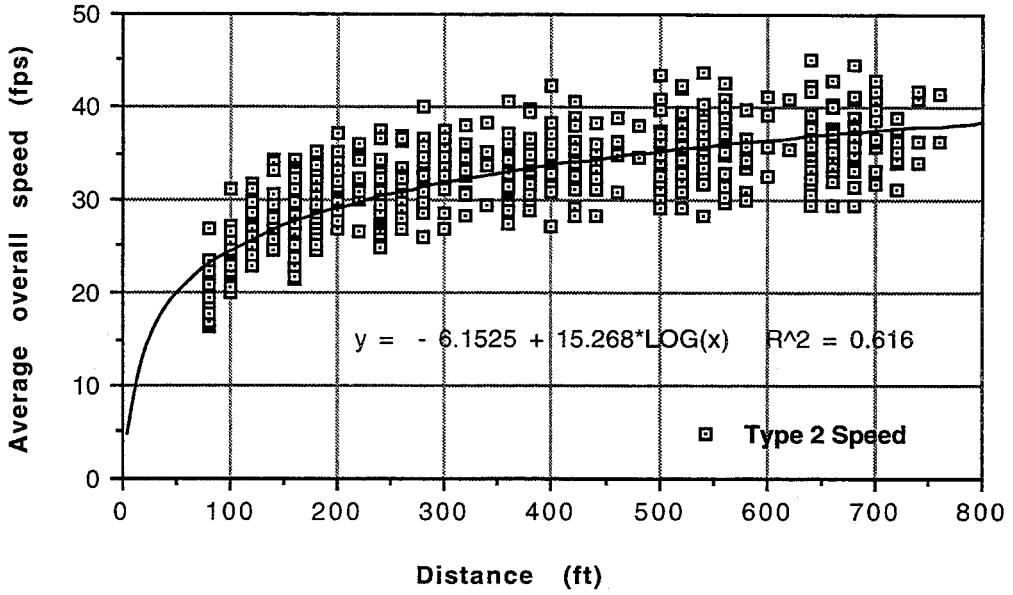


Figure 5.7: Relationship between Type 2 speed and travel distance (data points: 612)

5.4 ONE-WAY ARTERIAL OPERATION

As shown in Figure 5.8, a one-way two-lane arterial consisting of three intersections with one-way two-lane cross streets is considered. A common traffic signal cycle length along the arterial and basic two-phase operation are applied.

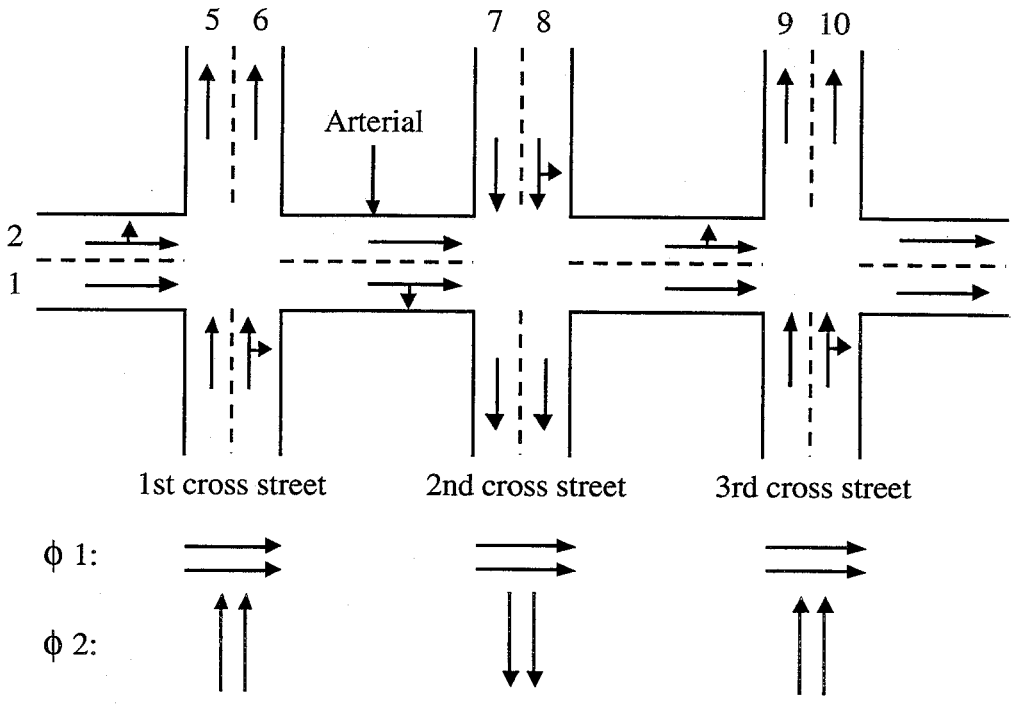


Figure 5.8: Arterial configuration for a one-way operation

5.4.1 Experimentation

With the experimental design derived in the previous section (see Table 5.4), one-way operation experimentation was conducted. As shown in Table 5.4, two different link lengths (200 and 600 feet) and four different cycle lengths (60, 75, 90, and 105 seconds) were used. A system performance measure was the time required to process a given number of vehicles (2000 here) through the arterial system consisting of two arterial lanes and six cross street lanes.

Tables 5.7 and 5.8 present summarized results when links are 200 feet and 600 feet long respectively. Each table shows results of 126 simulation runs that are one fourth of the total 504 design points. The revised experimental design requires two different levels of N for both arterial and cross street turns (turning movements as percentages of link length). However, twenty percent for turning movements into cross streets and 10 percent for turning movements into the arterial or one value per street type, was used. The reasons are: first, preliminary simulation results using each of the two N values for each street type were not noticeably different from each other. Second, since queue spillbacks from cross streets block the arterial causing more serious system performance effects than spillbacks from the arterial, a larger value (20 percent) was applied for turning movements into cross streets. Therefore, the total design points were reduced to 252.

Results of the simulation experiments indicate that for a given link length, the overall pattern of results is very similar regardless of the cycle length. In all cases, particular patterns, which consist of similar network crossing times, are formed along the diagonal whether they are maximum or minimum [see the shaded cells or adjacent cells in Tables 5.7 and 5.8]. Only the magnitude of the network crossing times, not the pattern, changes when a cycle length changes.

For 200 foot links, the simultaneous green, which means simultaneous onset of the green phases servicing the arterial, produces minimum network crossing time irrespective of the cycle length. This result is identical to what Lieberman et al. (1986) found in their study. The result of their study indicates that in the presence of moderate arterial queues, the optimal relative offsets along these arterials are approximately zero (simultaneous green). Moreover, their study is based upon closely spaced high traffic density networks which are very close to the conditions of this study (especially for a 200 foot link case).

With the same link length and cycle length, combinations of offset 2 and offset 3 produce a wide network crossing time range. When offset 2 is 15 seconds greater than offset 3, network crossing times are maximum (see the shaded cells in the Table 5.7) indicating lowest efficiency. When offset 2 is equal to offset 3, each network crossing time, within the same row or column, is minimum with a few exceptions. When offset 2 and offset 3 are 30 seconds respectively, network crossing times are greater than minimum. This indicates that there is a combined effect of offsets on the network crossing times, or system efficiency.

TABLE 5.7: NETWORK CROSSING TIME (SEC) [ONE-WAY OPERATION, L = 200 FT]

offset 3	0	15	30	45	60	75	90
offset 2	a. C = 60 sec. (g = 40, r = 20)						
0	1220	1370	2132	2007			
15	2501	1237	1385	1705			
30	1822	2127	1448	1517			
45	1499	1824	2426	1386			
	b. C = 75 sec. (g = 55, r = 20)						
0	1261	1344	1812	1917	3211		
15	3581	1268	1357	1751	2052		
30	2174	3581	1447	1510	1912		
45	1821	2174	3581	1515	1649		
60	1501	1938	2174	3581	1420		
	c. C = 90 sec. (g = 70, r = 20)						
0	1336	1419	1872	1917	2307	3856	
15	4301	1336	1423	1749	2189	2472	
30	2609	4289	1511	1562	1874	2391	
45	2385	2609	4257	1556	1655	1912	
60	1956	2193	2609	4299	1572	1683	
75	1535	1895	2148	2609	4301	1451	
	d. C = 105 sec. (g = 85, r = 20)						
0	1388	1495	1791	2032	2247	2697	4501
15	5021	1399	1460	1803	2079	2564	2892
30	3044	5009	1556	1689	1896	2237	2796
45	2459	3044	4977	1594	1690	1908	2266
60	2244	2446	3044	4962	1594	1690	1962
75	1940	2286	2568	3044	5019	1581	1681
90	1572	1892	2101	2508	3044	5021	1504

Figures 5.9 - 5.12 illustrate results of simulation runs when links are 200 feet long (same as Table 5.7).

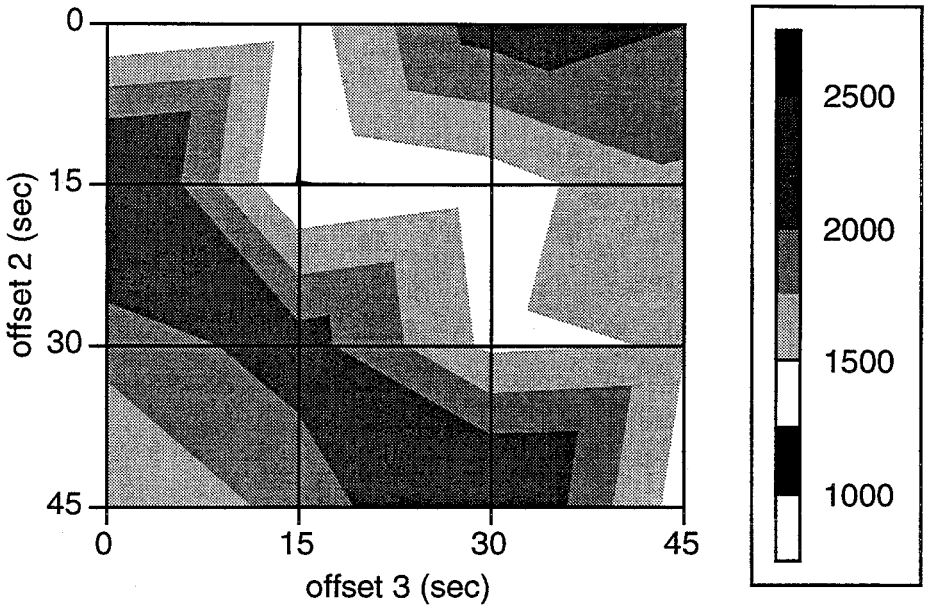


Figure 5.9: Network crossing time (sec) [one-way, L = 200 ft, C = 60 sec]

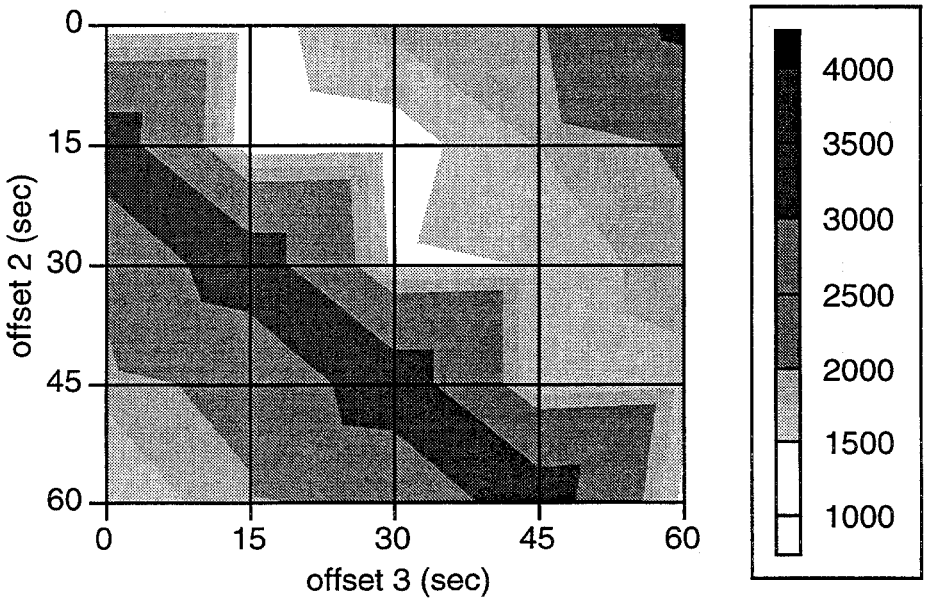


Figure 5.10: Network crossing time (sec) [one-way, L = 200 ft, C = 75 sec]

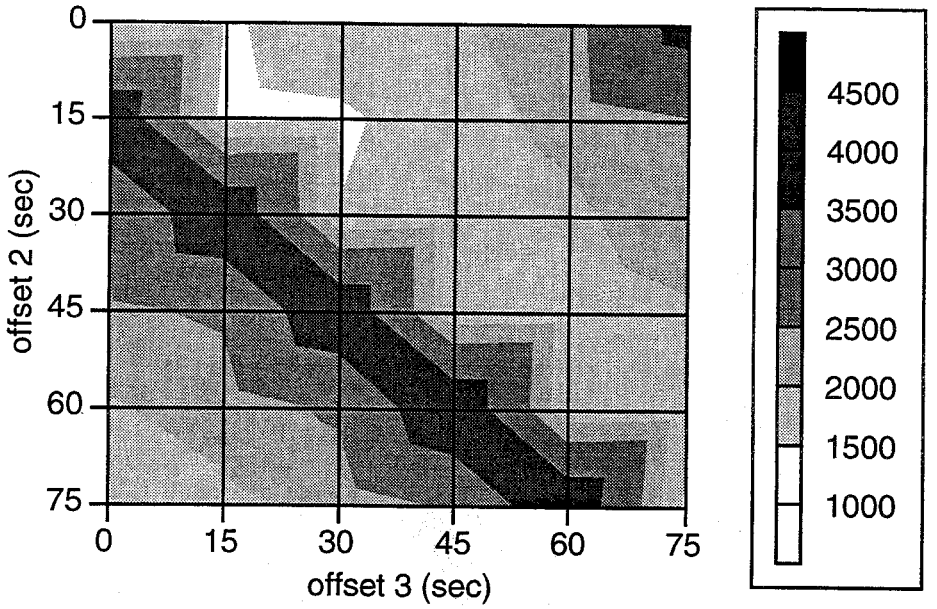


Figure 5.11: Network crossing time (sec) [one-way, L = 200 ft, C = 90 sec]

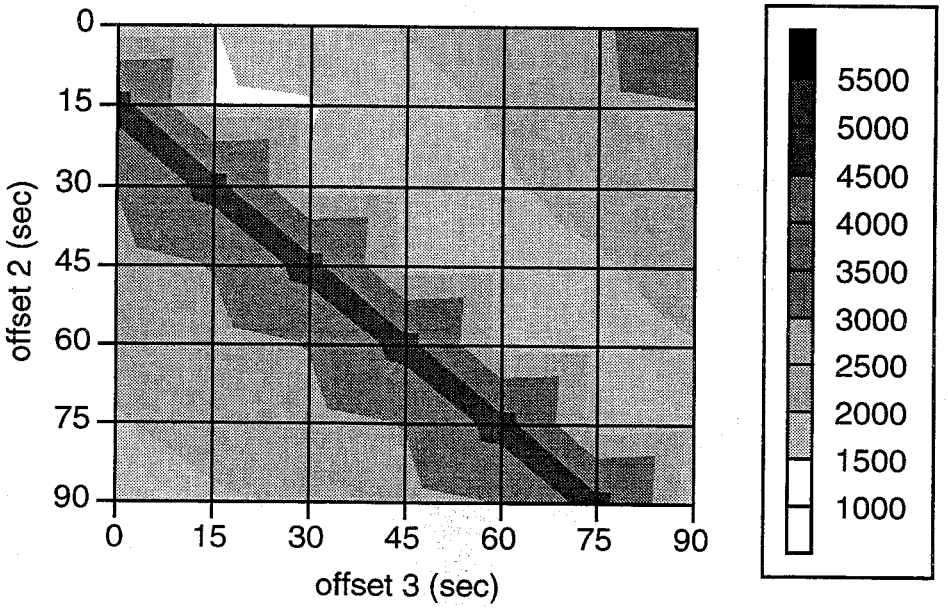


Figure 5.12: Network crossing time (sec) [one-way, L = 200 ft, C = 105 sec]

When links are 600 feet long, simultaneous green does not produce minimum network crossing times as shown in Table 5.8. Furthermore, compared with the shaded cells of Table 5.7, the shaded cells of Table 5.8, which indicate maximum network crossing times, are shifted downward by 15 seconds. This is due primarily to the link length difference which in turn leads to an increase in the travel time. When links are 200 feet long and are empty, it takes 12.4 seconds for the first vehicle in the queue to arrive at the downstream intersection (travel distance is 240 feet including cross street width of 40 feet). When a link length is 600 feet, it takes 21.3 seconds for the first vehicle in the queue to arrive at the downstream intersection. Thus, the difference of link length (400 feet) causes a 8.9 second increase in travel time, which is close to the 15 seconds mentioned above.

When offset 2 is 30 seconds greater than offset 3, network crossing times are maximum (see the shaded cells in the Table 5.8) indicating lowest efficiency. When offset 2 is 15 seconds greater than offset 3, network crossing times are minimum indicating highest efficiency. This result is different from that of a shorter link case (200 feet). Consequently, this result implies that under a fairly saturated condition, as link length becomes longer, the green indication of a downstream intersection should start sufficiently earlier than that of an upstream intersection so that the downstream link can have room for incoming vehicles. Figures 5.13 - 5.16 illustrate results of simulation runs when links are 600 feet long (same as Table 5.8).

TABLE 5.8: NETWORK CROSSING TIME (SEC) [ONE-WAY OPERATION, L = 600 FT]

offset 3	0	15	30	45	60	75	90
offset 2	a. C = 60 sec. (g = 40, r = 20)						
0	1488	2024	2600	1217			
15	1245	1594	2019	2289			
30	2464	1324	1591	2017			
45	1859	2450	1209	1432			
	b. C = 75 sec. (g = 55, r = 20)						
0	1432	1797	2388	3209	1272		
15	1268	1502	1917	2384	3196		
30	2956	1316	1583	1904	2384		
45	2382	3396	1373	1557	1887		
60	1693	2195	3383	1259	1427		
	c. C = 90 sec. (g = 70, r = 20)						
0	1512	1816	2148	2868	3854	1324	
15	1325	1502	1801	2298	3123	3841	
30	4052	1331	1563	1897	2288	3108	
45	3548	3731	1364	1622	1893	2277	
60	2262	3539	4063	1400	1619	1889	
75	1712	2004	3273	4094	1319	1504	
	d. C = 105 sec. (g = 85, r = 20)						
0	1533	1892	2001	2508	4202	4797	1387
15	1375	1533	1752	2102	3194	3648	4799
30	4722	1382	1527	1824	2217	3198	4160
45	4147	4478	1447	1579	1893	2217	3191
60	3179	4150	4064	1447	1629	1892	2213
75	2532	3168	4190	4745	1448	1603	1889
90	1750	2209	2757	3344	4784	1380	1572

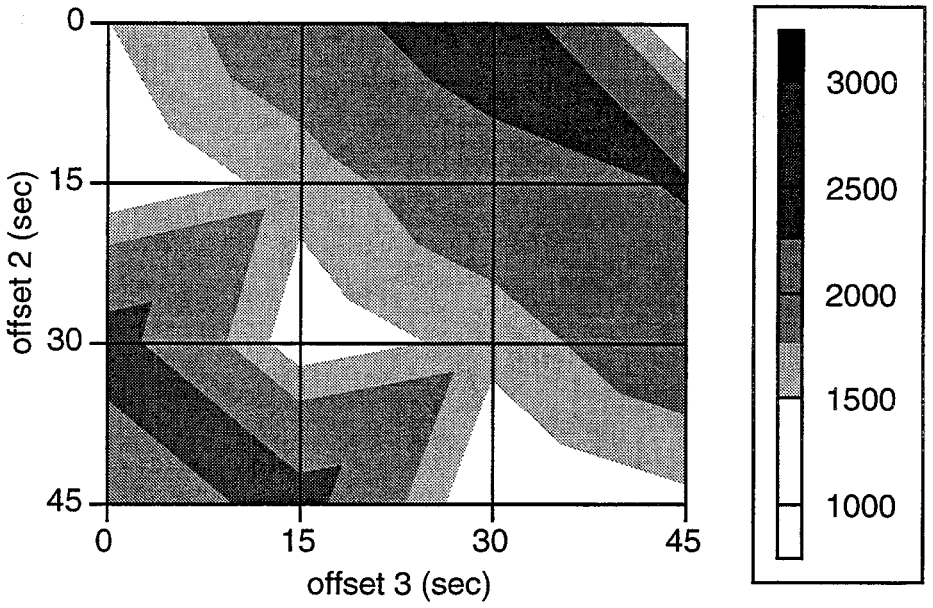


Figure 5.13: Network crossing time (sec) [one-way, L = 600 ft, C = 60 sec]

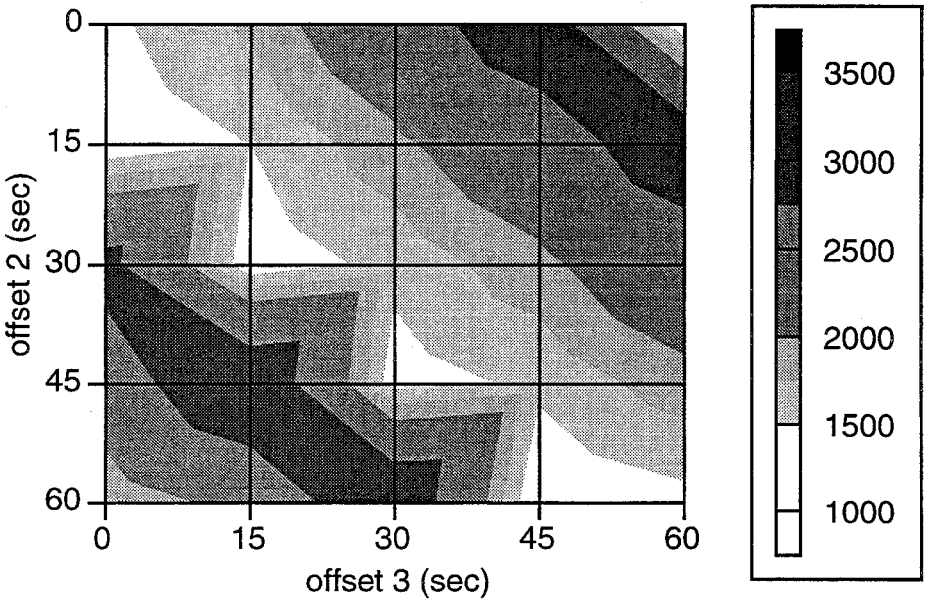


Figure 5.14: Network crossing time (sec) [one-way, L = 600 ft, C = 75 sec]

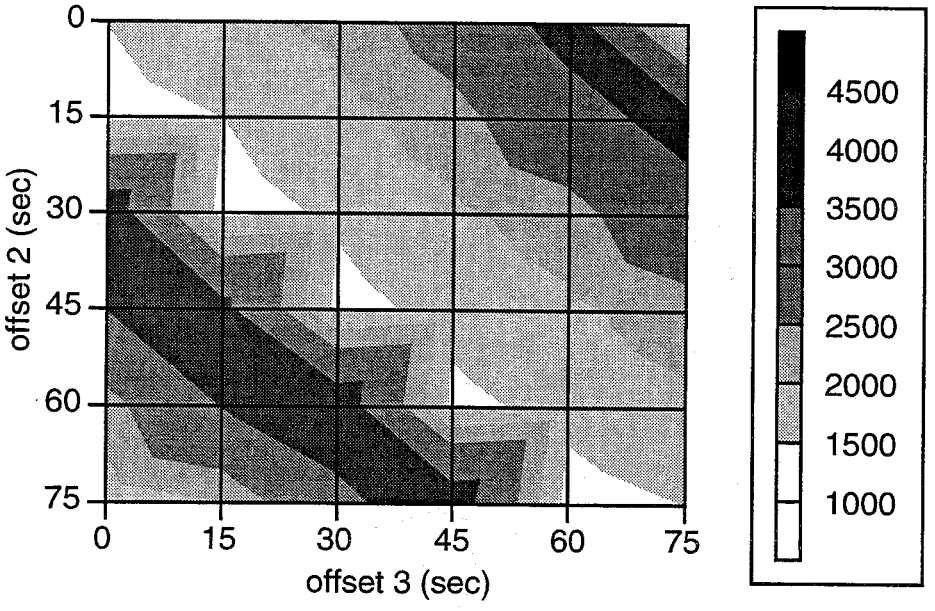


Figure 5.15: Network crossing time (sec) [one-way, L = 600 ft, C = 90 sec]

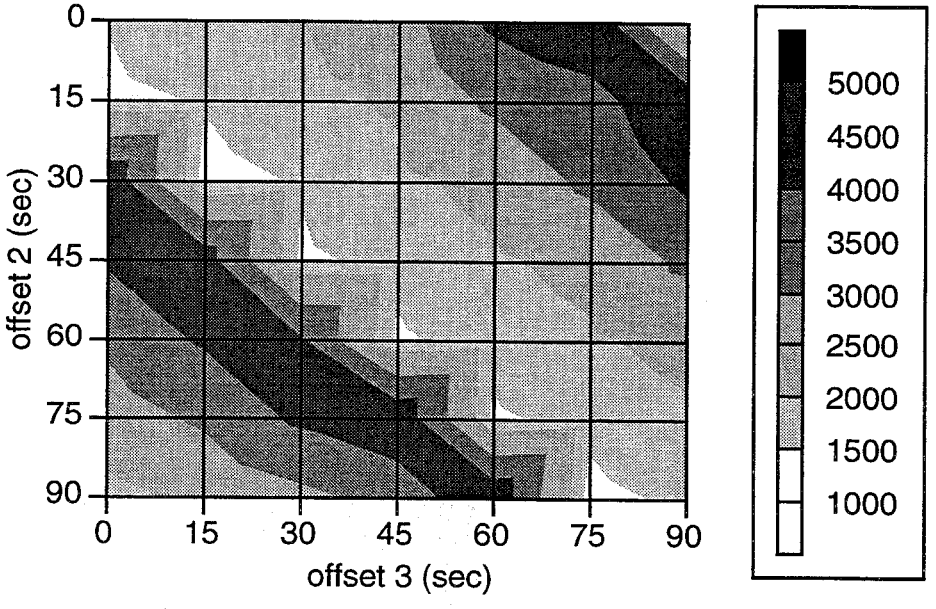


Figure 5.16: Network crossing time (sec) [one-way, L = 600 ft, C = 105 sec]

With the same link length, different offset combinations yielded a wide range of network crossing times. For example, the network crossing time ranges from 1388 seconds to 5021 seconds for a 105 seconds cycle length for a 200 feet link case. A small offset difference can

make a big difference in results (one second change in offset 2, from 7 seconds to 8 seconds, for a 105 seconds cycle length for a 200 feet link case, made 1762 seconds difference in network crossing time). Consequently, it can be said that offset is one of the dominant factors affecting system efficiency. Two different link lengths, 200 and 600 feet, produced significantly different results for the most and least efficient offset combinations even though the overall diagonal pattern of results was similar.

As shown in Tables 5.7 and 5.8, when best offsets are not used, network crossing times increase as the arterial green increases. However, when best offsets are used, the arterial green interval does not have a considerable effect on the results. When best offsets are not used, the delay caused by a queue spillback increases as the arterial green increases. For example, when one of the worst offset combinations, 15 second offset 2 and zero offset 3, was used, network crossing times for the 40, 55, 70, and 85 second arterial green intervals (60, 75, 90, 105 second cycle lengths, respectively), were 2501, 3581, 4301, and 5021 seconds, respectively. The delay, caused by a queue spillback in lane 7 on the arterial (see Figure 5.8), for the 60, 75, 90, and 105 second cycle length, was about 28, 48, 63, 78 seconds per cycle respectively.

To examine the effect of a link length on the results, minimum and maximum network crossing times when links are 200 and 600 feet long respectively, are compared. For each cycle length, a 600 foot link produces slightly more efficient results than a 200 foot link. Minimum network crossing times of a 600 foot link are slightly shorter than those of a 200 foot link. Maximum network crossing times of a 200 foot link are slightly longer than those of a 600 foot link. Since a shorter link has smaller link storage capacity, the link becomes saturated more quickly. With the same traffic conditions, therefore, queue spillback is more likely to occur and delay caused by queue spillback will increase as link length decreases.

5.4.2 Experimentation with Variability

As discussed in section 5.3 variability was introduced to parameters, departure headway and vehicle speed. Tables 5.9 and 5.10 present summarized results when links are 200 feet and 600 feet long respectively. In general, the experimentation result with variability is not very different from the result without variability. This is because each parameter, generated randomly with a given probability distribution, has little effect on the result. Combinations of these parameters, which may produce additive parameter deviations, are more likely to affect the result.

As shown in Tables 5.9 and 5.10, with random variability both the minimum and maximum network crossing times for each cycle length and link length tend to become more extreme compared to the no variability cases. However, generally, variability does not change

earlier conclusions regarding best cycle lengths or offset combinations. In other words, randomness of headway and speed variables is equally likely to have positive, negative, or no effect on the simulation results.

TABLE 5.9: NETWORK CROSSING TIME (SEC) [ONE-WAY OPERATION, WITH VARIABILITY, L = 200 FT]

offset 3	0	15	30	45	60	75	90
offset 2	a. C = 60 sec. (g = 40, r = 20)						
0	1379	1384	1855	2003			
15	2430	1333	1416	1992			
30	2169	2077	1679	1515			
45	1484	1856	2228	1486			
	b. C = 75 sec. (g = 55, r = 20)						
0	1443	1413	1831	1944	3045		
15	3287	1517	1617	1779	2068		
30	2233	3425	1965	1519	1960		
45	1979	2221	3047	1598	1596		
60	1497	1921	2232	3066	1485		
	c. C = 90 sec. (g = 70, r = 20)						
0	1657	1500	2093	2144	2327	3151	
15	4027	1551	1471	1824	2210	2758	
30	2660	4096	1676	1607	1911	2393	
45	2451	2667	3908	1711	1651	1908	
60	1906	2215	2685	3528	1708	1640	
75	1559	1873	2121	2682	4020	1633	
	d. C = 105 sec. (g = 85, r = 20)						
0	1564	1569	1883	2026	2489	3206	4183
15	4813	1570	1617	1867	2080	2587	2922
30	3116	4676	2003	1701	1912	2236	2817
45	2677	3138	4733	1748	1699	1904	2225
60	2234	2491	3121	4506	1617	1676	1959
75	1913	2232	2615	3134	3795	1585	1665
90	1569	1856	2237	2468	3101	4822	1677

TABLE 5.10: NETWORK CROSSING TIME (SEC) [ONE-WAY OPERATION, WITH VARIABILITY, L = 600 FT]

offset 3	0	15	30	45	60	75	90
offset 2	a. C = 60 sec. (g = 40, r = 20)						
0	1488	2034	2243	1226			
15	1279	1601	2027	2082			
30	2762	1328	1587	2036			
45	1857	2653	1209	1427			
	b. C = 75 sec. (g = 55, r = 20)						
0	1489	1795	2417	3373	1244		
15	1297	1489	1928	2446	3481		
30	3193	1316	1576	1934	2431		
45	2486	3331	1392	1564	1898		
60	1724	2729	3268	1257	1475		
	c. C = 90 sec. (g = 70, r = 20)						
0	1511	1761	2530	3187	3326	1422	
15	1380	1586	1797	2305	3210	4107	
30	4011	1354	1551	1916	2735	3047	
45	3546	3829	1397	1627	1890	2720	
60	2717	2901	4547	1422	1625	1887	
75	1892	2427	3271	3386	1603	1531	
	d. C = 105 sec. (g = 85, r = 20)						
0	1564	1897	2082	2517	3516	4781	1509
15	1393	1567	1768	2088	3221	3649	4808
30	4783	1538	1560	1787	2216	3208	4261
45	4249	4674	1489	1561	1880	2226	3196
60	3184	3401	3668	1434	1638	1891	2532
75	2532	2691	4211	4786	1498	1644	1872
90	1844	2543	2756	3191	4839	1551	1571

5.4.3 Examples of Simulation Runs

For convenience and without losing generality, a selected cycle length and link length were used to examine the results in detail. Short cycle length (75 seconds) with a short link length (200 feet), and long cycle length (105 seconds) with a long link length (600 feet) are selected. For each case, the best and worst offset combinations were chosen for further explanations.

Short cycle length and short link length

Input parameters include the following:

Σ Cycle length = 75 seconds (55 seconds arterial green, 20 seconds arterial red)
Σ Turning movements = 2 vehicles per cycle from arterial to cross street (N = 20 %), 1 vehicle per cycle from cross street to arterial (N = 10 %)
Σ Link length = 200 ft. (up to 10 vehicles can be stored)
Σ offset 2 and offset 3 = 0 to 60 seconds (15 seconds interval)
Σ Warm-up period = 10 signal cycles (10*75 = 750 seconds)
Σ Average vehicle space headway = 20 ft.
Σ Intersection width = 40 ft. (E-W), 60 ft. (S-N)

Results

Offset i is defined as the time difference between the initiation of green at intersection 1 and at intersection i. Offset 1 is fixed at zero and as discussed in the previous section, the intervals of offsets 2 and 3 are 15 seconds. Since the cycle length is 75 seconds, offsets 2 and 3, which have 25 combinations, vary from 0 to 60 seconds. Table 5.7-b shows a summarized result when a cycle length is 75 seconds.

To examine the effect of the number of simulated vehicles on the result, the number of vehicles simulated is reduced from 2000 to 1200. As shown in Table 5.11, the overall pattern formed in horizontal, vertical, and diagonal directions is very similar to that in Table 5.7-b. This implies that if the start-up time is determined correctly, the result is not sensitive to changes of the network crossing time.

TABLE 5.11: NETWORK CROSSING TIME (SEC) [FOR 1200 VEHICLES, L = 200 FT]

off2 off3	0	15	30	45	60
0	751	811	1082	1158	1935
15	2155	761	815	1054	1237
30	1310	2154	865	907	1149
45	1089	1306	2154	911	985
60	901	1151	1310	2154	854

Table 5.12 summarizes the number of vehicles simulated on each lane when offset 3 is zero and offset 2 varies. Table 5.13 shows the number of vehicles simulated on each arterial lane per 55 second green interval and offsets as in Table 5.12. Since up to 29 vehicles can be processed on each arterial lane during a 55 second green interval, with simultaneous greens, this offset combination is most efficient. When offsets 2 and 3 are zero, network crossing time is minimum and the number of vehicles simulated on each arterial lane per cycle is maximum. When offset 2 is 15 and offset 3 is 0, network crossing time is maximum and the number of vehicles simulated on arterial lanes per cycle is minimum. As offset 2 increases past 15 seconds, system efficiency increases. The more vehicles are processed on the arterial during the arterial green interval, the smaller network crossing time (the more efficient system performance) is produced. Therefore, system efficiency is proportional to the number of arterial vehicles simulated per cycle.

TABLE 5.12: NUMBER OF VEHICLES SIMULATED ON EACH LANE (FOR 2000 TOTAL VEHICLES)

offset		sim'n	lane number							
off2	off3	per'd	1	2	5	6	7	8	9	10
0	0	1261	510	510	164	162	164	162	164	162
15	0	3581	48	48	478	0	470	0	478	478
30	0	2174	261	290	290	0	290	289	290	290
45	0	1821	432	349	247	0	240	240	247	245
60	0	1501	520	480	200	0	200	200	200	200

TABLE 5.13: NUMBER OF VEHICLES SIMULATED ON EACH ARTERIAL LANE PER CYCLE (FOR 2000 TOTAL VEHICLES)

offset (sec)		simulation	arterial lane	
off2	off3	period (sec)	right	left
0	0	1261	29	29
15	0	3581	1	1
30	0	2174	9	9
45	0	1821	18	13
60	0	1501	26	23

Case 1. Simultaneous green (off2 = off3 = 0)

Simultaneous green produces the minimum 1261 second network crossing time. No queue spillback occurs in any lane throughout the simulation. A 3.5 seconds delay in the left lane of each cross street due to turning vehicles from the arterial indicates a queue spillback danger but none occurred.

Case 2. (off2 = 15 and off3 = 0)

This offset combination gives the maximum network crossing time of 3581 seconds indicating poor performance. The arterial offsets seem to initiate cross street queue spillback which, in turn, precipitates arterial queue spillback and system failure. As shown in Table 5.12, both arterial lanes accommodate only 48 vehicles during this network crossing time due to continuous cross street queue spillback. Since the right arterial lane is blocked by queue spillback from the left lane of the second cross street during most of the arterial green interval (52 out of 55 seconds), only one vehicle can be released per green interval. By the chain effect, the right lane of the first cross street is continuously blocked by queue spillback from the right arterial lane. Therefore no vehicle can be processed in this lane throughout the simulation. Queue spillback from the left lane of the third cross street blocks the left arterial lane during the most of the arterial green interval (52 out of 55 seconds). By the same chain effect, the right lane of the second cross street is continuously blocked by queue spillback from the left arterial lane. Also no vehicle can be processed in this lane throughout the network crossing time.

Long cycle length and long link length

The same input parameters as the previous section are used except 105 second signal cycles and 600 feet link lengths replace the 75 second cycles and 200 feet links. Parameters used are the following:

Σ Cycle length = 105 seconds (85 seconds arterial green, 20 seconds arterial red)

Σ Link length = 600 ft. (up to 30 vehicles can be stored)

Σ Turning movements = 6 vehicles/cycle from arterial to cross street (N = 20 %), 3 vehicle/cycle from cross street to arterial (N = 10 %)

Σ offset 2 and offset 3 = 0 to 90 seconds (15 seconds increment)

Σ Warm-up period = 10 cycles (10*105 = 1050 seconds)

Results

For the 105 second cycle, Table 5.8-d summarizes results which have the familiar diagonal pattern. To examine the effect of the number of simulated vehicles when links are longer (600 feet), the number of vehicles simulated is again reduced from 2000 to 1200. As shown in Table 5.14, like the 200 foot link case, the overall network crossing time pattern is very similar to that in Table 5.8-d.

TABLE 5.14: NETWORK CROSSING TIME (SEC) [FOR 1200 VEHICLES, L = 600 FT]

off2 off3	0	15	30	45	60	75	90
0	927	1141	1223	1503	2522	2900	829
15	822	921	1047	1262	1923	2184	2901
30	2819	823	912	1100	1325	1925	2480
45	2467	2693	872	947	1139	1318	1923
60	1917	2470	2442	870	977	1139	1318
75	1494	1902	2510	2832	868	961	1137
90	1047	1340	1662	1998	2845	834	942

The number of vehicles simulated during the warm-up period (10 cycles) is presented in Table 5.15. As indicated in the table, the number of vehicles processed during warm-up varies with offset combination much like the network crossing times shown in Table 5.14. As the efficiency of an arterial system increases, the number of vehicles processed during warm-up increases.

TABLE 5.15: NUMBER OF WARM-UP VEHICLES

off2 off3	0	15	30	45	60	75	90
0	1445	1309	1262	1209	1047	1032	1536
15	1576	1490	1374	1294	1095	1033	959
30	928	1586	1491	1348	1211	1089	974
45	894	959	1378	1300	1189	1111	998
60	964	873	947	1390	1290	1206	1126
75	1097	1002	958	944	1413	1328	1237
90	1339	1195	1110	1087	1012	1512	1402

Table 5.16 summarizes the number of vehicles simulated on each lane when offset 3 is zero and offset 2 varies from zero to 90 seconds. The number of vehicles simulated on each arterial lane per 85 second green interval is presented in Table 5.17. The combination of 15 second offset 2 and 0 second offset 3 is most efficient since the maximum 45 vehicles can be processed on each arterial lane per 85 second green interval. When offset 2 is 30 and offset 3 is 0, network crossing time is maximum and the number of vehicles simulated per arterial lane per cycle is minimum. As offset 2 increases past 30 seconds, system efficiency increases and the number of vehicles simulated on the arterial per cycle increases. As in the case of a shorter link, system efficiency is proportional to the number of vehicles simulated on the arterial per cycle. Although relational patterns between offsets 2 and 3 are similar for 200 and 600 feet link cases, the offset 2 magnitude for maximum efficiency, seems to increase with link length. This trend is, no doubt, related to greater link travel times for 600 versus 200 feet links.

TABLE 5.16: NUMBER OF VEHICLES SIMULATED ON EACH LANE (FIXED OFF3)

offset		sim'n	lane number							
off2	off3	per'd	1	2	5	6	7	8	9	10
0	0	1533	557	603	140	140	140	140	140	140
15	0	1375	590	630	130	130	130	130	130	130
30	0	4722	90	112	450	0	448	0	450	450
45	0	4147	197	224	398	0	390	0	396	395
60	0	3179	390	396	307	0	300	0	305	302
75	0	2532	504	515	247	6	240	0	247	241
90	0	1750	485	535	162	168	160	160	162	168

TABLE 5.17: NUMBER OF VEHICLES SIMULATED ON EACH ARTERIAL LANE PER CYCLE

offset (sec)		simulation	arterial lane	
off2	off3	period (sec)	right	left
0	0	1533	38	38
15	0	1375	45	45
30	0	4722	2	2
45	0	4147	5	5
60	0	3179	13	13
75	0	2532	21	21
90	0	1750	29	29

Case 1. (off2 = 15 and off3 = 0)

For 600 feet link lengths, this offset combination was most efficient, however, moderate duration queue spillback did occur. Each left lane of the three cross streets was blocked by queue spillback for 12.5 seconds out of the 20 second cross street green interval. These queue spillbacks were cleared during the same cross street green. Therefore, both arterial lanes are not affected by queue spillbacks. Only four vehicles (out of a maximum of ten) from the upstream link are processed in each left cross street lane at the second intersection in a cycle throughout the network crossing time. The other six vehicles turn from the arterial into each cross street.

Case 2. (off2 = 30 and off3 = 0)

This offset combination produces the maximum network crossing time of 4722 seconds. The left lane of the first and third intersection and the right lane of the second intersection on the arterial are blocked by queue spillbacks for 2 seconds during the cross street green and 80 seconds during the arterial green throughout the network crossing time. Furthermore, the right lanes of the first and second cross streets are continuously blocked by arterial queue spillbacks during cross street green. Therefore no vehicle can be released in these lanes throughout the simulation.

5.4.4 Offset and Link Length

The experimental design, featuring reduced condition combinations, permits easy examination of specific effects of control parameters. To examine the effect of offset on the network crossing time more closely, a smaller offset 2 increment, or two seconds, and zero

values of offset 1 and offset 3, were used. A two second offset 3 increment and zero values of offset 1 and offset 2 were also used. To look into the relationship between offset and link length, a 400 foot link case was added.

As shown in Figures 5.17 - 5.20, given one link length, graph shapes are very similar for all cycle lengths. As discussed before, with the same link length and fixed cross street green, maximum network crossing time increases as the arterial green increases. In other words, with worst offsets and fixed cross street greens, delay caused by queue spillback increases as arterial green increases. "Lost green" per cycle due to queue spillback is likely to increase as the arterial green interval increases. "Lost green" is defined as a green duration that cannot be used, and is therefore wasted, due to queue spillback.

As presented in Figures 5.17 and 5.18, when links are 200 feet long and offset 2 is about 10 seconds greater than offset 3, network crossing times are maximum. However simultaneous green produces minimum network crossing time regardless of the cycle length. This result is almost identical to that of the experimentation with 200 feet links conducted in subsection 5.4.1.

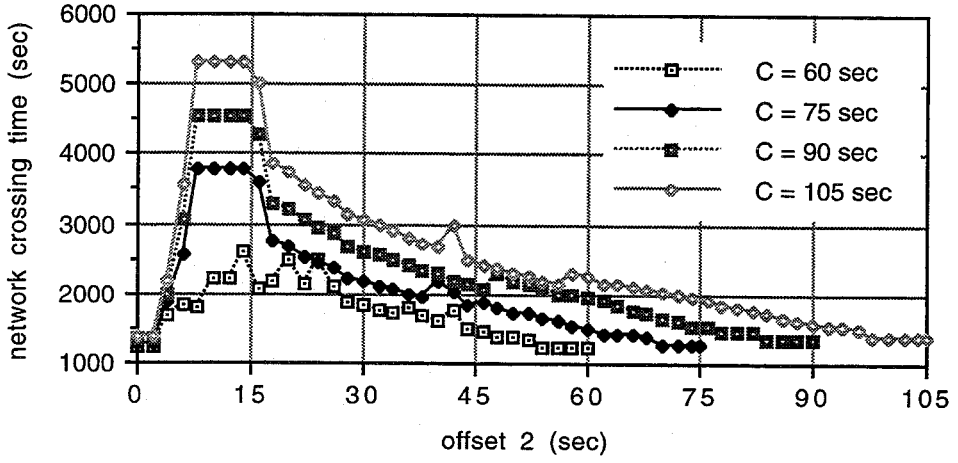


Figure 5.17: Offset 2 vs. network crossing time (one-way, L = 200 ft, offset 3 = 0)

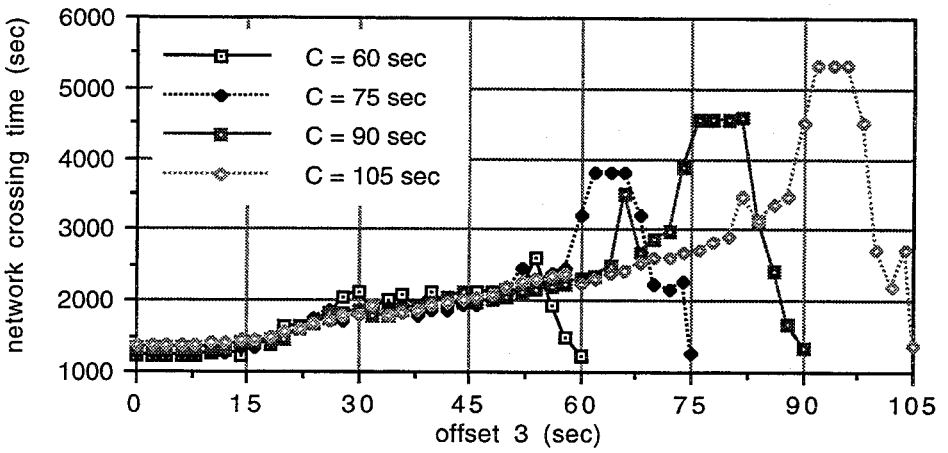


Figure 5.18: Offset 3 vs. network crossing time (one-way, L = 200 ft, offset 2 = 0)

When links are 600 feet long and offset 2 is about 15 seconds greater than offset 3, network crossing times are minimum, however when offset 2 is about 35 seconds greater than offset 3, network crossing times are maximum. This result is also almost identical to that of the experimentation with 600 foot links conducted in subsection 5.4.1.

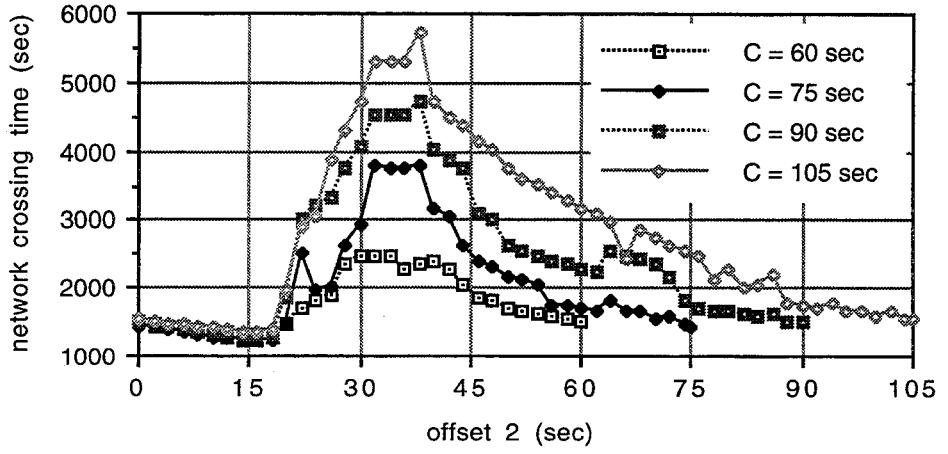


Figure 5.19: Offset 2 vs. network crossing time (one-way, L = 600 ft, offset 3 = 0)

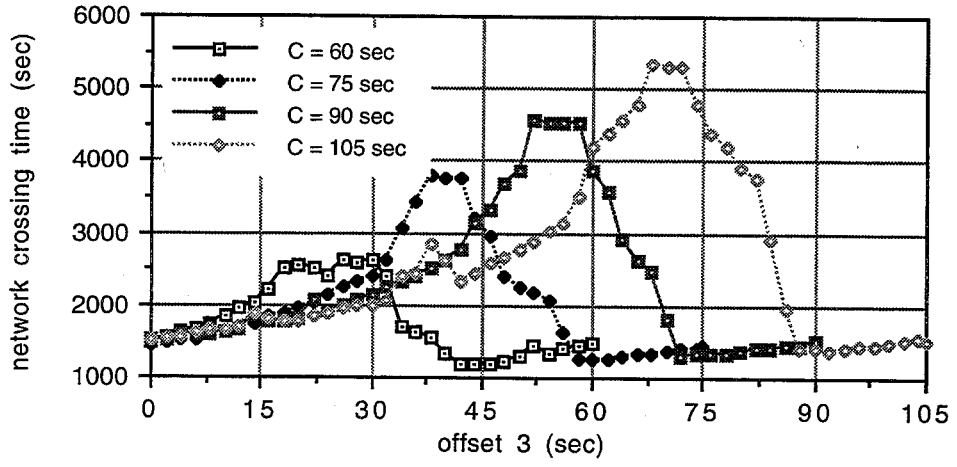


Figure 5.20: Offset 3 vs. network crossing time (one-way, L = 600 ft, offset 2 = 0)

Desirable and undesirable offset ranges producing minimum or maximum network crossing times were obtained for each cycle length and each link length as shown in Tables 5.18 and 5.19. The last row of each table indicates the median value for each offset range.

TABLE 5.18: DESIRABLE OFFSET RANGES (ONE-WAY OPERATION)

L (ft)	200		400		600	
	off 3 = 0	off 2 = 0	off 3 = 0	off 2 = 0	off 3 = 0	off 2 = 0
C (sec)	off 2 (sec)	off 3	off 2	off 3	off 2	off 3
60	(-6) thru 2	0 thru 8	NA	(-12) thru (-4)	14 thru 16	(-18) thru (-14)
75	(-5) thru 2	0 thru 12	2 thru 12	(-15) thru (-3)	14 thru 18	(-17) thru (-13)
90	(-6) thru 2	0 thru 12	2 thru 12	(-12) thru (-2)	14 thru 18	(-18) thru (-12)
105	(-7) thru 2	0 thru 10	2 thru 12	(-11) thru (-3)	14 thru 16	(-17) thru (-13)
Med	-2	5	7	-8	16	-15

TABLE 5.19: UNDESIRABLE OFFSET RANGES (ONE-WAY OPERATION)

L (ft)	200		400		600	
	off 3 = 0	off 2 = 0	off 3 = 0	off 2 = 0	off 3 = 0	off 2 = 0
C (sec)	off 2	off 3	off 2	off 3	off 2	off 3
60	14	-6	NA	NA	30 thru 34	(-34) thru (-30)
75	8 thru 14	(-13) thru (-9)	22 thru 26	(-23) thru (-21)	32 thru 38	(-37) thru (-33)
90	8 thru 14	(-14) thru (-8)	22 thru 24	(-24) thru (-22)	32 thru 38	(-38) thru (-32)
105	8 thru 14	(-13) thru (-9)	22 thru 24	(-25) thru (-21)	32 thru 38	(-37) thru (-33)
Med	12	-10	23	-23	34	-34

When links are 200 feet long, all desirable offset ranges include zero. For the 400 foot link case, network crossing time is minimum when offset 2 is 7 seconds greater than offset 3 and the 600 foot link best case occurs when offset 2 is 16 seconds greater than offset 3. Therefore, there exists a relationship among link length, offset and network crossing time. The offset 2 value producing minimum network crossing time (maximum efficiency) increases about nine seconds for each 200 feet link length. This change in offset 2, or 9 seconds, is approximately equal to the travel time of the increased 200 feet distance (200/18.35 = 10.9 seconds).

To maintain maximum efficiency, as link length increases, downstream intersection greens should begin earlier than that of upstream intersections by an increasing amount. This offset is defined as the *network throughput offset*. This finding is quite different from traditional arterial progression in which upstream intersection greens begin earlier than downstream intersection greens so that platoons of vehicles can pass downstream intersections without stopping. This can be defined as the *travel time offset*.

5.4.5 Green Split and Cycle Length

Within the experimental design green split and cycle length change simultaneously, therefore easy examination of their individual effects on the performance measure is not provided. In order to identify the effect of green split and cycle length, either the green split or the cycle length, not both, should be varied. To reduce computational complexity, the best and worst offset combinations were used. With a demand of 500 vehicles per hour per lane (vphpl), the cross street traffic becomes undersaturated as the g_C/g ratio (g_C: cross street green interval, g: arterial green interval) increases. For comparison purposes, a heavy cross street traffic demand (1800 vphpl) was added.

For a 200 foot link

Table 5.20 and Figures 5.21 - 5.24 present results of simulation runs, which examine the effect of green split on network crossing time, when links are 200 feet long. Given a fixed link length, cross street traffic demand, and offset combination, different cycle lengths produce very similar relationships between network crossing time and g_c/g ratio. However, the relationships differ as cross street traffic demand changes.

When the cross street traffic demand is 500 vphpl, an optimum g_c/g ratio, producing minimum network crossing time, can be identified. When the g_c/g ratio is larger than optimum, the cross street green is not fully used, but when the g_c/g ratio is smaller than optimum, cross street traffic demand exceeds capacity. Therefore, system efficiency decreases as the g_c/g ratio increases or decreases past the optimum.

When the cross street traffic demand is very heavy, 1800 vphpl, network crossing time decreases as the g_c/g ratio increases regardless of offset combination. Since the arterial system consists of two arterial lanes and six cross street lanes, cross streets have more influence on system efficiency than the arterial. The capacity of cross streets is proportional to the g_c/C ratio (C: cycle length). With a fixed cycle length, system efficiency increases as the g_c/g ratio increases. Therefore, in oversaturated conditions, sufficiently long cross street green intervals should be assigned to increase system efficiency and prevent avoidable queue spillbacks.

If cross street traffic demand is moderate, 500 vphpl, and best offsets are applied, network crossing time changes only slightly with g_c/g ratio and cycle length, except when the cross street green interval is very short. However, system efficiency rapidly deteriorates when the cross street green becomes too short, 10 seconds here, for the link length. Therefore, with moderate cross street traffic demand, a "practical" minimum cross street green interval is necessary to accommodate through traffic and turning vehicles from the arterial. When the cross street green is shorter than the minimum, even with the best offset combination, queue spillback occurs in the cross streets and system efficiency deteriorates. For 200 foot links, the minimum cross street green interval is 11 seconds .

TABLE 5.20: THE EFFECT OF GREEN SPLIT ON THE NETWORK CROSSING TIME (L = 200 FT)

g	gc	gc/g	with best offsets 1)		with worst offsets 2)	
			500 3)	1800	500	1800
a. C = 60 sec						
25	35	1.40	1426	887	1451	1713
30	30	1.00	1366	994	1529	2258
35	25	0.71	1285	1057	1456	2276
40	20	0.50	1220	1161	2501	2861
45	15	0.33	1340	1340	4003	4003
50	10	0.20	2122	2151		
b. C = 75 sec						
35	40	1.14	1409	930	1601	1826
40	35	0.88	1337	997	1452	2148
45	30	0.67	1292	1101	2209	2828
50	25	0.50	1238	1158	2687	2848
55	20	0.36	1261	1261	3581	3581
60	15	0.25	1402	1402	5008	5008
65	10	0.15	2176	2228		
c. C = 90 sec						
40	50	1.25	1442	902	1471	1677
45	45	1.00	1410	975	1403	1840
50	40	0.80	1335	1010	1485	2201
55	35	0.64	1288	1075	1321	2583
60	30	0.50	1232	1160	2850	3398
65	25	0.38	1243	1239	3482	3418
70	20	0.29	1336	1336	4301	4301
75	15	0.20	1458	1458	6013	6013
80	10	0.13	2290	2342		
d. C = 105 sec						
50	55	1.10	1431	931	1607	1727
55	50	0.91	1366	998	1401	1962
60	45	0.75	1335	1036	1328	2155
65	40	0.62	1296	1088	1311	2576
70	35	0.50	1255	1148	2386	3018
75	30	0.40	1245	1238	3324	3968
80	25	0.31	1269	1276	3982	3988
85	20	0.24	1388	1388	5021	5021
90	15	0.17	1508	1506	7018	7018
95	10	0.11	2326	2306		

1) offset 1 = offset 2 = offset 3 = 0 sec

2) offset 1 = 0 sec, offset 2 = 15 sec, offset 3 = 0 sec

3) hourly cross street traffic demand in vphpl

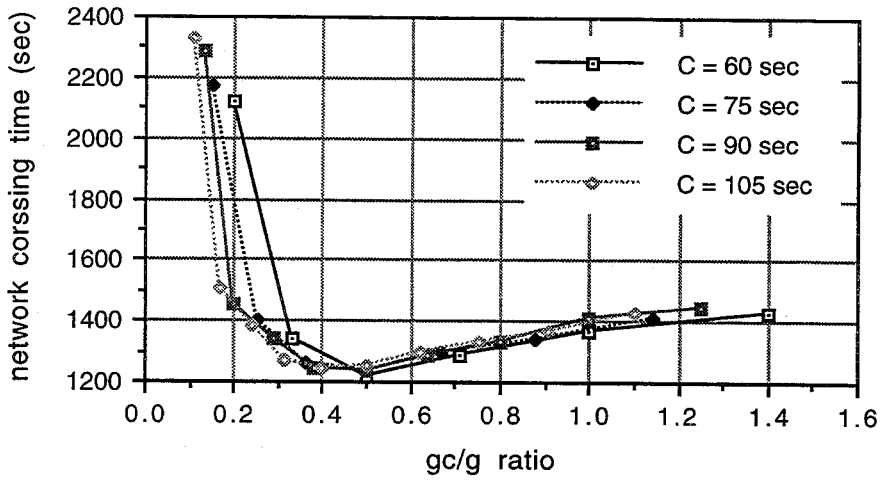


Figure 5.21: The effect of green split on the network crossing time (L = 200 ft, cross street V = 500 vphpl, with best offsets)

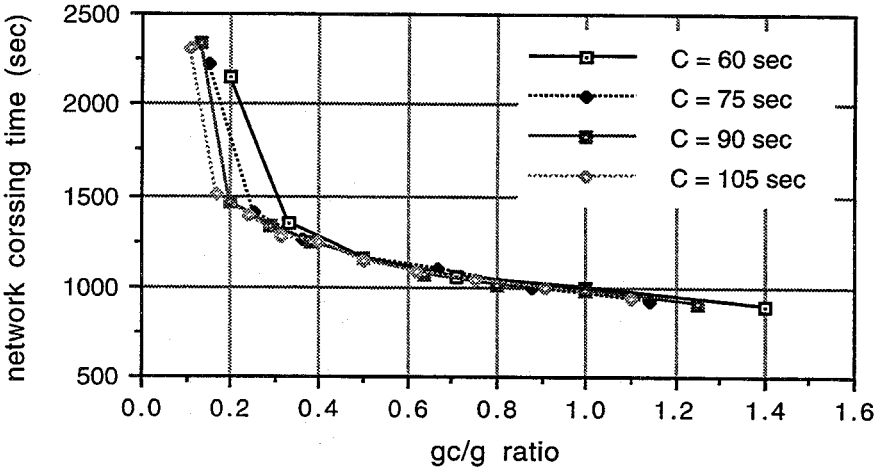


Figure 5.22: The effect of green split on the network crossing time (L = 200 ft, cross street V = 1800 vphpl, with best offsets)

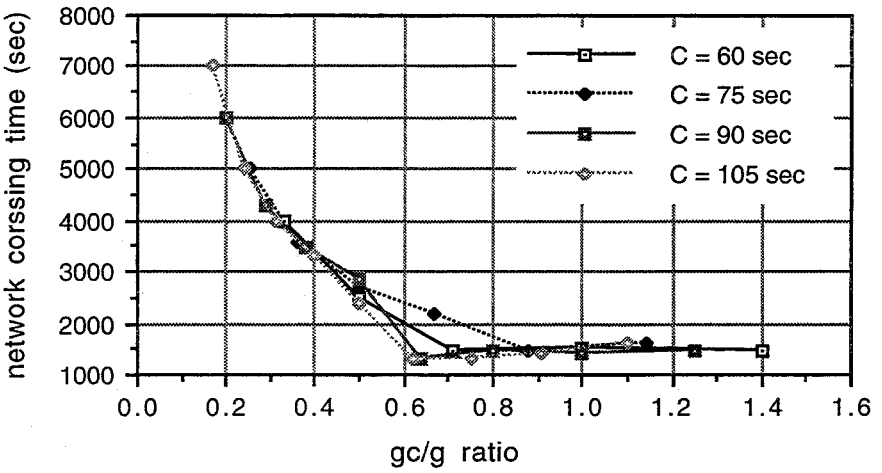


Figure 5.23: The effect of green split on the network crossing time (L = 200 ft, cross street V = 500 vphpl, with worst offsets)

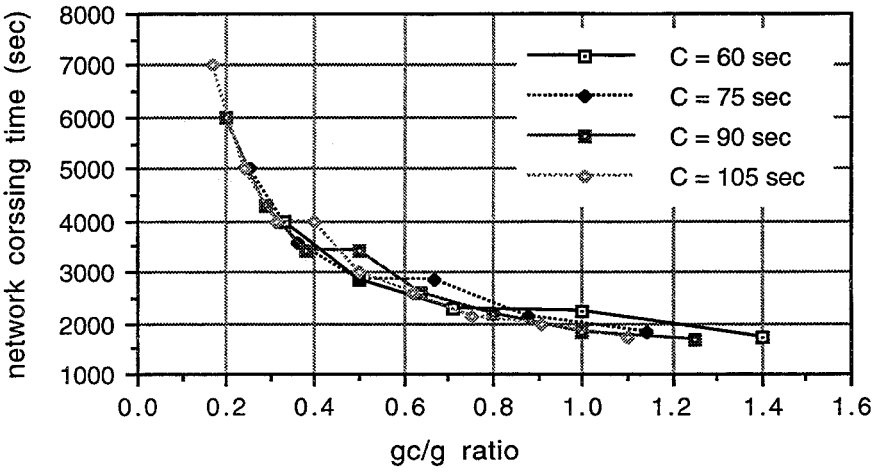
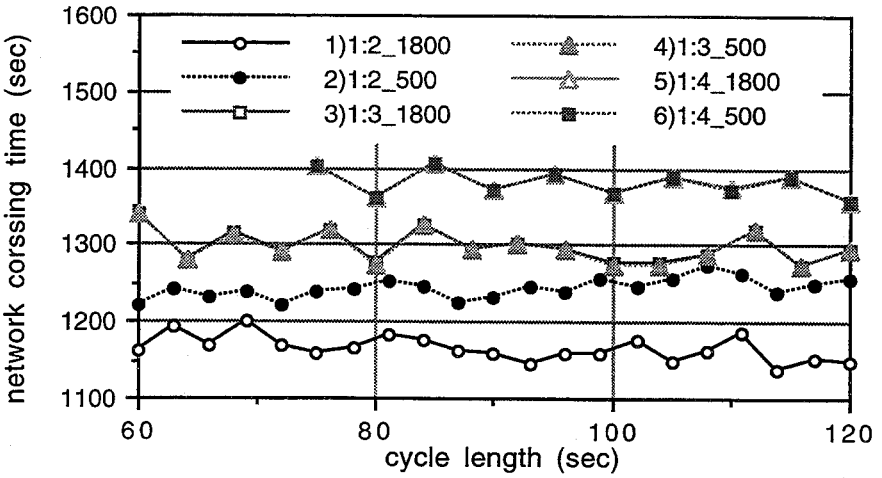


Figure 5.24: The effect of green split on the network crossing time (L = 200 ft, cross street V = 1800 vphpl, with worst offsets)

Figures 5.25 and 5.26 show the effect of cycle length on network crossing time with the best and worst offset combinations respectively when links are 200 feet long. As explained previously, network crossing time increases as the g_c/g ratio decreases. When the g_c/g ratio is 1:3 or 1:4, the cross street traffic demand exceeds the capacity whether it is 500 vphpl or 1800

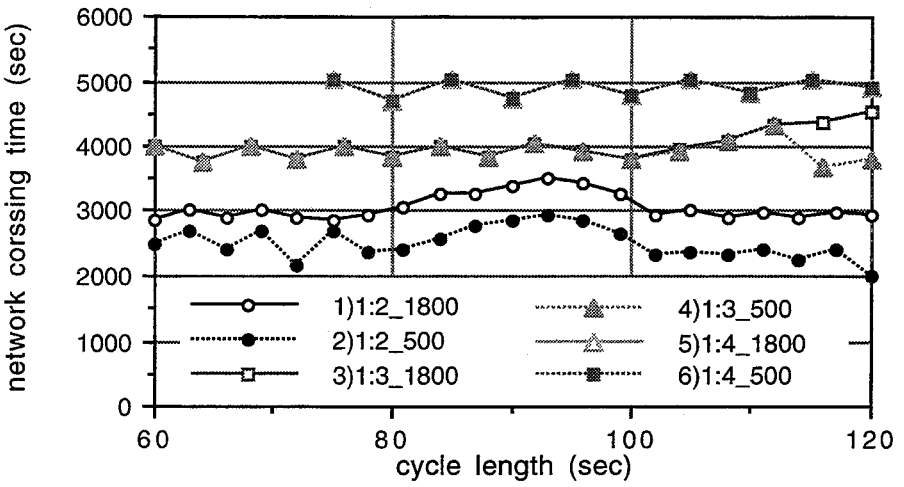
vphpl. Therefore, in Figures 5.25 and 5.26, series 3 and 4 are almost identical and at a different demand level series 5 and 6 are also nearly identical.

When the g_C/g ratio is 1:2 and the demand is 500 vphpl, the cross street is not saturated. Since the cross street green is not fully used, when the best offsets were applied, this case is less efficient than the one with the demand at 1800 vphpl as shown by series 1 and 2 in Figure 5.25. When the worst offsets were applied, the case with 500 vphpl demand is more efficient than the one with 1800 vphpl demand as shown by series 1 and 2 in Figure 5.26. This is because the delay resulting from queue spillbacks caused by wrong offsets has more effect on the network crossing time than the under utilized green time.



* Cross street to arterial street green ratio (g_C:g) _ Cross street traffic demand (vphpl)

Figure 5.25: The effect of cycle length on the network crossing time (L = 200 ft, with best offsets)



* Cross street to arterial street green ratio (g_C:g) _ Cross street traffic demand (vphpl)

Figure 5.26: The effect of cycle length on the network crossing time (L = 200 ft, with worst offsets)

For a 600 foot link

Table 5.21 and Figures 5.27 - 5.30 show the effect of green split on network crossing time, when links are 600 feet long. In general, results are almost identical to those of the 200 foot link case. With the same link length, cross street traffic demand, and offset combinations, the relationships between the network crossing time and g_C/g ratio are very similar regardless of cycle length. However, the relationships are different at different cross street traffic demand levels.

As in the 200 foot link case, when the cross street traffic demand is 500 vphpl, each cycle length seems to have an optimum g_C/g ratio. When the cross street traffic demand is 1800 vphpl, the network crossing time decreases as the g_C/g ratio increases regardless of the offset combinations.

With the best offsets, network crossing time changes little as the g_C/g ratio changes, irrespective of cycle length, except when the cross street green interval decreases to 14 seconds causing significant system efficiency deterioration. The practical minimum green interval for cross streets is 15 seconds for the 600 foot link, which is 4 seconds longer than that for the 200 foot link case. The difference can be mainly attributed to the link length difference and the difference in the number of turning vehicles from arterial to cross streets (6 for a 600 foot link and 2 for a 200 foot link).

TABLE 5.21: THE EFFECT OF GREEN SPLIT ON THE NETWORK CROSSING TIME (L = 600 FT)

g	gc	gc/g	with best offsets 1)		with worst offsets 2)	
			500 3)	1800	500	1800
a. C = 60 sec						
25	35	1.40	1276	984	1286	1435
30	30	1.00	1224	1059	1262	1688
35	25	0.71	1209	1102	1577	1861
40	20	0.50	1245	1219	2464	2461
45	15	0.33	1452	1452	3328	3389
46	14	0.30	2348	2350		
b. C = 75 sec						
35	40	1.14	1241	952	1295	1614
40	35	0.88	1180	1025	1229	1768
45	30	0.67	1227	1097	1622	2007
50	25	0.50	1195	1129	2093	2207
55	20	0.36	1268	1268	2956	2956
60	15	0.25	1911	1911	3861	3861
61	14	0.23	3063	3754		
c. C = 90 sec						
40	50	1.25	1298	927	1285	1624
45	45	1.00	1258	969	1268	1822
50	40	0.80	1216	1000	1203	2097
55	35	0.64	1159	1060	1202	2252
60	30	0.50	1217	1145	2293	2652
65	25	0.38	1226	1221	2755	2749
70	20	0.29	1325	1325	4052	4052
75	15	0.20	1501	1501	5187	5569
76	14	0.18	3700	3704		
d. C = 105 sec						
50	55	1.10	1305	916	1294	1885
55	50	0.91	1261	985	1262	2056
60	45	0.75	1214	1024	1201	2098
65	40	0.62	1209	1077	1192	2353
70	35	0.50	1212	1133	2359	2774
75	30	0.40	1232	1333	3152	3250
80	25	0.31	1257	1256	3337	3337
85	20	0.24	1375	1375	4722	4722
90	15	0.17	1546	1546	6049	6049
91	14	0.15	4317	4321		

1) offset 1 = 0 sec, offset 2 = 15 sec, offset 3 = 0 sec

2) offset 1 = 0 sec, offset 2 = 30 sec, offset 3 = 0 sec

3) hourly cross street traffic demand in vphpl

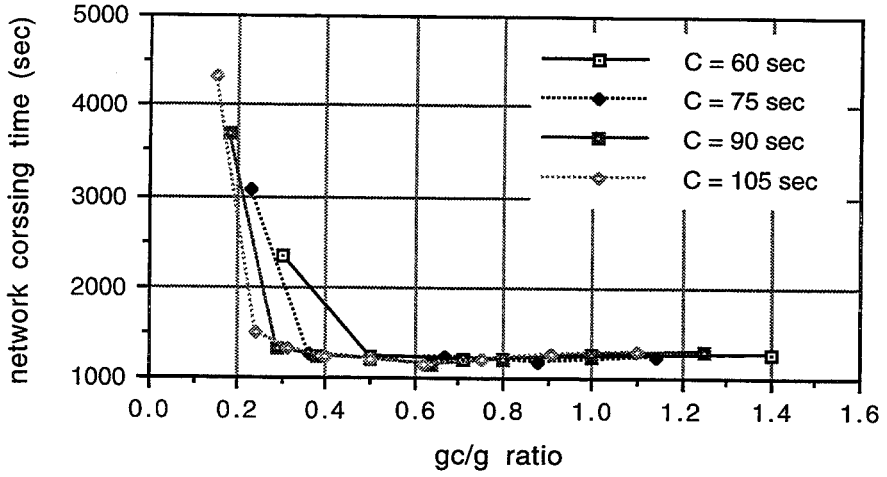


Figure 5.27: The effect of green split on the network crossing time (L = 600 ft, cross street V = 500 vphpl, with best offsets)

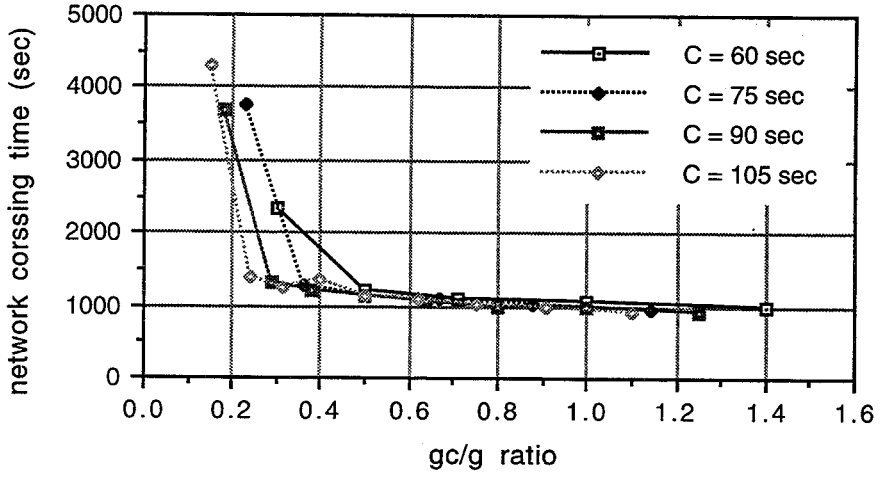


Figure 5.28: The effect of green split on the network crossing time (L = 600 ft, cross street V = 1800 vphpl, with best offsets)

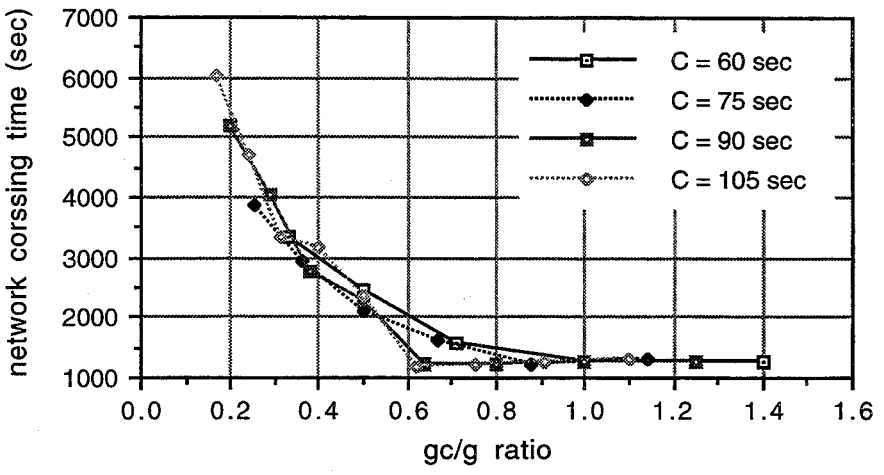


Figure 5.29: The effect of green split on the network crossing time (L = 600 ft, cross street V = 500 vphpl, with worst offsets)

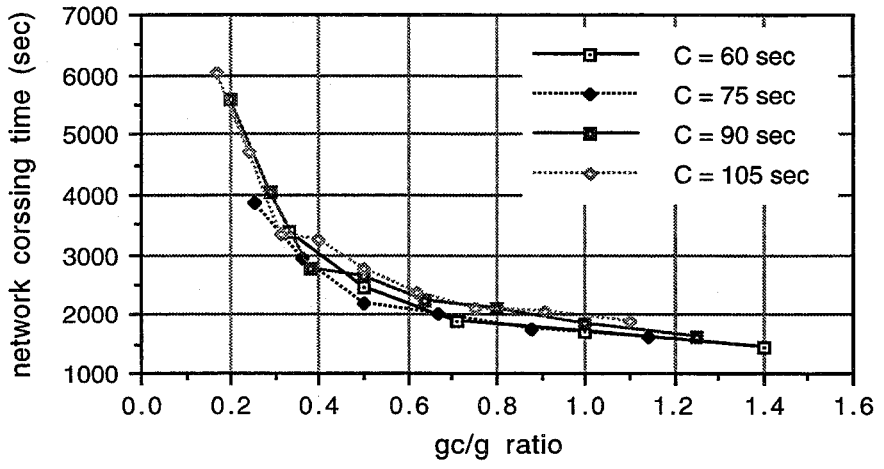
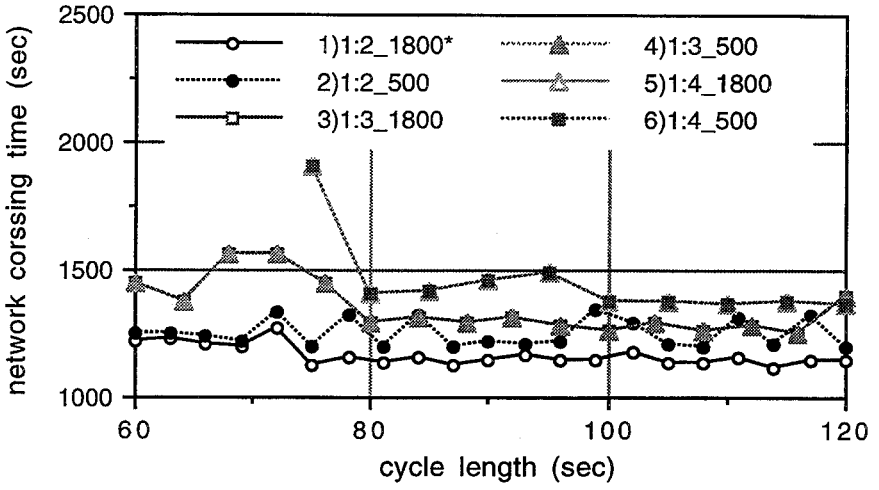


Figure 5.30: The effect of green split on the network crossing time (L = 600 ft, cross street V = 1800 vphpl, with worst offsets)

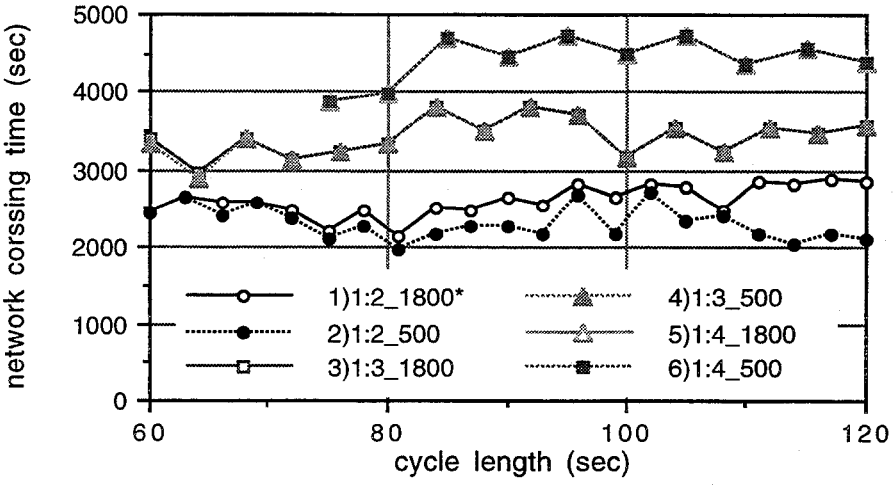
Figures 5.31 and 5.32 show the effect of cycle length on network crossing time with the best and worst offset combinations respectively when links are 600 feet long. In general, patterns are similar to those of the 200 foot link case. Network crossing time varies only slightly as the cycle length changes unless the green interval is too short. As shown in Figure 5.31 (see the

shortest cycle length cases of series 5 - 6), critically short cross street greens (15 seconds) for the link length (600 feet) produces much longer network crossing times. When the g_C/g ratio is 1:3 or 1:4, the cross street traffic demand exceeds the capacity whether it is 500 vphpl or 1800 vphpl. Series 3 and 4 as well as series 5 and 6 in Figures 5.31 and 5.32 are almost identical patterns but at different demand levels.



* Cross street to arterial street green ratio (g_C:g) _ Cross street traffic demand (vphpl)

Figure 5.31: The effect of cycle length on the network crossing time (L = 600 ft, with best offsets)



* Cross street to arterial street green ratio (g_C:g) _ Cross street traffic demand (vphpl)

Figure 5.32: The effect of cycle length on the network crossing time (L = 600 ft, with worst offsets)

5.4.6 Formulation of Relationships

In the previous subsections, the effect of offset, link length, green split and interval, and cycle length, on the main performance measure, network crossing time, was examined. Each analysis on each input parameter, however, does not show the overall relationship between traffic control parameters and the response variable. Multiple regression analysis was used to draw a relationship between these parameters and network crossing time. Tables 5.7 and 5.8 were used as data for the multiple regression. Results of regression analysis were summarized in Table 5.22.

The following regression equation was proposed:

$$NCT = b_1 + b_2\, C + b_3\, GCR + b_4\, OFFD \qquad (5.11)$$

where,

NCT = network crossing time (sec)

C = cycle length (sec)

GCR = g_C/g (:green ratio, g_C = cross street green, g = arterial green)

OFFD = off3 - off2 (if OFFD < 0, OFFD = OFFD + C)

The t-statistic for the coefficient of the OFFD is 20.83, which would be more than enough to reject the null hypothesis $b_4 = 0$ at conventional significance levels ($t_{table} = 1.98$ at 0.05 level). The coefficients of C and GCR, however, do not achieve significance at the 0.05 level (t-statistics are -0.64 and -0.74, respectively), that is, they are not significantly different from zero. The estimated t values were computed under the null hypothesis that the true population value of each regression coefficient is zero. Therefore, only the estimated regression coefficient of OFFD is statistically significant.

For the 600 foot link case, a different definition of the variable OFFD was used as follows:

OFFD = off3 - off2 + 15 (if OFFD < 0, OFFD = OFFD + C,

if OFFD ≥ C, OFFD = OFFD - C) (5.12)

The t-statistic for the coefficient of this OFFD is 34.03, which is again more than enough to reject the null hypothesis $b_4 = 0$ at conventional significance levels. As in the 200 foot link case, the coefficients of C and GCR are not significant at the 0.05 level (t-statistics are 0.87 and 0.61, respectively). Only the estimated regression coefficient of OFFD is statistically significant.

TABLE 5.22: RESULTS OF MULTIPLE REGRESSION ANALYSES

1. L = 200 ft			
Dependent variable:		network crossing time	
Number of observations		126	
R		0.8928	
R-spuared		0.7971	
Adjusted R-spuared		0.7921	
Standard error of estimate		441.76	
Intercept		2132.92	
Independent Variable	Estimated Coefficient	Standard Error	t-Statistic
C	- 6.80	10.624	- 0.640
GCR	- 1433.12	1926.453	- 0.744
OFFD	31.85	1.529	20.834
g-ratio = g_c/g		g_c = cross street green	
		g = arterial green	
OFFD = off3 - off2		if OFFD < 0, OFFD = OFFD + C	
2. L = 600 ft			
Dependent variable:		network crossing time	
Number of observations		126	
R		0.9559	
R-spuared		0.9137	
Adjusted R-spuared		0.9116	
Standard error of estimate		299.85	
Intercept		231.296	
Independent Variable	Estimated Coefficient	Standard Error	t-Statistic
C	6.30	7.211	0.874
GCR	791.07	1307.587	0.605
OFFD	35.31	1.038	34.028
OFFD = off3 - off2 + 15		if OFFD < 0, OFFD = OFFD + C	
		if OFFD ≥ C, OFFD = OFFD - C	

Since OFFD was the only statistically significant traffic control parameter for both cases, a curve fitting method instead of multiple regression analysis was used to draw a relationship between OFFD and network crossing time. As shown in Figures 5.33 and 5.34, the plotted data appear to have a relationship other than linear. Results indicate that second order equations fit the plotted data better than linear equations for both cases. The equations obtained by this method are as follows:

For a 200 foot link,

$$y = 1478.2 - 3.752x_1 + 0.442x_1^2 \quad (R^2 = 0.902) \qquad (5.13)$$

For a 600 foot link,

$$y = 1322.1 + 8.790x_2 + 0.332x_2^2 \quad (R^2 = 0.968) \qquad (5.14)$$

where

y = network crossing time

x_1 = offset 3 - offset 2, for a 200 foot link

(if $x_1 < 0$, $x_1 = x_1 + C$)

x_2 = offset 3 - offset 2 + 15, for a 600 foot link

(if $x_2 < 0$, $x_2 = x_2 + C$, if $x_2 \geq C$, $x_2 = x_2 - C$)

Figures 5.33 and 5.34 present results of the curve fitting and scattergrams that show the relationship between the network crossing time and OFFD when links are 200 and 600 feet long, respectively.

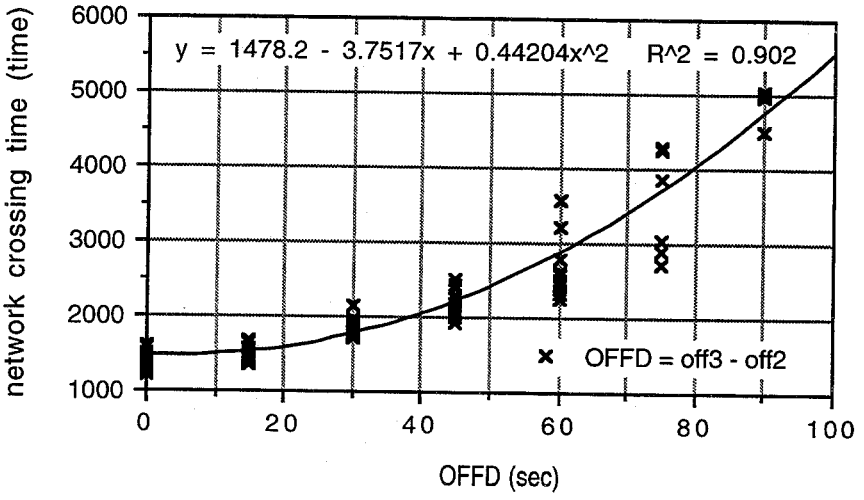


Figure 5.33: Relationship between simlation time and OFFD (L = 200 ft)

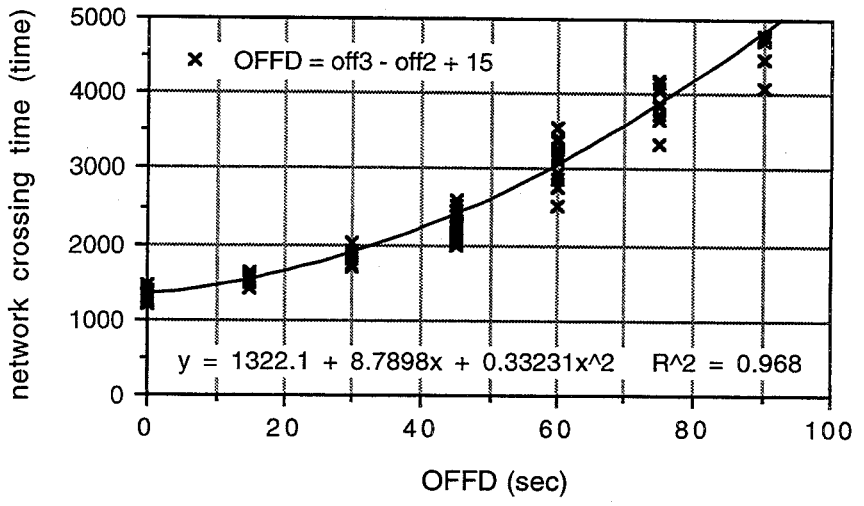


Figure 5.34: Relationship between simlation time and OFFD (L = 600 ft)

5.5 TWO-WAY ARTERIAL OPERATION

As shown in Figure 5.35, a two-way two-lane arterial consisting of three intersections with one-way two-lane cross streets is considered in this analysis. A center lane in the arterial is considered only when a protected left-turn phase on the arterial is included in a signal timing plan. A common traffic signal cycle length along the arterial and two- and three-phase operations are applied. For two-phase operation, a left-turn phase on the arterial is precluded. Three-phase operation includes an arterial protected left-turn phase.

5.5.1 Experimentation with Two-Phase Operation

With the experimental design derived in section 5.2, two-way and two-phase operation experimentation was conducted first. As in the one-way operation experimentation, two different link lengths (200 and 600 feet) and four different cycle lengths (60, 75, 90, and 105 seconds) were used. A system performance measure was the time required to process 2000 vehicles through the arterial system consisting of four arterial lanes and six cross street lanes.

Tables 5.23 and 5.24 summarize results of two-way and two-phase operation when links are 200 feet and 600 feet long respectively. For the same reason as in the one-way operation experimentation, 20 percent for turning movements into cross streets and 10 percent for turning movements into the arterial were used. Each table shows results of 126 simulation runs and the total design points are 252.

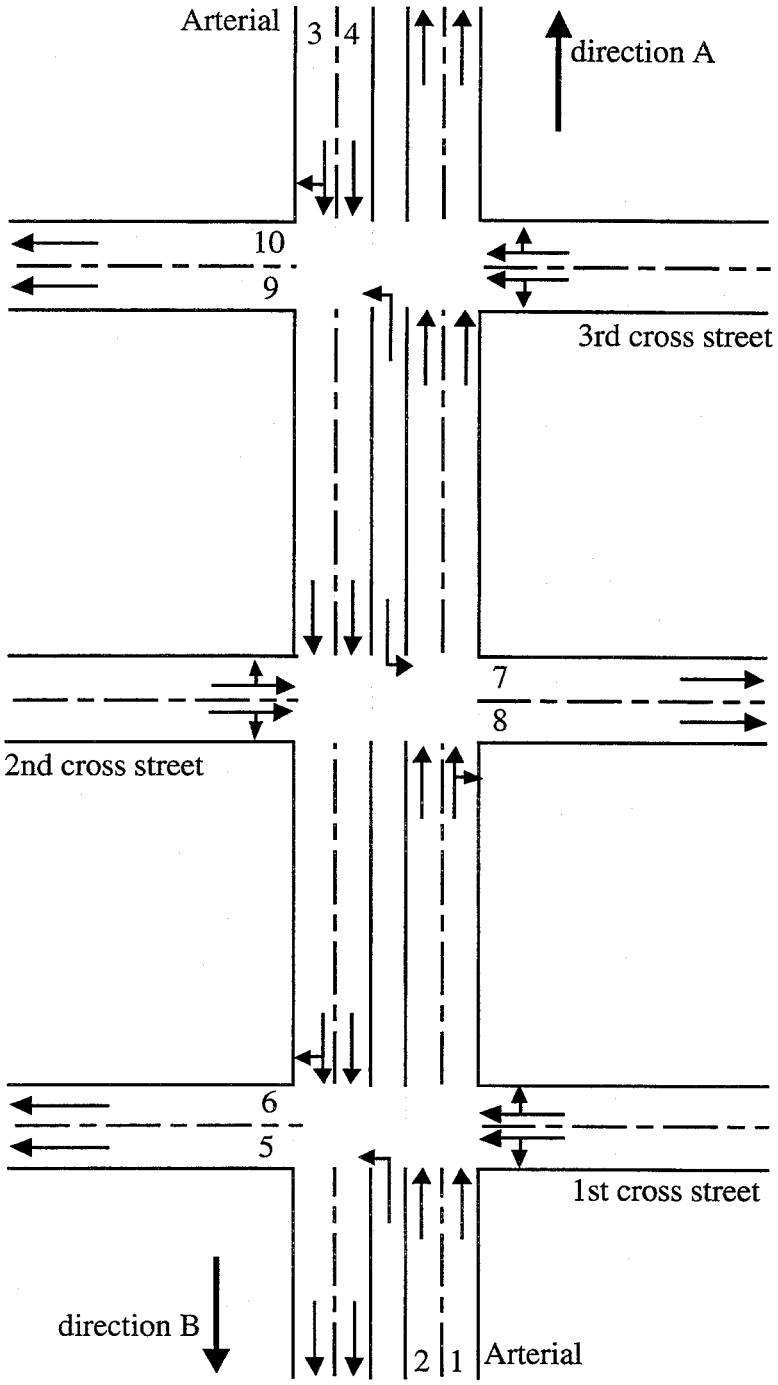


Figure 5.35: Arterial two-way operational configuration

With the two-way and two-phase operation, the difference between minimum and maximum network crossing times for each link length and cycle length is much smaller than that of one-way operation. This is because two more lanes (lanes 3 and 4, see Figure 5.35) are added to the arterial (in direction B) and a protected left-turn phase on the arterial is precluded. Precluding a left-turn on the arterial considerably reduces queue spillback possibility in each cross street left lane (lanes 5, 7, and 9), which in turn, enables arterial lanes 1 and 4 to accommodate more vehicles. Furthermore, traffic in arterial lanes 2 and 3 is uninterrupted by queue spillbacks from cross streets.

For a 200 foot link, the simultaneous green (zero offsets) produces minimum network crossing time irrespective of cycle length. This is identical to the one-way operation result. As absolute values of offsets 2 and 3 decrease to zero simultaneously, the network crossing time decreases. On the other hand, the network crossing time increases as absolute values of offsets 2 and 3 increase simultaneously.

Results for a 600 foot link are also different from the one-way operation results. As shown in Table 5.24, differences between minimum and maximum network crossing times is very small regardless of the cycle length, and compared to the 200 foot link case, differences between minimum and maximum network crossing times is smaller. As in the 200 foot link case, network crossing times decrease as absolute values of offsets 2 and 3 decrease simultaneously.

TABLE 5.23: NETWORK CROSSING TIME (SEC) [TWO-WAY, TWO-PHASE, L = 200 FT]

off3	0	15	30	45	60	75	90
off2	a. C = 60 sec. (g = 40, r = 20)						
0	897	974	1283	987			
15	1095	989	1054	1180			
30	1214	1210	1322	1306			
45	1053	1274	1052	974			
	b. C = 75 sec. (g = 55, r = 20)						
0	856	915	1092	1154	910		
15	1020	915	983	1140	1077		
30	1085	1077	1154	1144	1079		
45	1079	1079	1141	1092	1147		
60	983	1147	1144	990	913		
	c. C = 90 sec. (g = 70, r = 20)						
0	870	908	1102	1050	1101	909	
15	950	908	962	1137	1095	1042	
30	1096	1042	1101	1103	1224	1048	
45	1067	1048	1096	1050	1103	1050	
60	1058	1051	1219	1140	1049	1095	
75	945	1095	1103	1103	967	904	
	d. C = 105 sec. (g = 85, r = 20)						
0	879	911	1032	1074	1025	1057	912
15	977	911	953	1105	1110	1075	1003
30	1002	1003	1057	1069	1264	1176	1042
45	1060	1042	1075	1025	1081	1122	1042
60	1052	1042	1176	1116	1027	1077	1017
75	1013	1017	1109	1258	1110	1032	1067
90	955	1067	1075	1081	1069	962	915

TABLE 5.24: NETWORK CROSSING TIME (SEC) [TWO-WAY, TWO-PHASE, L = 600 FT]

off3	0	15	30	45	60	75	90
off2	a. C = 60 sec. (g = 40, r = 20)						
0	894	912	997	919			
15	938	919	934	1033			
30	981	917	936	963			
45	930	979	912	894			
	b. C = 75 sec. (g = 55, r = 20)						
0	855	846	918	932	855		
15	857	855	848	919	932		
30	941	872	875	868	940		
45	916	918	887	848	890		
60	839	913	925	848	846		
	c. C = 90 sec. (g = 70, r = 20)						
0	871	864	902	925	910	870	
15	869	869	864	902	925	910	
30	860	864	902	902	932	925	
45	952	932	910	932	926	967	
60	902	960	941	860	860	851	
75	855	902	918	907	864	864	
	d. C = 105 sec. (g = 85, r = 20)						
0	880	869	903	920	737	909	879
15	880	879	870	903	916	916	909
30	878	880	890	900	923	909	909
45	932	909	916	921	907	957	947
60	932	947	909	914	920	922	937
75	890	959	907	947	869	868	863
90	861	894	943	907	905	870	870

5.5.2 Offset and Link Length (Two-Phase Operation)

As in the one-way operation experimentation, to examine the effect of offset on network crossing time more closely, a smaller offset 2 increment, or two seconds, and zero values of offsets 1 and 3, were used. A two second offset 3 increment and zero values of offsets 1 and 2 were also used and a 400 feet link case was added.

As shown in Figures 5.36 - 5.39, with the same link length, the overall shape of each graph is very similar regardless of the cycle length. With the 200 feet link, network crossing times reach the minimum value, as absolute values of offsets 2 and 3 approach zero simultaneously and it increases as absolute values of offsets 2 and 3 increase simultaneously.

When links are 600 feet long, the overall shapes of the graphs are different from those of the 200 foot link cases. The maximum network crossing time for each cycle length is shorter and the offset range producing minimum network crossing time for each cycle is much wider.

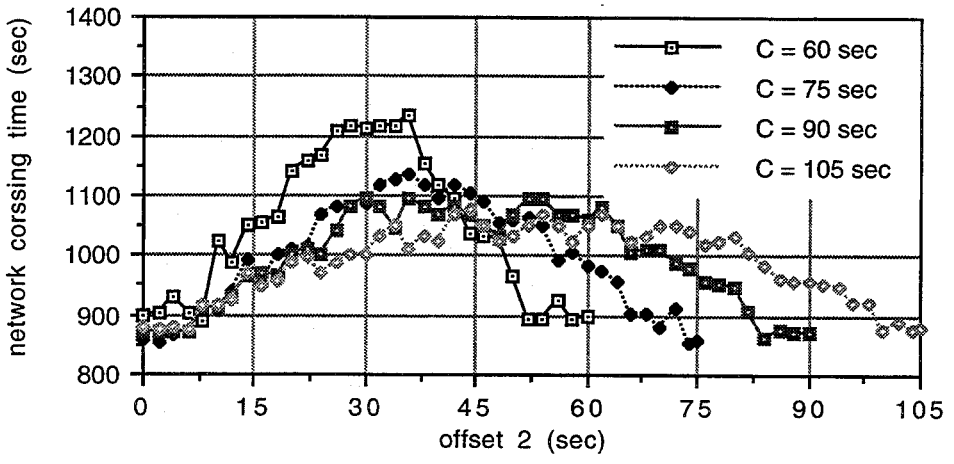


Figure 5.36: Offset 2 vs. network crossing time (two-way, two-phase, L = 200 ft, offset 3 = 0)

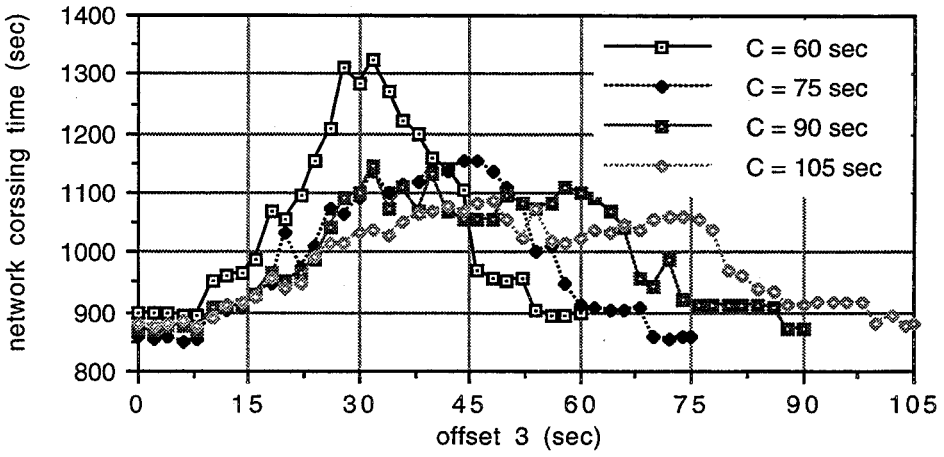


Figure 5.37: Offset 3 vs. network crossing time (two-way, two-phase, L = 200 ft, offset 2 = 0)

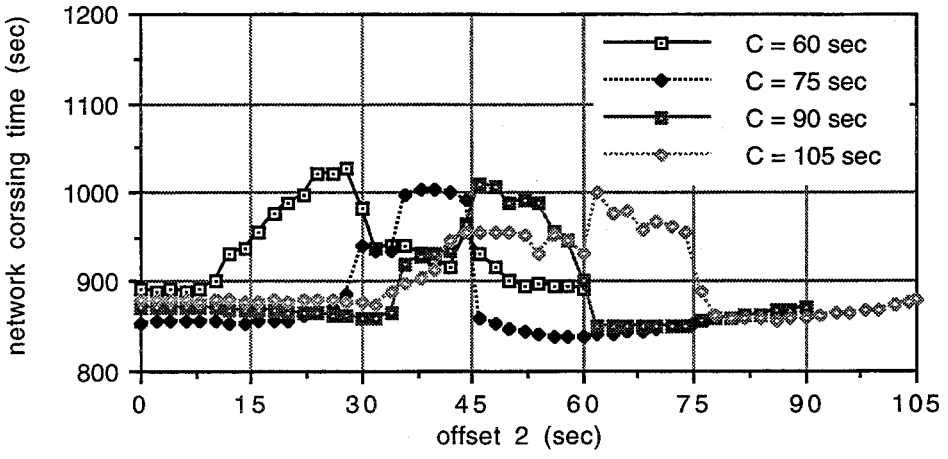


Figure 5.38: Offset 2 vs. network crossing time (two-way, two-phase, L = 600 ft, offset 3 = 0)

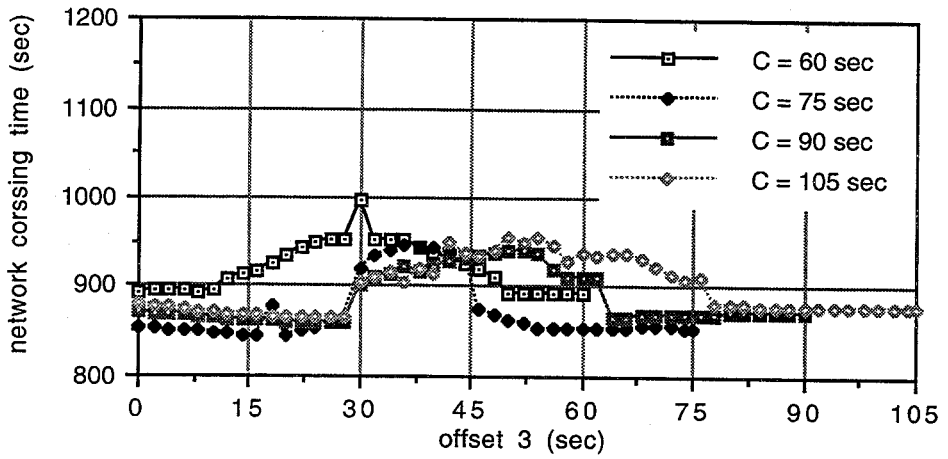


Figure 5.39: Offset 3 vs. network crossing time (two-way, two-phase, L = 600 ft, offset 2 = 0)

For the two-way and two-phase operation, Tables 5.25 and 5.26 summarize desirable and undesirable offset ranges respectively. The last row of Table 5.25 indicates the median desirable offset range value. The desirable offset range for each link length includes zero regardless of the cycle length. This result is quite different from one-way operation except when links are 200 feet long. Desirable offset ranges change with link length but not cycle length and undesirable offset ranges change with cycle length but not link length.

TABLE 5.25: DESIRABLE OFFSET RANGES (TWO-WAY AND TWO-PHASE OPERATION)

L (ft)	200		400		600	
	off 3 = 0	off 2 = 0	off 3 = 0	off 2 = 0	off 3 = 0	off 2 = 0
C (sec)	off 2	off 3	off 2	off 3	off 2	off 3
60	(-8) thru 8	(-6) thru 8	(-18) thru 18	(-12) thru 18	(-10) thru 10	(-10) thru 10
75	(-5) thru 6	(-5) thru 8	(-12) thru 18	(-12) thru 18	(-29) thru 28	(-29) thru 28
90	(-6) thru 6	(-2) thru 8	(-14) thru 18	(-16) thru 18	(-28) thru 34	(-26) thru 28
105	(-5) thru 6	(-5) thru 8	(-16) thru 18	(-15) thru 20	(-29) thru 36	(-27) thru 28
Med	0	2	2	2	2	0

TABLE 5.26: UNDESIRABLE OFFSET RANGES (TWO-WAY AND TWO-PHASE OPERATION)

L (ft)	200		400		600	
	off 3 = 0	off 2 = 0	off 3 = 0	off 2 = 0	off 3 = 0	off 2 = 0
C (sec)	off 2	off 3	off 2	off 3	off 2	off 3
60	20 thru 38	22 thru 44	20 thru 40	22 thru 36	18 thru 30	24 thru 36
75	24 thru 54	26 thru 52	32 thru 52	30 thru 48	30 thru 44	30 thru 44
90	26 thru 64	26 thru 66	34 thru 62	34 thru 56	36 thru 60	30 thru 62
105	32 thru 80	26 thru 78	34 thru 74	36 thru 68	38 thru 74	30 thru 76

5.5.3 Experimentation with Three-Phase Operation

Tables 5.27 and 5.28 summarize results of two-way and three-phase operation when links are 200 feet and 600 feet long respectively. For the same reason as in the one-way operation experimentation, 20 percent turning movements into cross streets and 10 percent turning movements into the arterial were used. Each table shows results of 126 simulation runs and the total design points are 252.

As shown in Tables 5.27 and 5.28, the result is quite different from the two-way and two-phase operation. Even though minimum network crossing times in both three-phase and two-phase operation are similar, maximum network crossing times for three-phase cycle lengths are much longer than two-phase. This is due primarily to the added arterial left-turn phase leading to a considerable increase in queue spillback possibility in each cross street left lane where turning movements from the arterial arrive. Therefore, arterial lanes 1 and 4 are more likely to be blocked by queue spillbacks from cross street lanes 5, 7, and 9 especially when poor offset combinations are applied.

As in the case of one-way operation, for a given link length, the overall pattern of results is very similar regardless of the cycle length. In most cases, particular patterns of maximum or minimum network crossing times, are formed along diagonals [see the shaded cells or adjacent cells in Tables 5.27 and 5.28]. Only the magnitude of the network crossing times, not the pattern, changes with cycle length.

For a 200 foot link, simultaneous greens produce minimum network crossing times regardless of the cycle length. This is identical to the two-way two-phase operation result. When offset 2 is 15 seconds greater than offset 3, network crossing times are maximum (see the shaded cells in the Table 5.27) except when the cycle length is 60 seconds. With few exceptions

when offset 2 equals offset 3, network crossing time is minimum. Figures 5.40 - 5.43 present results of simulation runs when links are 200 feet long (same as Table 5.27)

When links are 600 feet long, results are different than one-way operation. First, offset ranges producing minimum network crossing times are wider, and, second, compared with Table 5.8, the shaded cells of Table 5.28, indicating maximum network crossing times, are shifted downward 15 seconds.

When offset 2 is 45 seconds greater than offset 3, network crossing times are maximum indicating lowest efficiency and when offset 2 is 15 to 30 seconds greater than offset 3, network crossing times are minimum. Figures 5.44 - 5.47 illustrate results of simulation runs when links are 600 feet long (same as Table 5.28).

TABLE 5.27: NETWORK CROSSING TIME (SEC) [TWO-WAY, THREE-PHASE, L = 200 FT.]

off3	0	15	30	45	60	75	90
off2	a. C = 60 sec. (g = 35, l = 5, r = 20)						
0	868	910	1099	1321			
15	904	886	932	1002			
30	1033	1034	1028	1099			
45	993	1082	940	946			
	b. C = 75 sec. (g = 50, l = 5, r = 20)						
0	852	947	1215	1401	1510		
15	1502	881	965	1115	1316		
30	1411	2132	1133	1163	1212		
45	1269	1484	2144	1048	1109		
60	1032	1214	1361	1782	924		
	c. C = 90 sec. (g = 65, l = 5, r = 20)						
0	870	909	1188	1332	1486	1364	
15	1425	908	1024	1137	1231	1453	
30	1524	2068	1090	1134	1216	1279	
45	1373	1541	2222	1084	1186	1211	
60	1225	1340	1587	2184	1011	1090	
75	959	1197	1319	1538	1801	928	
	d. C = 105 sec. (g = 80, l = 5, r = 20)						
0	866	927	1090	1268	1347	1487	1738
15	1774	904	996	1230	1194	1337	1433
30	1537	1999	1069	1107	1322	1288	1347
45	1512	1608	2138	1066	1165	1197	1291
60	1302	1506	1709	2260	1051	1129	1165
75	1152	1289	1467	1745	2147	1007	1062
90	989	1137	1276	1412	1538	1794	913

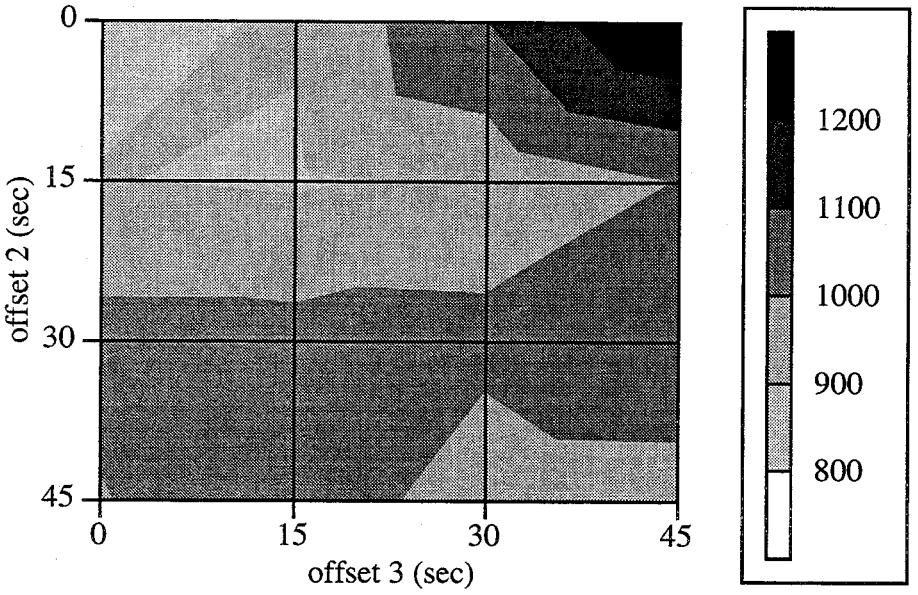


Figure 5.40: Network crossing time (sec) [two-way, three-phase, L = 200 ft, C = 60 sec]

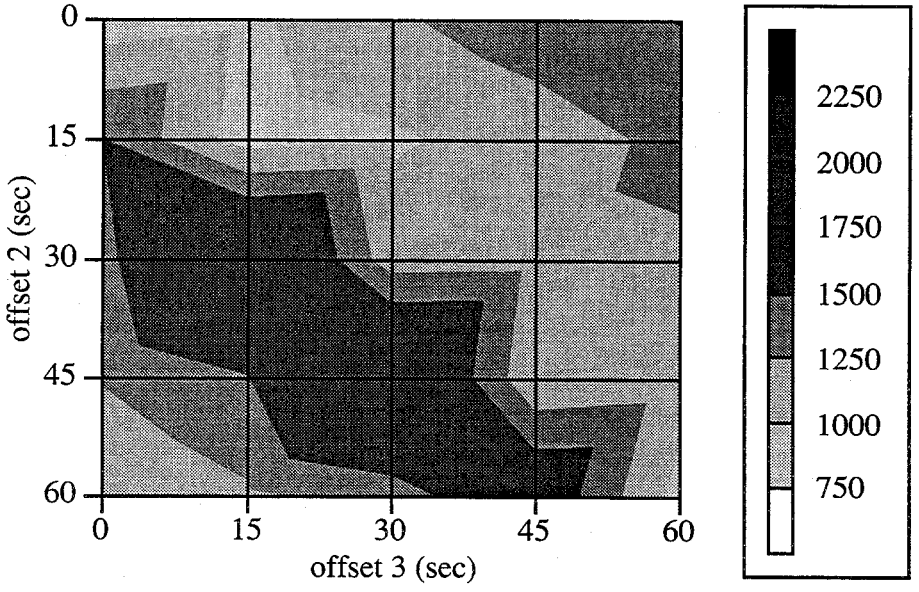


Figure 5.41: Network crossing time (sec) [two-way, three-phase, L = 200 ft, C = 75 sec]

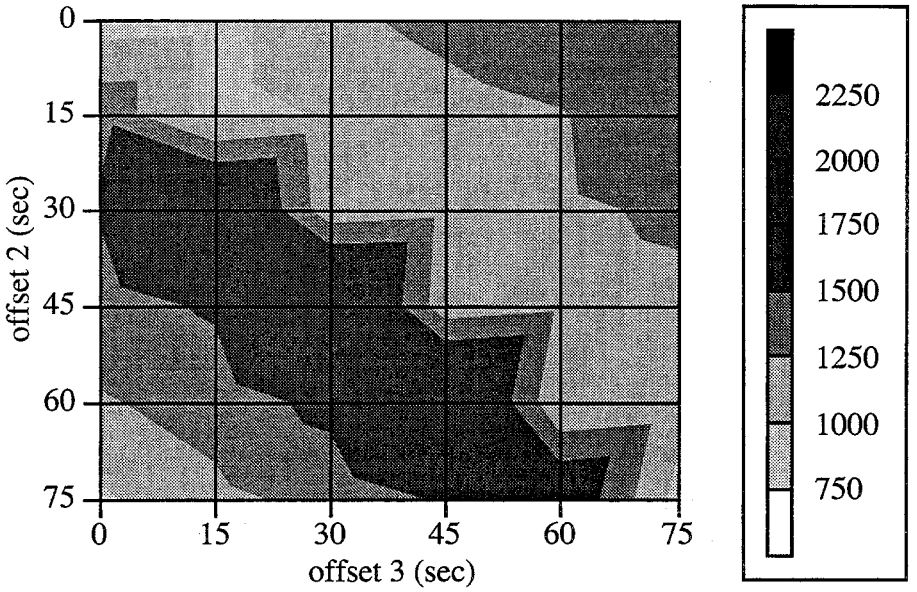


Figure 5.42: Network crossing time (sec) [two-way, three-phase, L = 200 ft, C = 90 sec]

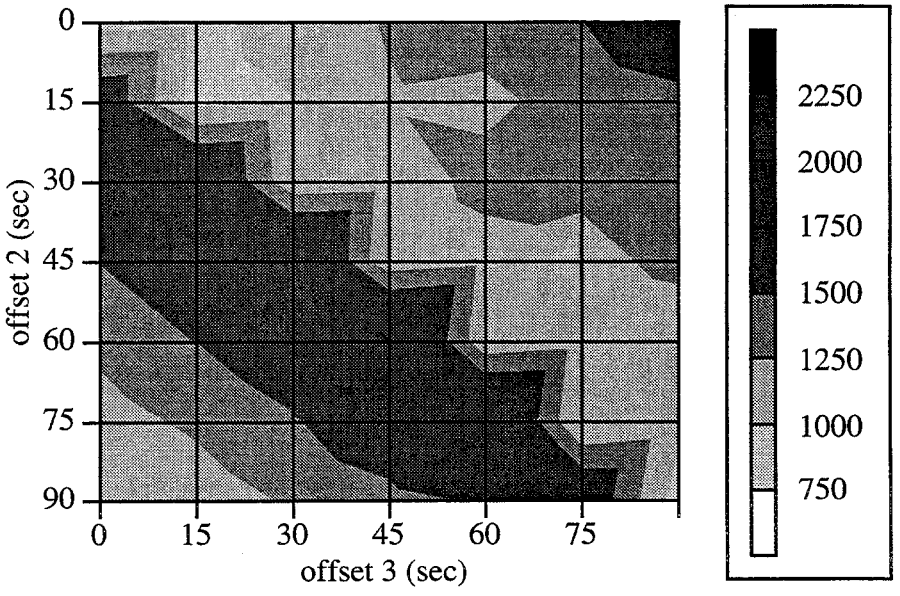


Figure 5.43: Network crossing time (sec) [two-way, three-phase, L = 200 ft, C = 105 sec]

TABLE 5.28: NETWORK CROSSING TIME (SEC) [TWO-WAY, THREE-PHASE, L = 600 FT]

off3	0	15	30	45	60	75	90
off2	a. C = 60 sec. (g = 27, l = 13, r = 20)						
0	834	1242	791	806			
15	813	873	1302	800			
30	853	863	908	1332			
45	1201	791	803	821			
	b. C = 75 sec. (g = 42, l = 13, r = 20)						
0	908	1005	1676	844	829		
15	823	884	1059	1666	848		
30	827	833	979	1039	1662		
45	1672	912	882	991	987		
60	986	1673	834	829	923		
	c. C = 90 sec. (g = 57, l = 13, r = 20)						
0	915	971	1083	1692	881	843	
15	840	892	991	1139	1689	929	
30	860	838	904	1000	1094	1689	
45	1714	850	845	963	1023	1117	
60	1135	1717	886	901	971	1045	
75	991	1110	1712	862	841	936	
	d. C = 105 sec. (g = 72, l = 13, r = 20)						
0	916	985	1061	1139	1810	897	854
15	855	901	986	1062	1248	1769	878
30	881	850	899	955	1036	1216	1769
45	1829	927	881	933	1058	1157	1253
60	1188	1869	966	881	988	1075	1152
75	1159	1157	1814	933	907	975	1056
90	982	1049	1192	1782	955	851	909

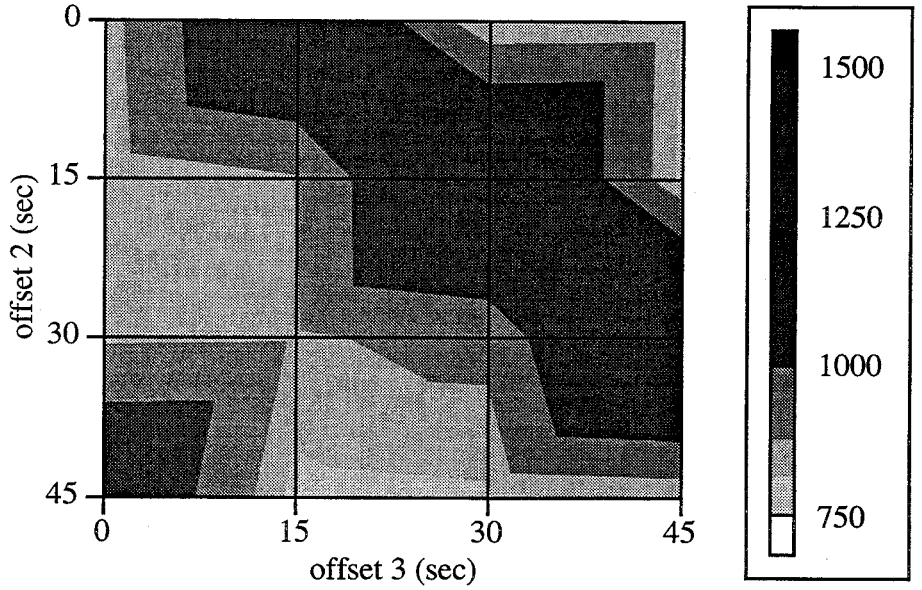


Figure 5.44: Network crossing time (sec) [two-way, three-phase, L = 600 ft, C = 60 sec]

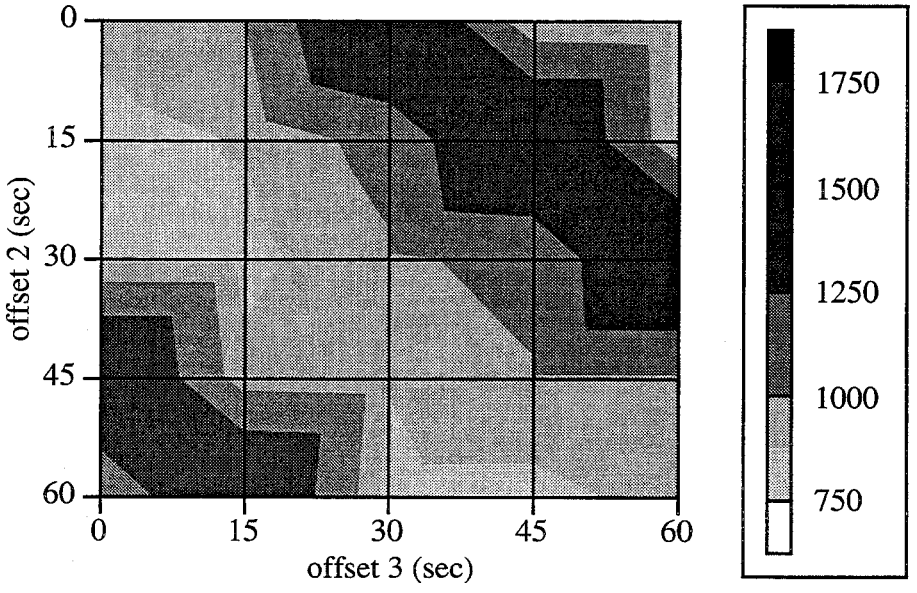


Figure 5.45: Network crossing time (sec) [two-way, three-phase, L = 600 ft, C = 75 sec]

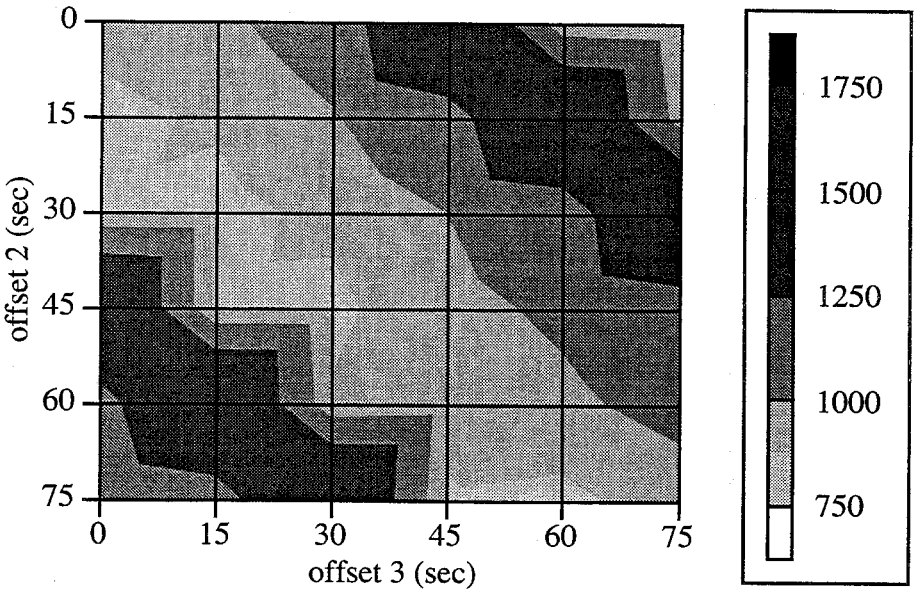


Figure 5.46: Network crossing time (sec) [two-way, three-phase, L = 600 ft, C = 90 sec]

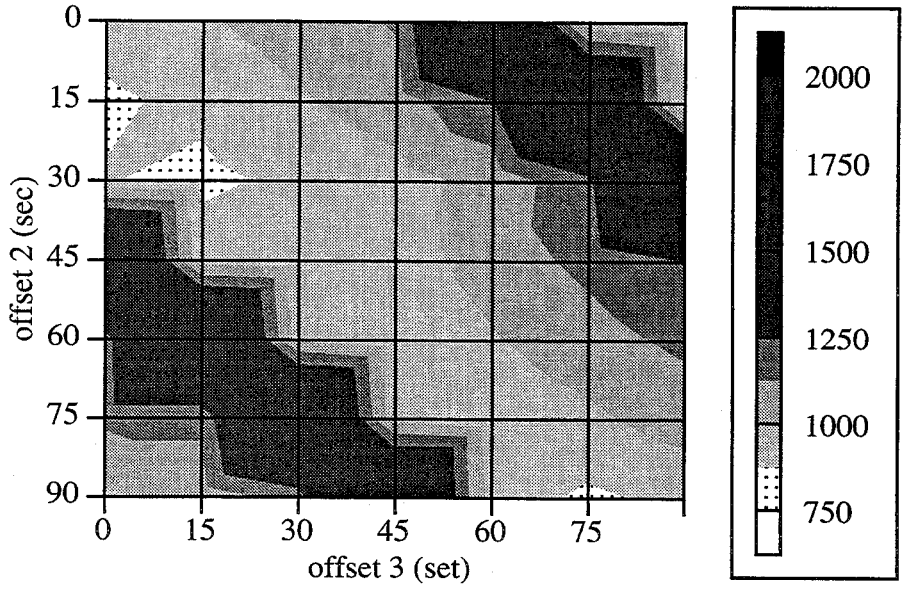


Figure 5.47: Network crossing time (sec) [two-way, three-phase, L = 600 ft, C = 105 sec]

5.5.4 Offset and Link Length (Three-Phase Operation)

The same procedure used in the two-way and two-phase operation, was applied to examine the offset effect on network crossing time more closely. As shown in the Figures 5.48 - 5.51, given any link length, overall graph shapes are similar. As discussed before, maximum network crossing times are much longer than those of two-way and two-phase operation.

As presented in Figures 5.48 and 5.49, when links are 200 feet long and offset 2 is about 10 seconds greater than offset 3, network crossing times are maximum. As in the two-way and two-phase case, with 200 foot links, network crossing times reach minimums, as absolute values of offsets 2 and 3 approach zero simultaneously.

When links are 600 feet long and offset 2 is about 24 seconds greater than offset 3, network crossing times are minimum. When offset 2 is about 41 seconds greater than offset 3, network crossing times are maximum. The desirable offset range producing minimum network crossing time, for each cycle length, is wider than that of the 200 foot link case.

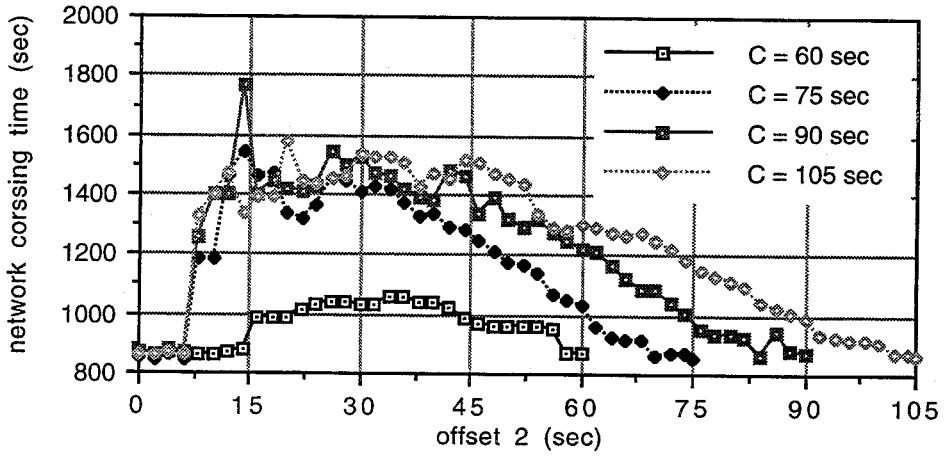


Figure 5.48: Offset 2 vs. network crossing time (two-way, three-phase, L = 200 ft, offset 3 = 0)

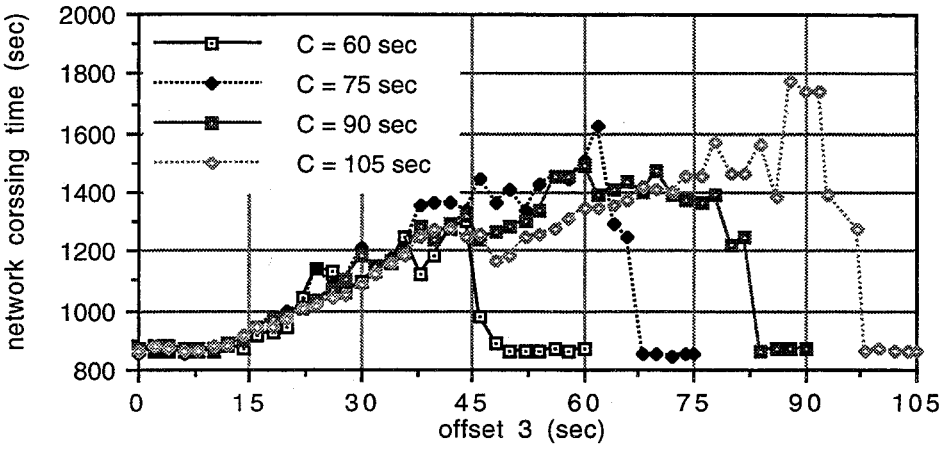


Figure 5.49: Offset 3 vs. network crossing time (two-way, three-phase, L = 200 ft, offset 2 = 0)

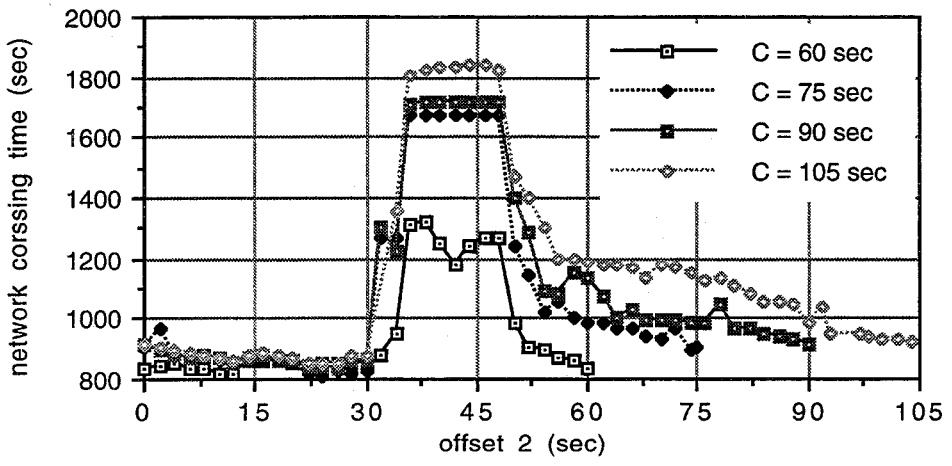


Figure 5.50: Offset 2 vs. network crossing time (two-way, three-phase, L = 600 ft, offset 3 = 0)

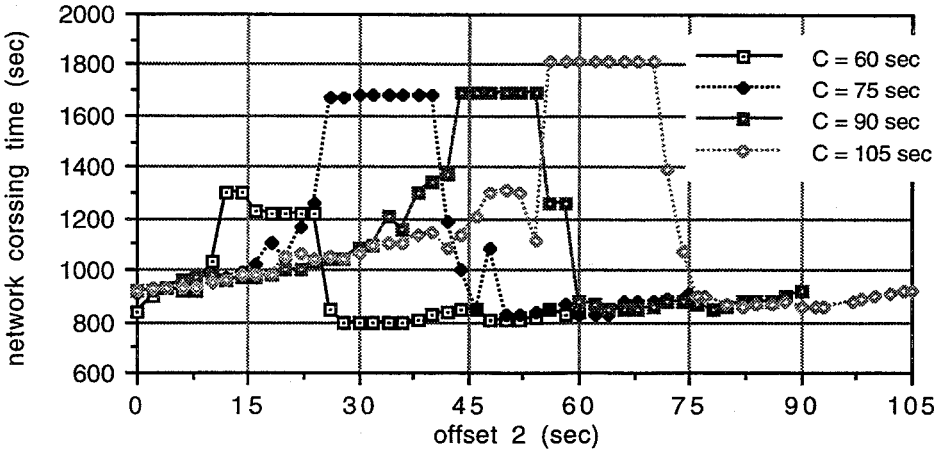


Figure 5.51: Offset 3 vs. network crossing time (two-way, three-phase, L = 600 ft, offset 2 = 0)

For the two-way and three-phase operation, Tables 29 and 30 summarize desirable and undesirable offset ranges and the last row of each table indicates median values. As in the previous cases, when links are 200 feet long, all desirable offset ranges include zero. The magnitudes of desirable offset ranges increase with link length but seem unrelated to cycle length. For the 400 feet link case, network crossing time is minimum when offset 2 is 9 seconds greater than offset 3, however, when links are 600 feet long, network crossing time is minimum when offset 2 is about 15 seconds greater than offset 3.

TABLE 5.29: DESIRABLE OFFSET RANGES (TWO-WAY ARTERIAL AND THREE-PHASE SIGNAL OPERATION)

L (ft)	200		400		600	
	off 3 = 0	off 2 = 0	off 3 = 0	off 2 = 0	off 3 = 0	off 2 = 0
C (sec)	off 2	off 3	off 2	off 3	off 2	off 3
60	(-2) thru 12	(-10) thru 10	(-2) thru 18	(-18) thru 0	(-4) thru 30	(-34) thru 0
75	(-5) thru 6	(-7) thru 10	(-1) thru 18	(-19) thru 0	4 thru 30	(-25) thru (-1)
90	(-2) thru 6	(-6) thru 10	0 thru 18	(-18) thru 0	4 thru 26	(-30) thru (-4)
105	(-3) thru 6	(-7) thru 12	(-1) thru 18	(-19) thru 0	4 thru 26	(-29) thru (-5)
Med	2	2	9	-9	15	-16

TABLE 5.30: UNDESIRABLE OFFSET RANGES (TWO-WAY ARTERIAL AND THREE-PHASE SIGNAL OPERATION)

L (ft)	200		400		600	
	off 3 = 0	off 2 = 0	off 3 = 0	off 2 = 0	off 3 = 0	off 2 = 0
C (sec)	off 2	off 3	off 2	off 3	off 2	off 3
60	NA	NA	24 thru 34	(-34) thru (-24)	36 thru 48	(-48) thru (-36)
75	14	-13	24 thru 32	(-31) thru (-25)	36 thru 48	(-49) thru (-35)
90	14	(-34) thru (-20)	24 thru 34	(-32) thru (-24)	36 thru 48	(-46) thru (-36)
105	20 thru 36	(-17) thru (-13)	24 thru 34	(-29) thru (-25)	36 thru 48	(-49) thru (-35)
Med	NA	NA	29	-28	42	-42

5.6 SUMMARY

This chapter discussed the experimental design, simulation experiments, and analysis of experimentation results. Introducing variability to parameters (departure headway, vehicle space in the queue, and vehicle speed) was described. Departure headways were based on the results of Efstathiadis' study (1992). Vehicle space uniformity was assumed throughout the simulation without loss of generality. Field data were collected to provide the simulation model with appropriate distributions of average overall speeds.

Oversaturated one-way street operations were examined first. To simplify the analysis, emphasis was placed on one-way arterial street operation. Offset was the dominant factor affecting system performance; although, link length was also important. When the link length was short, the optimum offset was approximately zero, irrespective of the cycle length. As the link length increases, downstream intersection green signal indications should begin before upstream greens creating downstream storage space for coming platoons. This is due to the longer queues possible with longer links which require more time to move the last queued vehicle before oncoming platoon arrival.

With the best offsets, the arterial green duration has little effect on the results, however, an inadequate cross street green interval is potentially damaging. System efficiency rapidly deteriorates when a cross street green is too short for a link length causing cross street queue spillback across the arterial. Therefore, a "practical" minimum cross street green interval is necessary.

When the best arterial offsets are not used, delay caused by arterial queue spillback increases as the arterial green interval increases. This is because "lost green" per cycle due to queue spillback increases as the arterial green interval increases.

When the cross street traffic demand is moderate to light (500 vehicles per hour per lane), an optimum g_C/g ratio exists. Therefore, the system efficiency decreases as the g_C/g ratio increases or decreases past the optimum. However, when the cross street traffic demand is heavy (1800 vehicles per hour per lane), no optimum g_C/g ratio exists because cross streets are oversaturated regardless of the g_C/g ratio. System efficiency increases as the g_C/g ratio increases regardless of offset combination.

Multiple regression analysis was used to derive a relationship between traffic control parameters and the response variable. For all cases, the coefficients of C (cycle length) and GCR (green ratio) did not achieve significance at the 0.05 level. Only the estimated regression coefficients of OFFD (offset difference) were statistically significant. This result is identical to the previous result in that offset is the dominant factor affecting system performance.

The two-way, three-phase operation, with short 200 feet links, produced almost the same results as one-way operation in which simultaneous greens produced minimum network crossing time regardless of cycle length. However, with long 600 foot links, results differ from one-way operation in that the offset range producing minimum network crossing times was much wider. Compared to two-way, two-phase operation with similar cycle length, maximum network crossing times for three-phase cases are much longer . This is due to the increased number of phases, resulting lost time, and the possibility that vehicles using the protected arterial left turn, from the arterial to the cross street, will spillback blocking the arterial.

CHAPTER 6 CONCLUSIONS

6.1 SUMMARY AND CONCLUSIONS

Many studies have been performed and applied successfully for the control of undersaturated traffic, but most of them have been ineffective or invalid in oversaturated conditions. There has been relatively limited research in the area of traffic control for oversaturated environments, and most of the research has been too theoretical to be applied in a real system. A number of traffic optimization models such as TRANSYT-7F and PASSER-II can develop optimal signal timing plans for undersaturated arterial networks, but none of these are applicable for oversaturated conditions.

In this study, a traffic simulation model was developed to provide a methodology for traffic signal timing in oversaturated urban arterial networks. Two control objectives of traffic signal timing in oversaturated conditions were taken into consideration. One was to maximize the throughput, or the number of vehicles processed during a given time period. The other was to prevent queue spillback or to minimize the occurrence of queue spillback if inevitable.

Critical simulation model traffic operational characterizations were field observation based. Departure headways were based on results of Efstathiadis' study (1992), which suggest that overall average queue start-up lost time of 1.34 seconds can be attributed to the first four vehicles and that average headways after the fourth vehicle were 1.82 seconds. Field data were collected to provide the simulation model with appropriate average overall speed distributions. The following relationship including all data points (1061) was selected for predicting Type 1 speed as a function of downstream clear space:

$$y = 13.033 + 0.026584\,x \quad (x \leq 730 \text{ ft}, \quad R = 0.894)$$

where

y = average overall speed (ft/sec)

x = downstream clear space (ft)

R = correlation coefficient

Signal timing offset was the dominant factor affecting system performance; although, link length was also important. When link length is short, the optimum offset is approximately zero, regardless of the cycle length, for both one-way and two-way arterial street operations. This result is identical to that found by Lieberman et al. (1986). Their study indicates that in closely spaced high traffic density networks, optimal relative arterial offsets are approximately zero (simultaneous green). As link length increases beyond the minimum 200 feet tested, for highest efficiency, downstream intersection greens should begin before upstream intersection greens. This relationship, opposite to conventional progression greens, moves downstream queues before incoming platoon arrival. This is due to the longer queues possible with longer links which require more time to move the last queued vehicle in the downstream link before oncoming platoon arrival. This offset was defined as the *network throughput offset*. This finding is quite different from the traditional arterial progression approach in which upstream intersection greens begin before downstream intersection greens so that a platoon of vehicles can pass the downstream intersection without stopping. This was defined as the *travel time offset*.

Cross street traffic operations can have significant effects upon arterial performance and system efficiency. When the cross street traffic demand is moderate to light (500 vehicles per hour per lane), an optimum cross street to arterial street green ratio (g_C/g) exists . When the g_C/g ratio is larger than optimum, the cross street green is not fully used, however, when the g_C/g ratio is smaller than optimum, cross street traffic demand exceeds capacity. Therefore, system efficiency decreases as the g_C/g ratio varies from the optimum. However, when the cross street traffic demand is heavy (1800 vehicles per hour per lane), no optimum g_C/g ratio exists because cross streets are oversaturated regardless of the g_C/g ratio. Furthermore, system efficiency increases as the g_C/g ratio increases regardless of offset combination. Therefore, in oversaturated conditions, a sufficiently long green interval should be assigned to cross streets to increase system efficiency and to prevent avoidable queue spillbacks. With fixed but adequate

cross street green and best offsets, the arterial green interval duration has little effect on system performance. However, system efficiency rapidly deteriorates when any cross street green becomes too short for the link length. Therefore, a "practical" minimum green interval for cross streets is necessary to accommodate the upstream cross street through traffic and turning vehicles from the arterial. When the cross street green is shorter than the minimum, even with the best offset combination, queue spillbacks occur in the cross streets and the system efficiency deteriorates.

Multiple regression analysis was used to derive a relationship between traffic control parameters and system efficiency as measured by network crossing time. For all cases, the coefficients of C (cycle length) and GCR (green ratio; cross street green / arterial green) were not significant at the 0.05 level. Only the estimated regression coefficients of OFFD (offset difference: = offset 3 - offset 2, for a 200 foot link; or offset 3 - offset 2 + 15, for a 600 foot link) were statistically significant. This result supports the previous conclusion that offset is the dominant factor affecting system performance.

A curve fitting method was used to derive a relationship between offset difference (x) and network crossing time (y). The equations obtained by this method are as follows:

$$y = 1478.2 - 3.752x_1 + 0.442x_1^2, \quad \text{for a 200 foot link,} \quad R^2 = 0.902$$

$$y = 1322.1 + 8.790x_2 + 0.332x_2^2, \quad \text{for a 600 foot link,} \quad R^2 = 0.968$$

where

y = network crossing time

x_1 = offset 3 - offset 2, for a 200 foot link

(if $x_1 < 0$, $x_1 = x_1 + C$)

x_2 = offset 3 - offset 2 + 15, for a 600 foot link

(if $x_2 < 0$, $x_2 = x_2 + C$, if $x_2 \geq C$, $x_2 = x_2 - C$)

For two-way arterial streets and two-phase signal operation, simultaneous green signal indications produced minimum network crossing time regardless of link length and cycle length.

This result is quite different from the one-way operation and the two-way and three-phase operation result except when links are 200 feet long. This is due to reduced queue spillback possibility in the two-way and two-phase operation when links are 400 or 600 feet long.

Two-way arterial, three-phase signal operation, with short 200 foot links, produced almost the same results as one-way operation in which simultaneous greens produced minimum network crossing time regardless of cycle length. However, with long 600 foot links, results differ from one-way operation in that the offset range producing minimum network crossing times was much wider. Compared to two-way, two-phase operation with similar cycle length, maximum network crossing times for three-phase cases are much longer. This is due to the increased number of phases and lost time, and the possibility that vehicles using the protected arterial left turn, from the arterial to the cross street, will spillback blocking the arterial. When this spillback situation occurs, consequences are grave because the arterial flow is partially stopped.

6.2 RESEARCH CONTRIBUTIONS

The most significant contribution of this research is the development of a traffic simulation model that deals with traffic signal timing in oversaturated conditions where existing models were not suitable for application. The model is capable of not only simulating a single case, evaluating existing traffic conditions, but also finding optimal solutions with respect to input parameters such as offsets, cycle length, and green split.

The new model utilizes departure headway and average overall speed to simulate vehicle movements, as explained in Chapters 4 and 5. This model is among the first in the area of network traffic simulation models to do so and has some advantages over the conventional way of simulating vehicle movements. With a large number of simulated vehicles in arterial networks, this approach is more efficiently utilized in the model so that simulations can be done with less computational effort. With appropriate field data, this approach includes vehicle interactions while moving, acceleration and deceleration rates, and cruising speed.

The development of the methodology for traffic signal timing in oversaturated traffic conditions is another contribution of this research. Offset was the dominant factor affecting system performance; although, link length was also important. With short link lengths, the optimum offset is approximately zero, regardless of the cycle length. With longer link lengths beyond the minimum 200 feet tested, downstream intersection greens should begin before upstream intersection greens for highest efficiency. This relationship, opposite to the conventional arterial progression approach in which upstream intersection greens begin before downstream intersection greens, is defined as network throughput offset. On the other hand, cycle length was not an important factor affecting system performance unless any green duration is too short for link length.

6.3 RECOMMENDATIONS

Application of Results

The traffic simulation model can be used to design a signal timing plan for an arterial street experiencing oversaturated traffic demand conditions. The model is recommended to be used in simulating not only a single case but also multiple cases (to find optimal timing). With given data such as arterial traffic signal timing and geometric configurations, the model may be used to simulate a single case evaluating existing conditions. Procedures for finding optimal input parameters (cycle length, green split, and offsets) are described below.

While fixing all other input parameters, an optimal cycle length is obtained by simulating cases with various cycle lengths and comparing the cases in terms of the network crossing time, which is the most comprehensive system performance measure. Minimum and maximum cycle lengths and a cycle length increment are required. An excessively long cycle length increment will result in less accurate solutions; on the other hand, an excessively short increment will greatly increase computer simulation time.

The same procedure can be applied to find an optimal green split. Minimum and maximum arterial/cross street greens and a green increment are also required. While fixing all other input parameters, an optimal green split is obtained by simulating cases with various combination of arterial and cross street green intervals (but fixed cycle length) and comparing the cases in terms of network crossing time.

To find optimal offsets, the model simulates cases with various combinations of offset 2 and offset 3 and while fixing all other input parameters. The best result, in terms of network crossing time, provides the best combination of offset 2 and offset 3. Minimum offset is set to zero while maximum offset is a cycle length minus an offset increment.

Further Study

This study was an initial attempt to develop a methodology for traffic signal timing in oversaturated arterial networks using the newly developed traffic simulation model. Thus, this study has some limitations in assumptions and arterial geometric configurations.

Since only one-way cross street operation was considered, the model should be extended to handle two-way cross streets. However, two-way cross street operation will add complexity in terms of the number of signal phases and turning movements. The type of intersection control is limited to pretimed signals. With oversaturated conditions, however, actuated signal control essential becomes virtual pretimed control. Simplifying assumptions regarding the number of vehicle classes (currently one) might be released. This would better enable analyses of those special arterial street cases in which trucks could be very important. The model, which can accommodate nine intersections that include three arterial intersections, should be extended to deal with more arterial and cross street intersections.

Appendix A

Source Code (One-way Operation Version):

Traffic Simulation Model For Oversaturated Arterial Networks

```
program

c traffic simulation model for oversaturated
  arterial networks
c this model is designed specifically for
  oversaturated arterial traffic conditions.
c traffic demand at the entry of the arterial
  street is greater than the capacity.

c definitions of input parameters

c mult: number of cases (0; single case,1;
  multiple cases for optimization)
c jcyc: cycle length optimization (0; no,1;
  yes)
c jvar: variability of parameters (0; no,1;
  yes)
c nvol: number of vehicles to be simulated
c ncwu: number of signal cycles for warm-up
  time
c tdc: cross street traffic demand (arriving
  headway)
c nac: number of turning vehicles per cycle
  from arterial to cross street(CS)
c nca: number of turning vehicles per cycle
  from CS to arterial
c dist(1): link length between the 1st and 2nd
  intersections [ft]
c dist(2): link length between the 2nd and 3rd
  intersections [ft]
c wida: width of arterial [ft]
c widc: width of cross streets [ft]
c avsh: average vehicle space headway [ft]
c icl: cycle length [sec]
c lgi: arterial green interval [sec]
c idxn: difference between maximum and minimum
  arterial greens [sec]
c incg: increment of idxn (for optimization)
  [sec]
c ipt: protected left turn phase duration (only
  for two-way operation) [sec]
c jdq: minimum delay to determine a queue
  spillback [sec]
c noff(1): offset 1 [sec]
c ioff1: minimum offset 2 (0,for optimization)
c ioff2: minimum offset 2 (C,for optimization)
c joff1: minimum offset 3 (0,for optimization)
c joff2: minimum offset 3 (C,for optimization)
c incre: increment of offsets (for
  optimization)

dimension arr1(0:1500), dep1(0:1500),
  arr2(0:1500)
dimension dep2(0:1500), arr3(0:1500),
  dep3(0:1500)
dimension arr11(0:1500), dep11(0:1500),
  arr21(0:1500)
dimension dep21(0:1500), arr31(0:1500),
  dep31(0:1500)
dimension zz(0:100), zsut(0:100), igf(20)
dimension arrr2(0:500), arrl3(0:500),
  arrr4(0:500)
dimension carrl1(0:500), carrl2(0:500),
  carrl3(0:500)
dimension ki2(0:1500), ki3(0:1500),
  ki21(0:1500), ki31(0:1500)
dimension dist(20), iveh(20), ir(20), ig(20),
  ir1(20), ig1(20)
dimension pta1(0:15, 0:15), pta2(0:15, 0:15)
dimension ptc1(0:15, 0:15), ptc2(0:15, 0:15)
dimension ptc3(0:15, 0:15), ptat(0:15, 0:15),
  ptct(0:15, 0:15)
dimension ptst(0:15, 0:15), spst(0:15, 0:15)
dimension mar(0:15, 0:15), mal(0:15, 0:15),
  mwu(0:15, 0:15)
dimension qs2(100), qs20(100), qs21(100),
  qs31(100)
dimension qst(100), qst1(100), qsti(100),
  qsti1(100)
dimension noff(20), jgn1(100), jgn2(100),
  jgn3(100)
dimension nka(100), nka0(100), nka1(100),
  nka2(100)
dimension qsi(100), qsi0(100), qsi1(100),
  qsi2(100)
dimension suts(0:100, 0:60), suts0(0:100, 0:60)
dimension suts1(0:100, 0:60), suts2(0:100,
  0:60)
dimension rqst(100), rqst1(100), dep(10)
dimension rqs2(100), rqs20(100), rqs21(100),
  rqs31(100)
dimension tuz(15), dtuz(15)
dimension kfa(10), ma(15), st(15), st1(15)

dimension mcl1(0:15, 0:15), mcr1(0:15, 0:15)
dimension carr1(0:1500), cdep1(0:1500),
  carr2(0:1500)
dimension cdep2(0:1500), carr3(0:1500),
  cdep3(0:1500)
dimension darr1(0:1500), ddep1(0:1500),
  darr2(0:1500)
dimension ddep2(0:1500), darr3(0:1500),
  ddep3(0:1500)
dimension kir2(0:1500), kir3(0:1500),
  kir21(0:1500)
dimension irr(20), igr(20), irr1(20), igr1(20),
  kir31(0:1500)
dimension nkar(100), nkar1(100)
dimension sutsr(0:100, 0:60), sutsr1(0:100,
  0:60)
dimension qsr2(100), qsr21(100), qstr(100),
  qstr1(100)
dimension qsir(100), qsir1(100), qstir(100),
  qstir1(100)
dimension rqsr2(100), rqsr21(100), rqstr(100),
  rqstr1(100)

dimension mcl2(0:15, 0:15), mcr2(0:15, 0:15)
dimension earr1(0:1500), edep1(0:1500),
  earr2(0:1500)
dimension edep2(0:1500), earr3(0:1500),
  edep3(0:1500)
dimension farr1(0:1500), fdep1(0:1500),
  farr2(0:1500)
dimension fdep2(0:1500), farr3(0:1500),
  fdep3(0:1500)
dimension kis2(0:1500), kis3(0:1500),
  kis21(0:1500)
```

```
dimension irs(20), igs(20), irs1(20), igs1(20),
  kis31(0:1500)
dimension nkas(100), nkas1(100)
dimension sutss(0:100, 0:60), sutss1(0:100,
  0:60)
dimension qss2(100), qss21(100), qsts(100),
  qsts1(100)
dimension qsis(100), qsis1(100), qstis(100),
  qstis1(100)
dimension rqss2(100), rqss21(100), rqsts(100),
  rqsts1(100)

dimension mcl3(0:15, 0:15), mcr3(0:15, 0:15)
dimension garr1(0:1500), gdep1(0:1500),
  garr2(0:1500)
dimension gdep2(0:1500), garr3(0:1500),
  gdep3(0:1500)
dimension harr1(0:1500), hdep1(0:1500),
  harr2(0:1500)
dimension hdep2(0:1500), harr3(0:1500),
  hdep3(0:1500)
dimension kit2(0:1500), kit3(0:1500),
  kit21(0:1500)
dimension irt(20), igt(20), irt1(20), igt1(20),
  kit31(0:1500)
dimension nkat(100), nkat1(100)
dimension sutst(0:100, 0:60), sutst1(0:100,
  0:60)
dimension qst2(100), qst21(100), qstt(100),
  qstt1(100)
dimension qsit(100), qsit1(100), qstit(100),
  qstit1(100)
dimension rqst2(100), rqst21(100), rqstt(100),
  rqstt1(100)
dimension mj(100), mj1(100), mj2(100), bj(30),
  bj1(30), bj2(30)
dimension dy1(0:60), dy2(0:60), dy3(0:60)
dimension dy4(0:60), dy5(0:60), dy6(0:60)
dimension dy11(0:60), dy12(0:60), dy13(0:60)
dimension dy14(0:60), dy15(0:60), dy16(0:60)
dimension dy21(0:60), dy22(0:60), dy23(0:60)
dimension dy24(0:60), dy25(0:60), dy26(0:60)
dimension dy31(0:60), dy32(0:60), dy33(0:60)
dimension dy34(0:60), dy35(0:60), dy36(0:60)
C----------------------------------------
open(4, file='input', status='old',
  form='formatted')
read(4, *) mult, jcyc, jvar
read(4, *) nvol, ncwu, tdc, nac, nca
read(4, *) dist(1), dist(2), wida, widc, avsh
read(4, *) icl, lgi, idxn, incg, ipt, jdq
read(4, *) noff(1), ioff1, ioff2, joff1, joff2,
  incre

open(unit=10, file='y1d', status='unknown',
  form='formatted')
open(unit=11, file='y2i', status='unknown',
  form='formatted')
open(unit=20, file='y5p', status='unknown',
  form='formatted')
open(unit=21, file='y6j', status='unknown',
  form='formatted')
open(unit=30, file='y7', status='unknown',
  form='formatted')
open(unit=31, file='y8', status='unknown',
  form='formatted')
open(unit=40, file='y9', status='unknown',
  form='formatted')
open(unit=41, file='y10', status='unknown',
  form='formatted')
open(unit=50, file='sc', status='unknown',
  form='unformatted')

ku1 = 10
ku2 = 11
ku5 = 20
ku6 = 21
ku7 = 30
ku8 = 31
ku9 = 40
ku10 = 41
ku15 = 50
C------
lri = icl - lgi
c lri: arterial red interval [sec]

nac1 = nac
ncas = nca
if(jdq.eq.11) then
nac1 = 3*nac
ncas = 3*nca
endif
if(mult.eq.1) then
write(ku1, 120) icl, lgi, lri, dist(1)
write(ku2, 120) icl, lgi, lri, dist(1)
write(ku5, 120) icl, lgi, lri, dist(1)
write(ku6, 120) icl, lgi, lri, dist(1)
120 format(/2x, 'C=', i3, 'sec.', ' g=', i2, '
  r=', i2,',link length =', f5.0, 'ft.')
write(ku1, 125) nac1, ncas
write(ku2, 125) nac1, ncas
write(ku5, 125) nac1, ncas
write(ku6, 125) nac1, ncas
125 format(2x, 'turn-in: ', '(art to cro) =',
  i2, ' veh/cycle, ' , ' (cro to art) =', i2,
  ' veh/cycle')
endif

do 400 ip = ioff1,ioff2
do 410 jp = joff1,joff2
do 420 kp = lgi,lgi+idxn,incg

noff(2) = ip*incre
noff(3) = jp*incre
if(noff(2).ge.icl) noff(2) = noff(2) - icl
if(noff(3).ge.icl) noff(3) = noff(3) - icl
ipz = noff(2)
jpz = noff(3)

lgi = kp
if(jcyc.eq.1) then
lri = kp/incg
icl = lgi + lri
else
lri = icl - kp
endif

if(mult.eq.0) then
call HEAD(ku1, icl, lgi, lri, noff(2), noff(3))
```

```
call HEAD(ku2, icl, lgi, lri, noff(2), noff(3))
endif
do 130 ij = 1,100
nka(ij) = 1
nka0(ij) = 1
nka1(ij) = 1
nka2(ij) = 1
130 continue
c nka: serial number,starts form the beginning
  of each green

do 140 km = 0,300
carr11(km) = 0.
carr12(km) = 0.
carr13(km) = 0.
arrr2(km) = 0.
arr13(km) = 0.
arrr4(km) = 0.
140 continue


***** Arterial,A direction**********
***** Right lane

call INPUT(icl, dist, iveh, speed, ni)
ixt11 = 0
ixt12 = 0
ixt13 = 0
ixt21 = 0
ixt22 = 0
ixt23 = 0
c speed: average overall speed (fps)
c ixt: 1; if signal timing updates (green
  ends),to exit or enter LEFTL or RIGHTL. 0;
  otherwise,1st digit: 1-right lane,2-left
  lane; 2nd digit: intersection #
c iveh: maximum number of vehicles can be
  stored in a link
c ni: number of arterial intersections

ntg = 99
jvol = 0
c jvol: cumulative number of vehicles simulated

kskip = 0
c kskip: 1; simulation ends,0; otherwise

do 150 j = 1,10
kfa(j) = 0
150 continue
c kfa: 1; initialize,0; skip

do 160 j = 1,15
 ma(j) = 1
 tuz(j) = 0.0
 dtuz(j) = 0.0
160 continue
c ma: do loop index
c tuz: the time queue spillback is cleared
c dtuz: departure delay in the next green
   interval

nibi = 0
nibj = 0
imq1 = 0
imq2 = 0
imq3 = 0
mqc1 = 0
mqc2 = 0
mqc3 = 0
mqc11 = 0
mqc21 = 0
mqc31 = 0
jac = 1
jor = 0
jor1 = 0
jor2 = 0
jdum = -302357
c imq: determine whether to exit or enter main
  subroutines
c mqc: number of vehicle stored in the link
c jac: 1; arterial,0; CR
c jor: number of turning vehicles during each
  green interval

c------ Initialization ------------------

if(kfa(1).eq.0) then
call SIGNAL(1, speed, icl, ni, ir, ig, noff,
  lgi)
call INIT(arr1, dep1, arr2, dep2, arr3, dep3,
  nc, ne, ng, k1, k2, k3, i, n, m)
c arri: arrival time at intersection i
c depi: departure time at intersection i
c nc, ne, ng, k1, k2, k3, i, n, m: index

call INIT1(jo, nr, iup, iqi2, iqi3, ncy1, ncy2,
  ncy3, nq2, nq3, nq2i, iqt, iqe, iqea, iqti,
  nqst)
c jo: number of turning-out vehicles from
  arterial per green
c nr, nl, nnn: number of turning-in vehicles
  from CR per cycle
c iup: 1: next turning-in vehicle comes in next
  green,0: otherwise
c iqi: number of queue spillback indicators
  (QSi) caused by thru traffic (link is full)
c ncy: number of cycles
c nq: number of vehicle stored in the link
c iqt: number of QSi caused by turning vehicles
  (link is full)
c iqe: 1; QS caused by turning-in vehicle(s)
  occurs,0; otherwise
c iqea: 1; use arrr(n) instead of arr(m-1) for
  variable time in NOQUE,0; otherwise
c iqti: number of QSi caused by turning
  vehicles (link is not full)
c nqst: number of queue spillback (QS) caused
  by turning vehicles

call INIT2(nii, isp, iqs, nqs, niq, dtiq, nua,
  dtua, nux, dtux)
c nii: 1; QSi by turning vehicles occurs,0;
  otherwise
c isp: number to detect new queue
  spillback,similar to number of cycles but
  different
c nqs: number of QS caused by thru traffic
c niq: 1; QS by turning vehicles occurs
  (affects next green),0; otherwise
```

```
c dtiq: delay of departure when niq=1
c nua: 1; iqi>0 or QS by thru traffic occurs
   (link is full),0; otherwise
c dtua: delay of departure when nua=1
c nux: 1; QS by thru traffic occurs (link is
   full),0; otherwise
c dtux: delay of departure when nux=1

call INIT2(nii0, isp0, iqs0, nqs0, niq0, dtiq0,
   nua0, dtua0, nux0, dtux0)
call SIGADJ(ir, ig, lri, lgi, icl, dist, speed,
   jac, kc, arrr2, jdum,jvar)

call SIGNA1(1, ni, ir, ig, ir1, ig1)

ismt = ig(2) + ncwu * icl
c ismt: warm-up time (10 signal cycles)

igf(1) = ig(1)
igf(2) = ig(2)
igf(3) = ig(3)

jgn1(1) = ig(1)
jgn2(1) = ig(2)
jgn3(1) = ig(3)
c jgn: beginning of green interval

do 190 k = 2,100
jgn1(k) = jgn1(k-1) + icl
jgn2(k) = jgn2(k-1) + icl
jgn3(k) = jgn3(k-1) + icl
190 continue
call DPHDWY(zz, zsut)
c zsut: time required for the nth vehicle in
   the queue
c to start moving after signal turns green

do 200 im = 1,100
do 210 jm = 0,60,1
suts(im, jm) = jgn2(im) + zsut(jm)
suts0(im, jm) = jgn3(im) + zsut(jm)
suts1(im, jm) = jgn2(im) + zsut(jm)
suts2(im, jm) = jgn3(im) + zsut(jm)
210 continue
200 continue
c suts: time to start moving

kfa(1) = 1
endif

smtj = 0.
krx1 = 0
krx2 = 0
ncy21 = 0
igt(2) = 0
c smtj: the time simulation ends
c krx1: 1; skip RIGHTL of 1st CR (QS in lane 7
   or 9 blocks lane 6 duringr the whole next
   green),0; otherwise
c krx2: 1; skip RIGHTL of 2nd CR (QS in lane 9
   blocks lane 8 duringr the whole the next
   green),0; otherwise

1000 continue
levp = 0
jac = 1
klt1 = 0
krt1 = 0
klt2 = 0
krt2 = 0
klt3 = 0
krt3 = 0
ips2 = 0
ips3 = 0
c klt1: 1; execute LEFTL of 1st CR,0; skip it
c krt2: 1; execute RIGHTL of 2nd CR,0; skip it
c isp2: 1; skip CROSST,0; otherwise

if(ncy21.ge.4) then

if(ig(1).gt.igr1(2).or.ig1(1).gt.igr(2)) then
if(ig(1).gt.igr1(2)) krt1 = 1
if(ig1(1).gt.igr(2)) klt1 = 1
ips2 = 1
go to 1010
endif
if(ig(2).gt.igs(2)) then
if(ig(2).gt.igs(2)) klt2 = 1
ips3 = 1
go to 1020
endif
if(ig(3).gt.igt1(2).or.ig1(3).gt.igt(2)) then
if(ig(3).gt.igt1(2)) krt3 = 1
if(ig1(3).gt.igt(2)) klt3 = 1
go to 1030
endif
endif

731 call RIGHTL(1, nvol, dist, iveh, speed,
   jdq, mult, icl, lgi, kul, arrr2, carrl2, m,
   m1, ma, ntg, jvol, imq1, ki2, ki3, jo, nr,
   at2, arr1, dep1, arr2, dep2, arr3, dep3, i,
   k1, k2, k3, ig, ir, nc, ne, ng, iqi2, iqi3,
   qs2, qs20, nq2, nq3, ncy1, ncy2, ncy3, nka,
   nka0, isp, isp0, nua, nua0, dtua, dtua0,
   suts, suts0, iqs, iqs0, qsi, qsi0, nqs,
   nqs0, rqs2, rqs20, niq, niq0, dtiq, dtiq0,
   nux, nux0, dtux, dtux0, jac, tuz, dtuz, 10,
   11, 12, nq2i, nuzj1, dtuzj1, n, iup, iqt,
   qst, iqe, iqea, nii, nii0, iqti, qsti, nqst,
   rqst, jor, mqc2, sutss1, isps1, nkas1, ismt,
   nibi, nibj, ixt11, ixt12, ixt13, icy, st,
   st1, nac, noff, krt3, krx1, irs, edep3,
   jdum, dy1, dy2, dy3, jvar)

***** Arterial,A direction *********
***** Left lane

if(kfa(2).eq.0) then
jo2 = 0
call INIT(arr11, dep11, arr21, dep21, arr31,
   dep31, nc1, ne1, ng1, k11, k21, k31, ii, nn,
   m)
call INIT1(jo1, n1, iup1, iqi21, iqi31, ncy11,
   ncy21, ncy31, nql2, nql3, nql3i, iqt1, iqe1,
   iqea1, iqti1, nqst1)
call INIT2(nii1, isp1, iqs2, nqs2, niq1, dtiq1,
   nua1, dtua1, nux1, dtux1)
```

```
call INIT2(nii2, isp2, iqs3, nqs3, niq2, dtiq2,
  nua2, dtua2, nux2, dtux2)
nux3 = 0
dtux3 = 0.
kfa(2) = 1
endif

call LEFTL(2, nvol, dist, iveh, speed, jdq,
  mult, icl, lgi, ku2, arrl3, carrl1, m, ma,
  ntg, jvol, imq2, imq3, ki21, ki31, krt2,
  klt3, ixt21, ixt22, ixt23, jo1, nl, at3,
  arrl1, dep11, arr21, dep21, arr31, dep31,
  ii, kl1, k21, k31, ig1, ir1, nc1, ne1, ng1,
  iqi21, iqi31, qs21, qs31, nql2, nql3, ncy11,
  ncy21, ncy31, nka1, nka2, isp1, isp2, nua1,
  nua2, dtua1, dtua2, suts1, suts2, iqs2,
  iqs3, qsi1, qsi2, nqs2, nqs3, rqs21, rqs31,
  niq1, niq2, dtiq1, dtiq2, nux1, nux2, dtux1,
  dtux2, jac, tuz, dtuz, 20, 21, 22, nql3i,
  nuzj2, dtuzj2, nn, iup1, iqt1, qst1, iqe1,
  iqea1, nii2, iqti1, qsti1, nqst1, rqst1,
  jor1, mqc1, mqc3, sutsr, ispr, nkar, ismt,
  carrl3, jo2, nuzj3, dtuzj3, jor2, sutst,
  ispt, nkat, icz, st, st1, nac, noff, krx1,
  krx2, nux3, dtux3, irr, irt, cdep3, gdep3,
  jdum, dy4, dy5, dy6, jvar)
if(krx1.eq.1.and.krx2.eq.1) levp = 1
if(krx2.eq.1) krt2 = 0

***** 1st Cross Street ***************

if((ixt21.eq.1.or.imq2.eq.1.or.ixt11.eq.1.and.
  krx1.ne.1).or.levp.eq.1) then
 if(ixt21.eq.1.or.imq2.eq.1) klt1 = 1
 if(ixt11.eq.1.and.krx1.ne.1) krt1 = 1
1010 write(ku15)(mj(imj), imj=1, 45)
write(ku15)(mj(imj), imj=46, 56)
write(ku15)(bj(ibj), ibj=1, 18)
rewind(ku15)
if(krx1.eq.1) krt1 = 0

if(levp.eq.1) klt1 = 1

if(klt1.eq.0.and.krt1.eq.0.and.ips2.eq.1) go to
  731

call CROSST(1, ku5, ku6, tdc, carrl1, arrr2,
  nvol, lri, noff, icl, jdq, ipz, jpz, speed,
  ntg, mult, jvol, tuz, dtuz, mqc1, mqc11,
  klt1, krt1, 30, 32, ispr, nkar, sutsr,
  ispr1, nkar1, sutsr1, dist, iveh, ismt, irr,
  igr, irr1, igr1, igf, kfa, ma, st, st1, nca,
  5, iqir2, qsr2, iqsr, qsir, nqsr, rqsr2,
  iqtr, qstr, iqtir, qstir, nqstr, rqstr,
  kir2, kir3, 6, iqir21, qsr21, iqsr1, qsir1,
  nqsr1, rqsr21, iqtr1, qstr1, iqtir1, qstir1,
  nqstr1, rqstr1, kir21, kir31, carr1, cdep1,
  carr2, cdep2, carr3, cdep3, darr1, ddep1,
  darr2, ddep2, darr3, ddep3, isp, nka, suts,
  nq2i, dep2,jdum, dy11, dy12, dy13, dy14,
  dy15, dy16, jvar, ispec)
if(ispec.eq.99) then
ispec = 0
endif
read(ku15)(mj(imj), imj=1, 45)
read(ku15)(mj(imj), imj=46, 56)
read(ku15)(bj(ibj), ibj=1, 18)
rewind(ku15)
endif

if(ips2.eq.1) go to 1070

***** 2nd Cross Street ***************

if(ixt12.eq.1.or.imq1.eq.1.or.krt2.eq.1.or.
  ixt22.eq.1.and.krx2.ne.1) then
 if(ixt12.eq.1.or.imq1.eq.1) klt2 = 1
 if(ixt22.eq.1.and.krx2.ne.1) krt2 = 1
1020 write(ku15)(mj1(imj), imj=1, 45)
write(ku15)(mj1(imj), imj=46, 56)
write(ku15)(bj1(ibj), ibj=1, 18)
rewind(ku15)
if(krx2.eq.1) krt2 = 0

call CROSST(2, ku7, ku8, tdc, carrl2, arrl3,
  nvol, lri, noff, icl, jdq, ipz, jpz, speed,
  ntg, mult, jvol, tuz, dtuz, mqc2, mqc21,
  klt2, krt2, 40, 42, isps, nkas, sutss,
  isps1, nkas1, sutss1, dist, iveh, ismt, irs,
  igs, irs1, igs1, igf, kfa, ma, st, st1, nca,
  7, iqis2, qss2, iqss, qsis, nqss, rqss2,
  iqts, qsts, iqtis, qstis, nqsts, rqsts,
  kis2, kis3, 8, iqis21, qss21, iqss1, qsis1,
  nqss1, rqss21, iqts1, qsts1, iqtis1, qstis1,
  nqsts1, rqsts1, kis21, kis31, earr1, edep1,
  earr2, edep2, earr3, edep3, farr1, fdep1,
  farr2, fdep2, farr3, fdep3, isp2, nka2,
  suts2, nql3i, dep31,jdum, dy21, dy22, dy23,
  dy24, dy25, dy26, jvar, ispec)
read(ku15)(mj1(imj), imj=1, 45)
read(ku15)(mj1(imj), imj=46, 56)
read(ku15)(bj1(ibj), ibj=1, 18)
rewind(ku15)
endif

if(ips3.eq.1) go to 1070

***** 3rd Cross Street ***************

if(ixt23.eq.1.or.imq3.eq.1.or.ixt13.eq.1.or.
  krt3.eq.1) then
 if(ixt23.eq.1.or.imq3.eq.1) klt3 = 1
 if(ixt13.eq.1) krt3 = 1
1030 write(ku15)(mj2(imj), imj=1, 45)
write(ku15)(mj2(imj), imj=46, 56)
write(ku15)(bj2(ibj), ibj=1, 18)
rewind(ku15)

if(igt(2).gt.ig1(3)+icl) klt3 = 0

call CROSST(3, ku9, ku10, tdc, carrl3, arrr4,
  nvol, lri, noff, icl, jdq, ipz, jpz, speed,
  ntg, mult, jvol, tuz, dtuz, mqc3, mqc31,
  klt3, krt3, 50, 52, ispt, nkat, sutst,
  ispt1, nkat1, sutst1, dist, iveh, ismt, irt,
  igt, irt1, igt1, igf, kfa, ma, st, st1, nca,
  9, iqit2, qst2, iqst, qsit, nqsu, rqst2,
  iqtt, qstt, iqtit, qstit, nqstt, rqstt,
  kit2, kit3, 10, iqit21, qst21, iqst1, qsit1,
  nqsu1, rqst21, iqtt1, qstt1, iqtit1, qstit1,
```

```
   nqstt1, rqstt1, kit21, kit31, garr1, gdep1,
   garr2, gdep2, garr3, gdep3, harr1, hdep1,
   harr2, hdep2, harr3, hdep3, isp2, nka2,
   suts2, 0, dep31,jdum, dy31, dy32, dy33,
   dy34, dy35, dy36, jvar, ispec)
read(ku15)(mj2(imj), imj=1, 45)
read(ku15)(mj2(imj), imj=46, 56)
read(ku15)(bj2(ibj), ibj=1, 18)
rewind(ku15)
endif

1070 if(jvol.ge.nvol+400.and.kskip.ne.1) then

ma10 = ma(1) - 1
mb10 = ma(2) - 1
mc10 = ma(5) - 1
md10 = ma(6) - 1
mc11 = ma(7) - 1
md11 = ma(8) - 1
mc12 = ma(9) - 1
md12 = ma(10) - 1

call NOVWUP(dep1, dep2, dep3, ismt, nv1, ma10)
call NOVWUP(dep11, dep21, dep31, ismt, nv2,
  mb10)
call NOVWUP(cdep1, cdep2, cdep3, ismt, nv5,
  mc10)
call NOVWUP(ddep1, ddep2, ddep3, ismt, nv6,
  md10)
call NOVWUP(edep1, edep2, edep3, ismt, nv7,
  mc11)
call NOVWUP(fdep1, fdep2, fdep3, ismt, nv8,
  md11)
call NOVWUP(gdep1, gdep2, gdep3, ismt, nv9,
  mc12)
call NOVWUP(hdep1, hdep2, hdep3, ismt, nv10,
  md12)
nwu = nv1+nv2+nv3+nv4+nv5+nv6+nv7+nv8+nv9+nv10

call findep(dep2, ma10)
call findep(dep21, mb10)
call findep(cdep2, mc10)
call findep(ddep2, md10)
call findep(edep2, mc11)
call findep(fdep2, md11)
call findep(gdep2, mc12)
call findep(hdep2, md12)
dep(1) = dep2(ma10)
dep(2) = dep21(mb10)
dep(3) = cdep2(mc10)
dep(4) = ddep2(md10)
dep(5) = edep2(mc11)
dep(6) = fdep2(md11)
dep(7) = gdep2(mc12)
dep(8) = hdep2(md12)
call piksrt(8, dep)
jmin = 1

ddif = dep(8)/5.

dp21 = dep(2) - dep(1)
if(dep(2).gt.1000.and.dp21.lt.100) jmin = 1
if(dep(2).gt.dep(1)+ddif) jmin = 2
if(dep(3).gt.dep(2)+ddif) jmin = 3
if(dep(4).gt.dep(3)+ddif) jmin = 4
if(dep(5).gt.dep(4)+ddif) jmin = 5
220depmin = dep(jmin)
diffd = abs(depmin - depmp)

if(diffd.gt.0.01) then
call NOVSIM(dep2, ma10, depmin, nov1)
call NOVSIM(dep21, mb10, depmin, nov2)
call NOVSIM(cdep2, mc10, depmin, nov5)
call NOVSIM(ddep2, md10, depmin, nov6)
call NOVSIM(edep2, mc11, depmin, nov7)
call NOVSIM(fdep2, md11, depmin, nov8)
call NOVSIM(gdep2, mc12, depmin, nov9)
call NOVSIM(hdep2, md12, depmin, nov10)
novt = nov1+nov2+nov5+nov6+nov7+nov8+nov9+
  nov10
if(novt.ge.nvol+nwu) then
kskip = 1
go to 250
endif
endif
depmp = depmin
endif

250 if(kskip.eq.1) then
do 260 time = depmin,depmin-1000,-0.5
call NOVSIM(dep2, ma10, time, nov1)
call NOVSIM(dep21, mb10, time, nov2)
call NOVSIM(cdep2, mc10, time, nov5)
call NOVSIM(ddep2, md10, time, nov6)
call NOVSIM(edep2, mc11, time, nov7)
call NOVSIM(fdep2, md11, time, nov8)
call NOVSIM(gdep2, mc12, time, nov9)
call NOVSIM(hdep2, md12, time, nov10)
nsim = nov1+nov2+nov3+nov4+nov5+nov6+nov7+
  nov8+nov9+ nov10-nwu
if(nsim.le.nvol) go to 270
260 continue
endif

if(jvol.lt.nvol+1500) go to 1000

270 smtj = time
nos1 = nov1 - nv1
if(nos1.lt.0) nos1 = 0
nos2 = nov2 - nv2
nos5 = nov5 - nv5
nos6 = nov6 - nv6
if(nos6.lt.0) nos6 = 0
nos7 = nov7 - nv7
nos8 = nov8 - nv8
if(nos8.lt.0) nos8 = 0
nos9 = nov9 - nv9
if(nos9.lt.0) nos9 = 0
nos10 = nov10 - nv10

****** Print queue spillback statistics

**arterial,right lane (A dirction)
smpd = smtj - ismt
jcy = int(smpd/icl) + 1
call PRNQS(ku1, 2, icl, lgi, iqi2, qs2, jcy,
  iqs, qsi, nqs,rqs2, prqs, pqs, pqsn, ptqs,
  ismt, smtj, mult)
```

```
call PRNQST(ku1, 2, iqt, qst, iqti, qsti, jcy,
  nqst, rqst, prqst, pqst, pqstn, ptqst, ismt,
  smtj, mult)
call PRNQS(ku1, 3, icl, lgi, iqi3, qs20, jcy,
  iqs0, qsi0, nqs0,rqs20, prqs1, pqs1, pqsn1,
  ptqs1, ismt, smtj, mult)

**arterial,left lane (A dirction)
call PRNQS(ku2, 2, icl, lgi, iqi21, qs21, jcy,
  iqs2, qsi1, nqs2,rqs21, prqs2, pqs2, pqsn2,
  ptqs2, ismt, smtj, mult)
call PRNQS(ku2, 3, icl, lgi, iqi31, qs31, jcy,
  iqs3, qsi2, nqs3,rqs31, prqs3, pqs3, pqsn3,
  ptqs3, ismt, smtj, mult)
call PRNQST(ku2, 3, iqt1, qst1, iqti1, qsti1,
  jcy, nqst1, rqst1, prqst3, pqst3, pqstn3,
  ptqst3, ismt, smtj, mult)

**cross street 1,left lane
call PRNQS(ku5, 3, icl, lri, iqir2, qsr2, jcy,
  iqsr, qsir, nqsr,rqsr2, prqa, pqa, pqna,
  ptqa, ismt, smtj, mult)
call PRNQST(ku5, 3, iqtr, qstr, iqtir, qstir,
  jcy, nqstr,rqstr, prqta, pqta, pqtna, ptqta,
  ismt, smtj, mult)

**cross street 1,right lane
call PRNQS(ku6, 3, icl, lri, iqir21, qsr21,
  jcy, iqsr1, qsir1,nqsr1, rqsr21, prqa1,
  pqa1, pqna1, ptqa1, ismt, smtj, mult)
call PRNQST(ku6, 3, iqtr1, qstr1, iqtir1,
  qstir1, jcy, nqstr1,rqstr1, prqta1, pqta1,
  pqtna1, ptqta1, ismt, smtj, mult)

**cross street 2,left lane
call PRNQS(ku7, 3, icl, lri, iqis2, qss2, jcy,
  iqss, qsis, nqss,rqss2, prqb, pqb, pqnb,
  ptqb, ismt, smtj, mult)
call PRNQST(ku7, 3, iqts, qsts, iqtis, qstis,
  jcy, nqsts,rqsts, prqtb, pqtb, pqtnb, ptqtb,
  ismt, smtj, mult)

**cross street 2,right lane
call PRNQS(ku8, 3, icl, lri, iqis21, qss21,
  jcy, iqss1, qsis1,nqss1, rqss21, ptqb1,
  pqb1, pqnb1, ptqb1, ismt, smtj, mult)
call PRNQST(ku8, 3, iqts1, qsts1, iqtis1,
  qstis1, jcy, nqsts1,rqsts1, prqtb1, pqtb1,
  pqtnb1, ptqtb1, ismt, smtj, mult)

**cross street 3,left lane
call PRNQS(ku9, 3, icl, lri, iqit2, qst2, jcy,
  iqst, qsit, nqsu,rqst2, prqc, pqc, pqnc,
  ptqc, ismt, smtj, mult)
call PRNQST(ku9, 3, iqtt, qstt, iqtit, qstit,
  jcy, nqstt,rqstt, prqtc, pqtc, pqtnc, ptqtc,
  ismt, smtj, mult)

**cross street 3,right lane
call PRNQS(ku10, 3, icl, lri, iqit21, qst21,
  jcy, iqst1, qsit1,nqsu1, rqst21, prqc1,
  pqc1, pqnc1, ptqc1, ismt, smtj, mult)
call PRNQST(ku10, 3, iqtt1, qstt1, iqtit1,
  qstit1, jcy, nqstt1,rqstt1, prqtc1, pqtc1,
  pqtnc1, ptqtc1, ismt, smtj, mult)

*****
if(mult.eq.1) then

write(ku1, 310) ip*incre, jp*incre
310 format(/' offset2=', i3, ',offset3=', i3)

pta1(ip, jp) = ptqs + ptqst + ptqs1
pta2(ip, jp) = ptqs2 + ptqs3 + ptqst3
ptc1(ip, jp) = ptqa + ptqta + ptqa1 + ptqta1
ptc2(ip, jp) = ptqb + ptqtb + ptqb1 + ptqtb1
ptc3(ip, jp) = ptqc + ptqtc + ptqc1 + ptqtc1
ptat(ip, jp) = pta1(ip, jp) + pta2(ip, jp)
ptct(ip, jp) = ptc1(ip, jp) + ptc2(ip, jp) +
  ptc3(ip, jp)
ptst(ip, jp) = ptat(ip, jp) + ptct(ip, jp)
spst(ip, jp) = smpd
nwu(ip, jp) = nwu

nar(ip, jp) = nos1
na1(ip, jp) = nos2
nc11(ip, jp) = nos5
ncr1(ip, jp) = nos6
nc12(ip, jp) = nos7
ncr2(ip, jp) = nos8
nc13(ip, jp) = nos9
ncr3(ip, jp) = nos10

write(ku1, 320) ptat(ip, jp), ptct(ip, jp),
  ptst(ip, jp), smpd
320 format(' P(A)=', f5.2, 'P(C)=', f5.2,
  'P(T)=', f5.2, 'Pd=', f6.0)
endif
*****
if(mult.eq.0) then
write(ku1, 330) smpd
330 format(2x, 'Simulation Period(Total):',
  f6.0, ' sec')
write(ku1, 340) nv1, nv2, nv5, nv6, nv7, nv8,
  nv9, nv10
write(ku1, 350) nos1, nos2, nos5, nos6, nos7,
  nos8,nos9, nos10
340 format(/2x, 'Wi(A1R,A1L,C1L,C1R,C2L,C2R,
  ',' C3L, ', ' C3R) :', /4x, 7(i4, ', '), i4)
350 format(/2x, 'Si(A1R,A1L,C1L,C1R,C2L,C2R,
  ',' C3L, ', ' C3R) :', /4x, 7(i4, ', '), i4)
jtot = nwu + nsim
write(ku1, 355) nwu
355 format(/2x, 'V (warmup) : ', i4, '
  vehicles')
write(ku1, 357) nsim
357 format(/2x, 'V (simulation) : ', i4, '
  vehicles')
write(ku1, 360) jtot
360 format(/2x, 'V (total) : ', i4, '
  vehicles')
endif

420 continue
410 continue
400 continue

*****Print summarized outputs

if(mult.eq.1) then
```

```
write(ku5, 510)
510 format(/2X, 'No.of queue spillback per
  cycle')
call PRNOP1(ku5, 1, 1, incre, ioff1, ioff2,
  joff1, joff2, pta1)
call PRNOP1(ku5, 1, 2, incre, ioff1, ioff2,
  joff1, joff2, pta2)
call PRNOP1(ku5, 2, 1, incre, ioff1, ioff2,
  joff1, joff2, ptc1)
call PRNOP1(ku5, 2, 2, incre, ioff1, ioff2,
  joff1, joff2, ptc2)
call PRNOP1(ku5, 2, 3, incre, ioff1, ioff2,
  joff1, joff2, ptc3)
call PRNOP1(ku5, 3, 9, incre, ioff1, ioff2,
  joff1, joff2, ptat)
call PRNOP1(ku5, 4, 9, incre, ioff1, ioff2,
  joff1, joff2, ptct)
call PRNOP1(ku5, 5, 9, incre, ioff1, ioff2,
  joff1, joff2, ptst)
call PRNOP1(ku2, 6, 9, incre, ioff1, ioff2,
  joff1, joff2, spst)
call PRNOP1(ku2, 5, 9, incre, ioff1, ioff2,
  joff1, joff2, ptst)

write(ku6, 520)
520 format(/2X, 'No.of vehicles simulated')
call PRNOP2(ku6, 1, incre, ioff1, ioff2, joff1,
  joff2, mar)
call PRNOP2(ku6, 2, incre, ioff1, ioff2, joff1,
  joff2, mal)
call PRNOP2(ku6, 5, incre, ioff1, ioff2, joff1,
  joff2, mcl1)
call PRNOP2(ku6, 6, incre, ioff1, ioff2, joff1,
  joff2, mcr1)
call PRNOP2(ku6, 7, incre, ioff1, ioff2, joff1,
  joff2, mcl2)
call PRNOP2(ku6, 8, incre, ioff1, ioff2, joff1,
  joff2, mcr2)
call PRNOP2(ku6, 9, incre, ioff1, ioff2, joff1,
  joff2, mcl3)
call PRNOP2(ku6, 10, incre, ioff1, ioff2,
  joff1, joff2, mcr3)
call PRNOP2(ku2, 99, incre, ioff1, ioff2,
  joff1, joff2, mwu)
endif

stop
end

*****************************************
******SUBROUTINES*********************
*****************************************

subroutine RIGHTL(j1, nvol, dist, iveh, speed,
  jdq, mult, icl, lgi, ku1, arrr2, carrl2, m,
  m1, ma, ntg, jvol, imq1, ki2, ki3, jo, nr,
  at2, arr1, dep1, arr2, dep2, arr3, dep3, i,
  k1, k2, k3, ig, ir, nc, ne, ng, iqi2, iqi3,
  qs2, qs20, nq2, nq3, ncy1, ncy2, ncy3, nka,
  nka0, isp, isp0, nua, nua0, dtua, dtua0,
  suts, suts0, iqs, iqs0, qsi, qsi0, nqs,
  nqs0, rqs2, rqs20, niq, niq0, dtiq, dtiq0,
  nux, nux0, dtux, dtux0, jac, tuz, dtuz,
  jwh1, jwh2, jwh3, nq2i, nuzj1, dtuzj1, n,
  iup, iqt, qst, iqe, iqea, nii, nii0, iqti,
  qsti, nqst, rqst, jor, mqc2, sutss, isps,
  nkas, ismt, nibi, nibj, ixt1, ixt2, ixt3,
  icy, st, st1, nac, noff, krt3, krx1, irs,
  edep3, jdum, dy1, dy2, dy3, jvar)
c processes vehicles in the right arterial lane
dimension arr1(0:1500), dep1(0:1500),
  arr2(0:1500)
dimension dep2(0:1500), arr3(0:1500),
  dep3(0:1500)
dimension arrr2(0:500), carrl2(0:500),
  edep3(0:1500)
dimension ki2(0:1500), ki3(0:1500), zz(0:100),
  zsut(0:100)
dimension dist(20), iveh(20), ir(20), ig(20),
  irs(20)
dimension qs2(100), qs20(100), qst(100),
  qsti(100)
dimension nka(100), nka0(100), nkas(100)
dimension qsi(100), qsi0(100), rqst(100)
dimension suts(0:100, 0:60), suts0(0:100, 0:60)
dimension rqs2(100), rqs20(100), sutss(0:100,
  0:60)
dimension tuz(15), dtuz(15)
dimension ma(15), st(15), st1(15), noff(20)
dimension dy1(0:60), dy2(0:60), dy3(0:60)

C---- Arrival time at intersection 1 -----
C
do 100 m = ma(j1),ma(j1)+ntg
if(ig(2).ge.ismt) jvol = jvol + 1
if(m.eq.1) st0 = 0.0

if(ixt1.eq.1) go to 145
if(ixt2.eq.1.and.imq1.ne.1) go to 150
if(ixt3.eq.1) go to 151

ki2(m) = 0
ki3(m) = jo
IF(nr.eq.0) then
arr1(m) = 2.0 * (m-i)
C
C--- Departure time at intersection 1---
C
icx = 14
jwh = jwh1
c jwh: indicates intersection position (used
  before DEPQLS)
call DEPQLS(k1, ig(1), ir(1), zz, dep1, m, icl,
  arr1, nc, arr2, dep2, iqi2, qs2, ki2, nq2,
  ncy1, iveh(1), zsut, nka, isp, nua, dtua,
  suts, iqs, qsi, nqs, rqs2, niq, dtiq, jdq,
  nux, dtux, icx, jwh, jac, ixt1, tuz, dtuz,
  0, j1, 14, st, st1, jdum, dy1, jvar)
icx = 0
if(imq1.eq.1) then
imq1 = 0
jor = 0
call UPDATE(ne, k2, ig(2), ir(2), zz, dep2, m,
  icl, ncy2)
k2 = k2 - 1
endif

if(ixt1.eq.1) go to 110
145 if(ixt1.eq.1) ixt1 = 0
C
```

```
C--- Arrival time at intersection 2 ------
C
ELSEIF(iup.eq.1) then
arr1(m) = parr1
dep1(m) = pdep1
nr = 0
iup = 0
iqe = 0
iqea = 0
ELSE
arr1(m) = parr1
dep1(m) = pdep1
ENDIF
if(imq1.eq.1) then
imq1 = 0
jor = 0
call UPDATE(ne, k2, ig(2), ir(2), zz, dep2, m,
  icl, ncy2)
k2 = k2 - 1
endif
icx = 79
call ARRQLA(arr2, dep2, arr1, dep1, m, nc,
  dist(1), speed, at2, arrr2, i, n, nr, iup,
  ki2, nq2i, iqt, qst, iqe, iqea, iveh(1),
  icx, nka, isp, suts, nii, iqti, qsti, niq,
  dtiq, nqst, rqst, jdq, arnp, icw, jac,jdum,
  jvar)
icx = 0
if(dep1(m).gt.ir(1)+icl) then
call UPDATE(nc, k1, ig(1), ir(1), zz, dep1, m,
  icl, ncy1)
ixt1 = 1
niq = 0
go to 110
endif
C
C--- Departure time at intersection 2 ----
C
jwh = jwh2
jz2 = 7
call DEPQLS(k2, ig(2), ir(2), zz, dep2, m, icl,
  arr2, ne, arr3, dep3, iqi3, qs20, ki3, nq3,
  ncy2, iveh(1), zsut, nka0, isp0, nua0,
  dtua0, suts0, iqs0, qsi0, nqs0, rqs20, niq0,
  dtiq0, jdq, nux0, dtux0, icx, jwh, jac,
  ixt2, tuz, dtuz, 0, 13, jz2, st, st1, jdum,
  dy2,jvar)
if(ixt2.eq.1) then
jor = 0
endif

mprod = (nua0-1)*(niq0-1)
call UPNKA(k2, ig(2), ir(2), zz, dep2, m, jdq,
  lgi, icl, arr2, ne, nq2i, nq3, ncy2, zsut,
  iveh(1), nka, isp, suts, mprod, jac,
  tuz,dtuz, nibi, nibj)
C
C--- Arrival time at intersection 3 ------
C
if(ixt2.eq.1) go to 110
150 if(ixt2.eq.1) then
ixt2 = 0
krx1 = 0

if(tuz(7).gt.ig(2)) then
dtuz(7) = tuz(7) - ig(2)
dep2(m) = tuz(7) + 2.04
dslgi = lgi - dtuz(7)
jveh = iveh(1) * 2
if(dslgi.lt.jveh) nibj = 1
ircl1 = ir(2)+icl

if(dep2(m).ge.ircl1) then
call UPDATE(ne, k2, ig(2), ir(2), zz, dep2, m,
  icl, ncy2)
call UPDATE(ng, k3, ig(3), ir(3), zz, dep3, m,
  icl, ncy3)
nka(isp) = 0
isp = isp + 1
nka0(isp0) = 0
isp0 = isp0 + 1
ixt2 = 1
krt3 = 1
krx1 = 1
go to 110
endif
endif
endif

120 if((nac.eq.1.and.(k2.eq.2.or.jdq.eq.
  11.and.(k2.eq.4.or.k2.eq.5))).or.(nac.eq.2.a
  nd.(k2.eq.2.or.k2.eq.4.or.jdq.eq.11.and.(k2.
  eq.5.or.k2.eq.7.or.k2.eq.8.or.k2.eq.10))))
  then

jo = jo + 1
carrl2(jo) = dep2(m)
if(ma(7).ge.iveh(1)) then
ijor = iveh(1)-(jor+1)
call FINDST(ma(7), edep3, iveh(1), ijor, st0)
if(st0.gt.dep2(m)) then
sss = st0
ssd = sss - dep2(m)
if(sss.ge.ir(2)+icl) then
imq1 = 1
elseif(ssd.gt.0) then
nux0 = 1
dtux0 = ssd
endif
if(ssd.gt.lgi-20) nibi = 1
endif
endif
jor = jor + 1
go to 155
endif

if(dep2(m).lt.arr2(m)) dep2(m) = arr2(m)

call ARRNT(arr3, dep3, arr2, dep2, m, ne,
  dist(2), speed, nq3, jac, jdum,jvar)
C
C--- Departure time at intersection 3 ----
C
jwh = jwh3
call DEPART(k3, ig(3), ir(3), zz, dep3, m, icl,
  arr3, ng, ncy3, jac,ixt3, jdum, dy3, jvar)
mprod = 1
call UPDNKA(k3, ig(3), ir(3), zz, dep3, m, jdq,
  lgi, icl, arr3,ng, nq3, 0, ncy3, zsut,
  iveh(1), nka0, isp0, suts0, mprod, jac)
```

```
if(ixt3.eq.1) go to 110
151 if(ixt3.eq.1) ixt3 = 0
C
C--- Print arrival and departure time at each
  intersection ---------
C
155 if(nr.ne.0) then
parr1 = arr1(m)
pdep1 = dep1(m)
arr1(m) = 0.
dep1(m) = 0.
endif

ki3p = jo

if(dep3(m).lt.arr3(m)) dep3(m) = arr3(m)
if(mult.eq.0) then
call FRINT(arr1, dep1, arr2, dep2, arr3, dep3,
  ku1, m, i, jo, nq2i)
endif
C
if(ncy2.lt.isp) then
do 160 ik = isp,isp
nka(ik) = nka(ik) + 1
160 continue
else
do 161 ik = isp,ncy2
nka(ik) = nka(ik) + 1
161 continue
endif

if(ki3p.ne.ki3(m)) go to 164
if(ncy3.lt.isp0) then
do 162 ik = isp0,isp0
nka0(ik) = nka0(ik) + 1
162 continue
else
do 163 ik = isp0,ncy3
nka0(ik) = nka0(ik) + 1
163 continue
endif
164 continue

if(nr.ne.0) then
arr1(m) = parr1
dep1(m) = pdep1
endif

if(imq1.eq.1) go to 110

100 continue
110 ma(j1) = m
if(ig(2).ge.ismt) jvol = jvol - 1
if(imq1.eq.1.and.ixt1.ne.1.and.ixt2.ne.1.and.ix
  t3.ne.1) then
ma(j1) = m + 1
if(ig(2).ge.ismt) jvol = jvol + 1
endif
return
end
*******
subroutine LEFTL(j1, nvol, dist, iveh, speed,
  jdq, mult, icl, lgi, ku2, arrl3, carrl1, m,
  ma, ntg, jvol, imq2, imq3, ki21, ki31, krt2,
  klt3, ixt1, ixt2, ixt3, jo1, nl, at3, arr11,
  dep11, arr21, dep21, arr31, dep31, ii, k11,
  k21, k31, ig1, ir1, nc1, ne1, ng1, iqi21,
  iqi31, qs21, qs31, nql2, nql3, ncy11, ncy21,
  ncy31, nka1, nka2, isp1, isp2, nua1, nua2,
  dtua1, dtua2, suts1, suts2, iqs2, iqs3,
  qsi1, qsi2, nqs2, nqs3, rqs21, rqs31, niq1,
  niq2, dtiq1, dtiq2, nux1, nux2, dtux1,
  dtux2, jac, tuz, dtuz, jwh1, jwh2, jwh3,
  nql3i, nuzj2, dtuzj2, nn, iup1, iqt1, qst1,
  iqe1, iqea1, nii2, iqti1, qsti1, nqst1,
  rqst1, jor1, mqc1, mqc2, sutsr, ispr, nkar,
  ismt, carrl3, jo2, nuzj3, dtuzj3, jor2,
  sutst, ispt, nkat, icz, st, st1, nac, noff,
  krx1, krx2, nux3, dtux3, irr, irt, cdep3,
  gdep3, jdum, dy1, dy2, dy3, jvar)
c processes vehicles in the left arterial lane
dimension arr11(0:1500), dep11(0:1500),
  arr21(0:1500)
dimension dep21(0:1500), arr31(0:1500),
  dep31(0:1500)
dimension cdep3(0:1500), gdep3(0:1500)
dimension arrl3(0:500), carrl1(0:500)
dimension ki21(0:1500), ki31(0:1500),
  zz(0:100), zsut(0:100)
dimension dist(20), iveh(20), ir1(20), ig1(20),
  irr(20), irt(20)
dimension qs21(100), qs31(100), qst1(100),
  qsti1(100)
dimension nka1(100), nka2(100), nkar(100),
  nkat(100)
dimension qsi1(100), qsi2(100), rqst1(100)
dimension suts1(0:100, 0:60), suts2(0:100,
  0:60)
dimension sutsr(0:100, 0:60), sutst(0:100,
  0:60)
dimension rqs21(100), rqs31(100), carrl3(0:500)
dimension tuz(15), dtuz(15)
dimension ma(15), st(15), st1(15), noff(20)
 dimension dy1(0:60), dy2(0:60), dy3(0:60)

C---- Arrival time at intersection 1 -----
C
do 200 m = ma(j1),ma(j1)+ntg
if(ig1(2).ge.ismt) jvol = jvol + 1
if(m.eq.1) st0 = 0.0
if(m.eq.1) st0i = 0.0

if(ixt1.eq.1.and.imq2.ne.1) go to 245
if(ixt2.eq.1) go to 250
if(ixt3.eq.1.and.imq3.ne.1) go to 251

ki21(m) = ii + jo1
ki31(m) = jo1
C
IF(nl.eq.0) then
arr11(m) = 2.0 * (m-ii)
C
C--- Departure time at intersection 1---
C
if(imq2.eq.1) then
imq2 = 0
call UPDATE(nc1, k11, ig1(1), ir1(1), zz,
  dep11, m, icl, ncy11)
k11 = k11 - 1
```

```
jor1 = 0
endif

icx = -2
jwh = jwh1
jz2 = 5
call DEPQLS(k11, ig1(1), ir1(1), zz, dep11, m,
  ic1, arr11, nc1, arr21, dep21, iqi21, qs21,
  ki21, nql2, ncy11, iveh(1), zsut, nka1,
  isp1, nua1, dtua1, suts1, iqs2, qsi1, nqs2,
  rqs21, niq1, dtiq1, jdq, nux1, dtux1, icx,
  jwh, jac, ixt1, tuz, dtuz,0, 13, jz2, st,
  st1, jdum, dy1, jvar)
icx = 0
if(ixt1.eq.1) then
jor1 = 0
endif

if(ixt1.eq.1) then
if(imq3.eq.1) then
imq3 = 0
call UPDATE(ng1, k31, ig1(3), ir1(3), zz,
  dep31, m, ic1, ncy31)
k31 = k31 - 1
jor2 = 0
endif
go to 210
endif

245 if(ixt1.eq.1) then
ixt1 = 0
 if(tuz(5).gt.ig1(1)) then
dep11(m) = tuz(5) + 2.04
ircl1 = ir1(1)+ic1
if(dep11(m).ge.ircl1) then
 call UPDATE(nc1, k11, ig1(1), ir1(1), zz,
  dep11, m, ic1, ncy11)
 call UPDATE(ne1, k21, ig1(2), ir1(2), zz,
  dep21, m, ic1, ncy21)
 call UPDATE(ng1, k31, ig1(3), ir1(3), zz,
  dep31, m, ic1, ncy31)
nka1(isp1) = 0
isp1 = isp1 + 1
nka2(isp2) = 0
isp2 = isp2 + 1
ixt1 = 1
krt2 = 1
klt3 = 1
go to 210
endif
endif
endif
C
C---- Arrival time at intersection 2 -----
C
220 if((nac.eq.1.and.(k11.eq.2.or.jdq.eq.
  11.and. (k11.eq.4.or.k11.eq.5))).or.
  (nac.eq.2.and. (k11.eq.2.or.k11.eq.4.or.
  jdq.eq.11.and. (k11.eq.5.or.k11.eq.7.or.
  k11.eq.8.or.k11.eq.10)))) then
jo1 = jo1 + 1
carr11(jo1) = dep11(m)
if(ma(5).ge.iveh(1)) then
ijor1 = iveh(1)-(jor1+1)
call FINDST(ma(5), cdep3, iveh(1), ijor1, st0)
 if(st0.gt.dep11(m)) then
sss = st0
ssd = sss - dep11(m)
if(sss.ge.ir1(1)+ic1) then
imq2 = 1
elseif(ssd.gt.0) then
nux1 = 1
dtux1 = ssd
endif
endif
endif
jor1 = jor1 + 1
go to 255
endif

115 call ARRNTE(arr21, dep21, arr11, dep11, m,
  nc1, dist(1), speed, nql2, iveh(1), nka1,
  isp1, suts1, ki21, jac, jdum, jvar)
icx = 0
C
C--- Departure time at intersection 2 ----
C
jwh = jwh2
call DEPQLS(k21, ig1(2), ir1(2), zz, dep21, m,
  ic1, arr21, ne1, arr31, dep31, iqi31, qs31,
  ki31, nql3, ncy21, iveh(1), zsut, nka2,
  isp2, nua2, dtua2, suts2, iqs3, qsi2, nqs3,
  rqs31, niq2, dtiq2, jdq, nux2, dtux2, icx,
  jwh, jac, ixt2, tuz, dtuz,0, j1, 14, st,
  st1, jdum, dy2, jvar)

mprod = (nua2-1)*(nii2-1)*(niq2-1)
call UPDNKA(k21, ig1(2), ir1(2), zz, dep21, m,
  jdq, lgi, ic1, arr21, ne1, nql2, nql3,
  ncy21, zsut, iveh(1), nka1, isp1,
  suts1,mprod, jac)

if(imq3.eq.1) then
imq3 = 0
call UPDATE(ng1, k31, ig1(3), ir1(3), zz,
  dep31, m, ic1, ncy31)
k31 = k31 - 1
jor2 = 0
endif
C
C---- Arrival time at intersection 3 -----
C
if(ixt2.eq.1) go to 210
250 if(ixt2.eq.1) ixt2 = 0

ELSEIF(iup1.eq.1) then
arr11(m) = parr11
dep11(m) = pdep11
arr21(m) = parr21
dep21(m) = pdep21
nl = 0
iup1 = 0
iqe1 = 0
iqea1 = 0
ELSE
arr11(m) = parr11
dep11(m) = pdep11
arr21(m) = parr21
dep21(m) = pdep21
ENDIF
```

```
if(imq3.eq.1) then
imq3 = 0
call UPDATE(ng1, k31, ig1(3), ir1(3), zz,
  dep31, m, icl, ncy31)
k31 = k31 - 1
jor2 = 0
endif

if(dep21(m).lt.arr21(m)) dep21(m) = arr21(m)
C
icx = 32
call ARRQLA(arr31, dep31, arr21, dep21, m, ne1,
  dist(2), speed, at3, arr13, ii, nn, nl,
  iup1, ki31, nql3i, iqt1, qst1, iqe1, iqea1,
  iveh(1), icx, nka2, isp2, suts2, nii2,
  iqti1, qsti1, niq2, dtiq2, nqst1, rqst1,
  jdq, amp, icw, jac, jdum, jvar)
icx = 0
C
C--- Departure time at intersection 3 ----
C
jwh = jwh3
call DPARTA(k31, ig1(3), ir1(3), zz, dep31, m,
  icl, arr31, ng1, ncy31, jac, ixt3, jwh, tuz,
  dtuz, nux3, dtux3, jdum, dy3, jvar, icx)
mprod = 1
call UPDNKA(k31, ig1(3), ir1(3), zz, dep31, m,
  jdq, lgi, icl, arr31, ng1, nql3i, 0, ncy31,
  zsut, iveh(1), nka2, isp2, suts2,mprod, jac)

if(ixt3.eq.1) jor2 = 0

if(ixt3.eq.1) go to 210
251 if(ixt3.eq.1) then
ixt3 = 0
krx1 = 0
krx2 = 0

if(tuz(9).gt.ig1(3)) then
dep31(m) = tuz(9) + 2.04
ircl1 = ir1(3)+icl
if(dep31(m).ge.ircl1) then
call UPDATE(ng1, k31, ig1(3), ir1(3), zz,
  dep31, m, icl, ncy31)
ixt3 = 1
krx1 = 1
krx2 = 1
go to 210
endif
endif
endif

230 if((nac.eq.1.and.(k31.eq.2.or.jdq.eq.
  11.and.(k31.eq.4.or.k31.eq.5))).or.(nac.eq.2
  .and.(k31.eq.2.or.k31.eq.3.or.jdq.eq.11.and.
  (k31.eq.5.or.k31.eq.6.or.k31.eq.8.or.k31.eq.
  9)))) then

jo2 = jo2 + 1
carrl3(jo2) = dep31(m)
if(m.ge.123) then
endif

if(ma(9).ge.iveh(1)) then
ijor2 = iveh(1)-(jor2+1)
call FINDST(ma(9), gdep3, iveh(1), ijor2, st0i)

if(st0i.gt.dep31(m)) then
sss = st0i
ssd = sss - dep31(m)
if(sss.ge.ir1(3)+icl) then
imq3 = 1
elseif(ssd.gt.0) then
nux3 = 1
dtux3 = ssd
endif
endif
endif
jor2 = jor2 + 1
endif
C
C------ Print arrival and departure time at
  each intersection ---------
C
255 if(nl.ne.0) then
parr11 = arr11(m)
pdep11 = dep11(m)
parr21 = arr21(m)
pdep21 = dep21(m)
arr11(m) = 0.
dep11(m) = 0.
arr21(m) = 0.
dep21(m) = 0.
endif

ki21p = ii + jo1
ki31p = jo1
if(dep31(m).lt.arr31(m)) dep31(m) = arr31(m)
if(mult.eq.0) then
call PRN(arr11, dep11, arr21, dep21, arr31,
  dep31, ku2, m, ii,jo1, nql2, nql3, nql3i)
endif
if(ki21p.ne.ki21(m)) go to 261
do 260 ik = isp1,ncy21
nka1(ik) = nka1(ik) + 1
260 continue
261 continue

if(ki31p.ne.ki31(m)) go to 266
do 265 ik = isp2,ncy31
nka2(ik) = nka2(ik) + 1
265 continue
266 continue
C
if(nl.ne.0) then
arr11(m) = parr11
dep11(m) = pdep11
arr21(m) = parr21
dep21(m) = pdep21
endif
C
if(imq2.eq.1) go to 210
if(imq3.eq.1) go to 210
200 continue
210 ma(j1) = m
if(ig1(2).ge.ismt) jvol = jvol - 1
if(imq2.eq.1.or.imq3.eq.1) then
ma(j1) = m + 1
if(ig1(2).ge.ismt) jvol = jvol + 1
```

```
endif
return
end
*******
subroutine findep (dep, mxl)
c finds arrival or departure time of previous
  vehicle
dimension dep(0:1500)
100 if(dep(mxl).eq.0) then
mxl = mxl - 1
if(mxl.le.0) then
mxl = 0
go to 200
endif
go to 100
endif
200 return
end
*******
subroutine NOVSIM(dep, mxl, time, novs)
c counts the number of vehicles simulated
dimension dep(0:1500)
if(mxl.eq.0) then
novs = 0
go to 200
endif
do 100 ij = mxl,1,-1
if(time.gt.dep(ij).and.dep(ij).gt.0.1) then
novs = ij
go to 200
endif
if(dep(ij).gt.time.and.dep(ij-1).le.time.and.
  dep(ij-1).gt.0.1) then
novs = ij - 1
if(ij.eq.0) go to 200
ijs = ij
250 if(dep(ijs).lt.0.01) then
ijs = ijs - 1
novs = ijs - 1
go to 250
endif
go to 200
endif
100 continue
200 return
end
*******
subroutine NOVWUP(dep1, dep2, dep3, iwp, novs,
  mxl)
c counts the number of warm-up vehicles
dimension dep1(0:1500), dep2(0:1500),
  dep3(0:1500)
if(mxl.eq.0) then
novs = 0
go to 200
endif
do 100 ij = 1,600
if(dep1(ij).lt.0.01.and.dep2(ij).lt.0.01.and.
  dep3(ij).lt.0.01) then
novs = ij - 2
go to 200
endif
if(dep2(ij).gt.iwp) then
novs = ij - 1
go to 200
endif
100 continue
200 return
end
*******
subroutine piksrt(n, dep)
c sorts n values in ascending order
dimension dep(10)
do 12 j = 2,n
a = dep(j)
do 11 i = j-1,1,-1
if(dep(i).le.a) go to 10
dep(i+1) = dep(i)
11 continue
i = 0
10 dep(i+1) = a
12 continue
return
end
*******
subroutine ARRQLA(arr, dep, arrp, depp, m, nce,
  dist, speed, at, arrr, j, n, nnn, iup, ki,
  nq, iqt, qst, iqe, iqea, iveh1, icx, nka,
  isp, suts, nii, iqti, qsti, niq, dtiq, nqst,
  rqst, jdq, arrrnp, icw, jac,jdum, jvar)
c calculates arrival time of a vehicle when
  there are turn-in movements
dimension dep(0:1500), depp(0:1500),
  arr(0:1500), arrp(0:1500)
dimension arrr(0:500), ki(0:1500), qst(100),
  qsti(100)
dimension nka(100), suts(0:100, 0:60),
  rqst(100)
wid = 40.
if(jac.eq.2) wid = 60.
arrrnp = arrr(n)
if(arr(m-1).eq.dep(m-1).and.arr(m-1).ne.0.) nce
  = 0
ipo = 1
c ipo: incator to calculate number of queue in
  NOQUE

call NOQUE(arr, dep, 0, arrr, iveh1, ki, m, m1,
  nq, n, iqe, iqea, ipo, 0)
ipo = 0
call FINDEV(m, dep, iveh1, lks, st11)
jsubs = iveh1 - 1
call FINDEV(m, dep, jsubs, lks, st2)

IF(nq.ge.iveh1+1.and.arr(m-1).eq.arrr(n-1))
  then
iqt = iqt + 1
qst(iqt) = arrr(n-1)
dids = abs(depp(m)-st11)
if(dids.lt.0.001) then
iqt = iqt - 1
elseif(depp(m).lt.st11) then
niq = 1
nqst = nqst + 1
iqt = iqt - 1
rqst(nqst) = arrr(n-1)
dtiq = st11 - depp(m)
depp(m) = st11
endif
```

```
if(arrr(n).eq.0.) go to 110
iqea = 1
if(arrr(n).lt.arr(m-1)) iqea = 0

call NOQUE(arr, dep, 0, arrr, iveh1, ki, m, m1,
  nq, n, iqe, iqea,ipo, 0)
if(nq.ge.iveh1+1) then
iqe = 1
arrr(n) = st11 + 1.0
if(arrr(n).ge.depp(m) ) then
call NEXTTI(n, arrr, iup, nnn, iqe, nii, niq)
go to 150
endif
endif
ENDIF

110 if(m.ge.2) nce = nq

dcs = dist+wid-20.*nce
if(m.eq.1) dcs = dist + wid

if(jvar.eq.1) then
call ranspd(jdum, nce, arrp, depp, dcs, speed,
  icx)
else
call cspeed(m, nce, arrp, depp, speed, dcs,
  icx)
endif

at = depp(m) + dcs/speed

intv = int(iveh1*0.3)
nisp = nka(isp)
nispt = nka(isp+1)
IF(nce.ne.0) then

atx = depp(m) + (wid+dist-(nisp-1)*20.)/speed
atxt = depp(m) + (wid+dist-(nispt-1)*20.)/speed

if(nisp.le.iveh1) then
 if(arr(m-1).ne.0.0) then
if(arr(m-1).ne.suts(isp, nka(isp)-1).and.
  atx.lt.suts(isp, nka(isp)-1).and.nisp.gt.nq)
  then
at = atx
elseif(arr(m-1).eq.suts(isp, nka(isp)-1).and.
  atxt.lt.suts(isp+1, nka(isp+1)-1).and.
  arr(m-1).ne.suts(isp+1, nka(isp+1)-1)) then
at = atxt
endif
 endif
 if(arr(m-1).eq.0.0.and.nisp.gt.nq) then
if(arr(m-2).ne.suts(isp, nka(isp)-1)) then
at = atx
endif
endif
endif

if(nisp.eq.iveh1+1.and.depp(m).lt.suts(isp,
  nka(isp)-1). and.nq.ge.intv) then
 if(arr(m-1).ne.0.0.and.arr(m-1).ne.suts(isp,
  nka(isp)-1))then
 at = depp(m) + wid/speed
 elseif(arr(m-1).eq.0.0.and.arr(m-
  2).ne.suts(isp, nka(isp)-1)) then
 at = depp(m) + wid/speed
 endif
endif

ENDIF

if(niq.eq.1.and.arrr(n).ge.depp(m)) go to 130
if(niq.eq.1.and.arrr(n).ne.0.0) then
call NEXTTI(n, arrr, iup, nnn, iqe, nii, niq)
go to 150
endif

130 if(arrr(n).lt.depp(m).and.arrr(n).ne .0.)
  then

if(nq.ge.iveh1) then
iqea = 1
if(arrr(n).lt.arr(m-1)) iqea = 0
call NOQUE(arr, dep, 0, arrr, iveh1, ki, m, m1,
  nq, n, iqe, iqea,ipo, 0)
if(nq.gt.0.and.arrr(n).gt.arr(m-1)) nq = nq - 1
endif

dcs = dist+wid-20.*nce

if(jvar.eq.1) then
call ranspd(jdum, nce, arrp, depp, dcs, speed,
  icx)
else
call cspeed(m, nce, arrp, depp, speed, dcs,
  icx)
endif

if(20.*nq.gt.dist) then
arrr(n) = arrrmp + wid/speed
else
arrr(n) = arrrmp + dcs/speed
endif

if(nce.ne.0.and.nisp.gt.nq) then
att = arrrmp + (wid+dist-(nisp-1)*20.)/speed
if(nisp.le.iveh1.and.att.lt.suts(isp, nka(isp)-
  1).and.arr(m-1).ne.suts(isp, nka(isp)-1))
  then
if(att.gt.arr(m-1)) then
arrr(n) = att
endif
endif
endif
if(nisp.eq.iveh1+1.and.arrrmp.lt.suts(isp,
  nka(isp)-1). and.nq.ge.intv.and.arr(m-
  1).ne.suts(isp, nka(isp)-1)) then
 arrr(n) = arrrmp + wid/speed
endif
if(arrr(n).lt.arrr(n-1).and.nnn.eq.0) then
arrr(n) = arrr(n-1) + 1.0
endif
tt = arrrmp + (dist+wid)/30.9
if(nnn.eq.0.and.tt.gt.dep(m-1).and.dep(m-
  1).gt.0.1) then
arrr(n) = tt
endif

ELSE
 go to 150
```

```
ENDIF

if(nii.eq.1.and.arrr(n).lt.st11) then
arrr(n) = st11
if(arrr(n).ge.depp(m) ) then
call NEXTTI(n, arrr, iup, nnn, iqe, nii, niq)
go to 150
endif
go to 151
endif
if(iqe.eq.1) then
arrr(n) = arrr(n-1) + 1.0
iqe = 0
if(arrr(n).ge.depp(m) ) then
call NEXTTI(n, arrr, iup, nnn, iqe, nii, niq)
go to 150
endif
endif

151 if(arrr(n).lt.arr(m-1).and.arrr(n).lt.at)
  then
arrr(n) = arrr(n-1) + 1.0
if(arrr(n).lt.arr(m-1).and.arrr(n).lt.at) then
call NEXTTI(n, arrr, iup, nnn, iqe, nii, niq)
endif
endif
iqea = 1
if(arrr(n).lt.arr(m-1)) iqea = 0
call NOQUE(arr, dep, 0, arrr, iveh1, ki, m, m1,
  nq, n, iqe, iqea, ipo, 0)
if(nq.gt.0.and.arrr(n).gt.arr(m-1)) nq = nq - 1
if(m.ge.2) nce = nq
at = depp(m) + (dist+wid-20.*nce)/speed
if(20.*nce.gt.dist) at = depp(m) + wid/speed
if(at.lt.arr(m-1)+1.0) at = arr(m-1) + 1.0

IF(at.ge.arrr(n).and.arr(m-1).le.arrr(n)) then
if(nka(isp).eq.iveh1+1.and.arrr(n).lt.st2-
  1.1.and.arr(m-1).ne.suts(isp, nka(isp)-1))
  then
if(nq.eq.iveh1+1) go to 160
if(arrr(n).lt.dep(m-lks+1)) go to 160
if(suts(isp, nka(isp))-arrr(n).lt.jdq) go to
  170
iqti = iqti + 1
qsti(iqti) = arrr(n)
170 nii = 1
160 if(depp(m).lt.st2.and.arr(m-1).ne.suts(isp,
  nka(isp)-1)) then
if(suts(isp, nka(isp))-arrr(n).lt.jdq) go to
  190
nqst = nqst + 1
if(iqti.gt.0) iqti = iqti - 1
rqst(nqst) = arrr(n)
190niq = 1
dtiq = suts(isp, nka(isp)) - depp(m)
depp(m) = suts(isp, nka(isp))
endif
endif

if(nka(isp).eq.iveh1+2.and.arrr(n).lt.st2-
  1.1.and.arr(m-1).ne.st2-1.1) then
arrr(n) = st2-1.1

if(st2-1.1.eq.depp(m))then
call NEXTTI(n, arrr, iup, nnn, iqe, nii, niq)
go to 150
endif
nii = 1
if(arrr(n).ge.depp(m)) then
call NEXTTI(n, arrr, iup, nnn, iqe, nii, niq)
go to 150
endif
endif

nnn = nnn + 1
j = j + 1
nce = nce + 1
at = depp(m) + (dist+wid-20.*nce)/speed
if(20.*nce.gt.dist) at = depp(m) + wid/speed

if(nq.ge.iveh1+1) then
iqea = 1
call NOQUE(arr, dep, 0, arrr, iveh1, ki, m, m1,
  nq, n, iqe, iqea,ipo, 0)
if(nq.ge.iveh1+1) then
iqe = 1
arrr(n) = st11 + 1.0
if(arrr(n).ge.depp(m)) then
 nnn = nnn - 1
 j = j - 1
 nce = nce - 1
 call NEXTTI(n, arrr, iup, nnn, iqe, nii, niq)
 go to 150
endif
endif
endif
if(nnn.eq.1) then
arr(m+1) = at
else
arr(m+1) = arr(m)
endif
arr(m) = arrr(n)

n = n + 1
diff = arrr(n) - arrrmp
if(diff.gt.25.0) iup = 1
if(arrr(n).eq.0.) iup = 1
ELSE
150 arr(m) = at
nii = 0
nce = nce + 1
ENDIF

if(arr(m).le.arr(m-1).and.arr(m).gt.0.) then
arr(m) = arr(m-1) + 1.0
endif

m10 = m - 1
call findep(arr, m10)
if(arr(m).lt.arr(m10)+0.5) then
arr(m) = arr(m10) + 0.5
endif

return
end
*******
subroutine NOQUE(arr, dep, depp, arrr, iveh1,
  ki, m, m1, nq, n, iqe,iqea, ipo, jgt)
c countes the number of queued vehicles.
```

```
dimension arr(0:1500), dep(0:1500),
  depp(0:1500)
dimension ki(0:1500), arrr(0:500)
ki(0) = 0
if(m.gt.1) then
m1 = m - 1
time = arr(m1)
if(ipo.eq.19) time = depp(m)
na = m1
if(iqe.eq.1.or.iqea.eq.1) then
if(ipo.eq.0) then
time = arrr(n)
na = m
endif
endif
110 if(arr(m1).eq.0) then
m1 = m1 - 1
if(ipo.ne.19) time = arr(m1)
na = m1
if(ipo.eq.9.and.nq.ge.iveh1+1) then
jgt = 1
go to 100
endif
go to 110
endif
if(ipo.eq.10) then
time = arr(m)
na = m
endif
if(ipo.eq.29) then
time = dep(m)
na = m - 1
endif
if(dep(m1).le.time.and.dep(m1).ne.0) then
nd = m1
go to 200
endif
do 300 ij = 0,m1-1
if(dep(ij+1).gt.time.and.dep(ij).le.time) then
nd = ij
if(ij.eq.0) go to 200
ijs = ij
250 if(dep(ijs).eq.0.) then
ijs = ijs - 1
nd = ijs
go to 250
endif
go to 200
endif
300 continue
200 nq = na - ki(na) - (nd - ki(nd))
100 continue
endif
return
end
*******
subroutine NEXTTI(n, arrr, iup, nnn, iqe, nii,
  niq)
c finds the next available turn-in vehicle
dimension arrr(0:500)
iloop = 0
100 np = n
200 n = n + 1
iloop = iloop + 1
if(iloop.gt.30) then
write(10, *)'stopped by endless go to (in
  NEXTTI)', 'n, np,arrr(n), arrr(np)', n, np,
  arrr(n), arrr(np)
stop
endif
if(arrr(n).eq.0.0) go to 300
if(arrr(n).lt.arrr(np)) go to 200
diff = arrr(n) - arrr(np)
if(diff.lt.25.) go to 100
300 iup = 1
nnn = 0
iqe = 0
nii = 0
niq = niq
return
end
*******
subroutine DEPART(k, ig, ir, z, dep, m, icl,
  arr, nce, ncy, jac, ixt,jdum, dphy, jvar)
c calculates vehicle departure times
dimension z(0:100),dep(0:1500), arr(0:1500)
dimension zsut(0:100), dphy(0:60)
k = k + 1
call DPHDWY(z, zsut)
if(jvar.eq.1.and.k.eq.1.or.k.eq.2) then
call nordev(jdum, dphy, icx)
endif

ircl = ir + icl*(2-jac)
if(arr(m).le.ircl) then
if(jvar.eq.0) then
dep(m) = z(k) + ig
else
dep(m) = dphy(k) + ig
endif
if(dep(m).gt.ircl)then
call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
ixt = 1
endif
if(arr(m).ge.dep(m)) dep(m) = arr(m)
elseif(arr(m).gt.ircl) then
call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
ixt = 1
if(arr(m).ge.dep(m)) dep(m) = arr(m)
endif

m10 = m - 1
call findep(dep, m10)
if(dep(m).lt.dep(m10)+0.7) then
dep(m) = dep(m10) + 0.7
endif

ircl = ir + icl*(2-jac)
if(k.gt.2.and.arr(m-1).eq.dep(m-1).and.arr(m-
  1).gt.1.and.
  arr(m).gt.ig.and.arr(m).le.ircl) then
dep(m) = arr(m)
endif

if(dep(m).gt.ircl) then
call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
ixt = 1
```

```
endif
return
end
*******
subroutine DPARTA(k, ig, ir, z, dep, m, icl,
  arr, nce, ncy, jac, ixt,jwh, tuz, dtuz, nux,
  dtux, jdum, dphy, jvar, icx)
c calculates vehicle departure times
dimension z(0:100), dep(0:1500), arr(0:1500)
dimension zsut(0:100), tuz(15), dtuz(15),
  dphy(0:60)
k = k + 1
call DPHDWY(z, zsut)
if(jvar.eq.1.and.k.eq.1.or.k.eq.2) then
call nordev(jdum, dphy, icx)
endif
ircl = ir + icl*(2-jac)
if(arr(m).le.ircl) then
if(jvar.eq.0) then
dep(m) = z(k) + ig
else
dep(m) = dphy(k) + ig
endif
if(nux.eq.1) dep(m) = dep(m) + dtux
 if(tuz(9).gt.ig.and.jwh.eq.22) then
dep(m) = dep(m) + dtuz(9)
endif

if(dep(m).gt.ircl)then
call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
nux = 0
ixt = 1
endif
if(arr(m).ge.dep(m)) dep(m) = arr(m)
elseif(arr(m).gt.ircl) then
call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
nux = 0
ixt = 1
if(arr(m).ge.dep(m)) dep(m) = arr(m)
endif

m10 = m - 1
call findep(dep, m10)
if(dep(m).lt.dep(m10)+0.7) then
dep(m) = dep(m10) + 0.7
endif

ircl = ir + icl*(2-jac)
if(k.gt.2.and.arr(m-1).eq.dep(m-1).and.arr(m-
  1).gt.1.and.
  arr(m).gt.ig.and.arr(m).le.ircl) then
dep(m) = arr(m)
endif

if(dep(m).gt.ircl) then
call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
nux = 0
ixt = 1
endif
return
end
*******
subroutine UPDNKA(k, ig, ir, z, dep, m, jdq,
  lgi, icl, arr, nce, nq, nqd, ncy, zsut,
  iveh1, nka, isp, suts, mprod, jac)
c updates nka
dimension z(0:100),dep(0:1500), arr(0:1500)
dimension zsut(0:100)
dimension nka(100), suts(0:100, 0:60)
ircl = ir+icl*(2-jac)
if(nqd.ge.iveh1+1) go to 100
if(mprod.eq.0) go to 100
if(nq.eq.0.and.arr(m).gt.ig.and.arr(m).le.ircl)
  then
dep(m) = arr(m)
endif
100 intv = int(iveh1*0.3)

if(suts(isp, nka(isp)).lt.arr(m)) then
suts(isp, nka(isp)) = arr(m)
endif
if(suts(isp, nka(isp)).gt.dep(m)) then
suts(isp, nka(isp)) = dep(m)
endif
if(nka(isp).ge.intv.and.arr(m).gt.suts(isp,
  nka(isp)-1))then
suts(isp, nka(isp)) = arr(m)
endif
c
if(m.gt.1.and.isp.le.2) then
if(dep(m-1).ne.0.and.arr(m-1).ne.0.and.dep(m-
  1).eq.arr(m-1).and.dep(m).ne.arr(m)) then
nka(isp) = 0
isp = isp + 1
endif
endif
if((jdq.eq.7.and.lgi.ge.30).or.(jdq.eq.9.and.lg
  i.ge.50).or.(jdq.eq.11.and.lgi.ge.70)) then
isp = ncy
else
if(nka(isp).ge.iveh1+6) then
nka(isp) = 0
isp = isp + 1
endif
endif
if(m.le.6.and.ncy.gt.isp) isp = ncy

m10 = m - 1
call findep(dep, m10)
if(dep(m).lt.dep(m10)+0.7) then
dep(m) = dep(m10) + 0.7
endif

if(dep(m).gt.ircl) then
call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
endif
return
end
*******
subroutine UPNKA(k, ig, ir, z, dep, m, jdq,
  lgi, icl, arr, nce, nq, nqd, ncy, zsut,
  iveh1, nka, isp, suts, mprod, jac, tuz,dtuz,
  nibi, nibj)
c updates nka
dimension z(0:100),dep(0:1500), arr(0:1500)
dimension zsut(0:100), tuz(15), dtuz(15)
```

```
dimension nka(100), suts(0:100, 0:60)
ircl = ir+icl*(2-jac)
if(nqd.ge.iveh1+1) go to 100
if(mprod.eq.0) go to 100
if(nq.eq.0.and.arr(m).gt.ig.and.arr(m).le.ircl)
  then
dep(m) = arr(m)
endif
100 intv = int(iveh1*0.3)

if(suts(isp, nka(isp)).lt.arr(m)) then
suts(isp, nka(isp)) = arr(m)
endif
if(suts(isp, nka(isp)).gt.dep(m)) then
suts(isp, nka(isp)) = dep(m)
endif
if(nka(isp).ge.intv.and.arr(m).gt.suts(isp,
  nka(isp)-1))then
suts(isp, nka(isp)) = arr(m)
endif
c
if(m.gt.1.and.isp.le.2) then
if(dep(m-1).ne.0.and.arr(m-1).ne.0.and.dep(m-
  1).eq.arr(m-1).and.dep(m).ne.arr(m)) then
nka(isp) = 0
isp = isp + 1
endif
endif

if(tuz(8).gt.ig) dtuz(8) = tuz(8) - ig
if(tuz(8).gt.ig.and.lgi.ge.40.and.dtuz(8).gt.lg
  i-20.and. nibi.ne.1) then
nibi = 1
endif

if(nibi.ne.1.and.nibj.ne.1) then
if((jdq.eq.7.and.lgi.ge.30).or.(jdq.eq.9.and.lg
  i.ge.50).or.(jdq.eq.11.and.lgi.ge.70)) then
isp = ncy
else
if(nka(isp).ge.iveh1+6) then
nka(isp) = 0
isp = isp + 1
endif
endif
elseif(nibi.eq.1.or.nibj.eq.1) then
if(nka(isp).ge.iveh1+3) then
nka(isp) = 0
isp = isp + 1
endif
endif
if(m.le.6.and.ncy.gt.isp) isp = ncy
if(dep(m).gt.ircl) then
call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
endif
return
end
*******
subroutine DPHDWY(z, zsut)
c stores departure headway and the time queued
  vehicles start moving forward when the
  signal turns green
dimension z(0:100),zsut(0:100)
z(0) = 0.
z(1) = 2.04
z(2) = 4.50
z(3) = 6.62
do 100 n = 4,100
 z(n) = 1.34 + 1.82 * n
100 continue
zsut(0) = 0.
zsut(1) = 2.04
zsut(2) = 2.75
zsut(3) = 3.46
do 200 n = 4,100
 zsut(n) = 1.1 * n
200 continue
return
end
*******
subroutine ARRNT(arr, dep, arrp, depp, m, nce,
  dist, speed, nq, jac, jdum , jvar)
c calculates arrival time of a vehicle when
  there are no turn-in movements
dimension dep(0:1500), depp(0:1500),
  arr(0:1500), arrp(0:1500)
wid = 40.
if(jac.eq.2) wid = 60.
if(arr(m-1).eq.dep(m-1).and.arr(m-1).ne.0.) nce
  = 0

if(m.ge.2) nce = nq

dcs = dist+wid-20.*nce

if(jvar.eq.1) then
call ranspd(jdum, nce, arrp, depp, dcs, speed,
  icx)
else
call cspeed(m, nce, arrp, depp, speed, dcs,
  icx)
endif

arr(m) = depp(m) + dcs/speed

if(20.*nce.gt.dist) arr(m) = depp(m) +
  wid/speed
nce = nce + 1

if(arr(m).le.arr(m-1).and.arr(m).gt.0.) then
arr(m) = arr(m-1) + 1.0
endif

m10 = m - 1
call findep(arr, m10)
if(arr(m).lt.arr(m10)+0.5) then
arr(m) = arr(m10) + 0.5
endif
return
end
*******
subroutine DEPQLS(k, ig, ir, z, dep, m, icl,
  arr, nce, arrd, depd, iq, qs, ki, nq, ncy,
  iveh1, zsut, nka, isp, nua, dtua, suts, iqs,
  qsi, nqs, rqs, niq, dtiq, jdq, nux, dtux,
  icx, jwh, jac, ixt, tuz, dtuz, ijk, jz1,
  jz2, st, st1, jdum, dphy, jvar)
```

```
c calculates vehicle departure times and
  includes an option that delays departure
  when a queue spillback occurs
dimension dep(0:1500), arr(0:1500),
  depd(0:1500), arrd(0:1500)
dimension ki(0:1500), suts(0:100, 0:60),
  nka(100)
dimension z(0:100), zsut(0:100), dphy(0:60)
dimension qs(100), qsi(100), rqs(100)
dimension tuz(15), dtuz(15), st(15), st1(15)
jpd1 = (jwh-10)*(jwh-21)*(jwh-30)*(jwh-
  40)*(jwh-50)
jpd2 = (jwh-11)*(jwh-20)*(jwh-32)*(jwh-42)
if(m.eq.1) then
nua = 0
ki(0) = 0
endif
k = k + 1
call DPHDWY(z, zsut)
if(jvar.eq.1.and.k.eq.1.or.k.eq.2) then
call nordev(jdum, dphy, icx)
endif
jgt = 0
ipo = 9
call NOQUE(arrd, depd, 0, 0, iveh1, ki, m, m1,
  nq, 0, 0, 0, ipo, jgt)
call FINDEV(m, depd, iveh1, lks, st11)
call FINDEX(m, depd, iveh1, lks, st, jz1)
call FINDEX(m, depd, iveh1+1, lks, st1, jz1)

jib = 0
if(jgt.eq.1) go to 290
ircl = ir + icl*(2-jac)
irclp = ir + icl*(2-jac)

if(m.gt.iveh1+1.and.nq.le.iveh1.and.arrd(m-
  1).ne.0..and. st11.gt.arrd(m-1)+4.)then
dep(m) = st11
if(st11.gt.ircl) then
if(jib.eq.0.and.jpd1.eq.0) then
jib = 1
tuz(jz1) = st11
dtuz(jz1) = tuz(jz1) - irclp
endif
igcl = ig + icl
if(dep(m).gt.igcl) then
call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
call UPDAT1(nux, niq, nua)
dtux = st11 - ig - 2.05
nux = 1
ixt = 1
dep(m) = st11
go to 500
endif
call UPDATE(nce, k, ig, ir, z, dep, M, icl,
  ncy)
call UPDAT1(nux, niq, nua)
ixt = 1
go to 500
endif
nua = 1
dtua = st11 - arrd(m-1)
go to 490
endif

IF(arr(m).le.ircl) then
if(nq.ge.iveh1+1) then
if(jvar.eq.0) then
depm = z(k) + ig
else
depm = dphy(k) + ig
endif
dep(m) = st11
iq = iq + 1
qs(iq) = arrd(m1)
if(st11.gt.ircl) then
if(jib.eq.0.and.jpd1.eq.0) then
 jib = 1
 tuz(jz1) = st11
 dtuz(jz1) = tuz(jz1) - irclp
endif
nqs = nqs + 1
rqs(nqs) = arrd(m1)
iq = iq - 1
igcl = ig + icl
if(dep(m).gt.igcl) then
 call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
 call UPDAT1(nux, niq, nua)
dtux = st11 - ig - 2.05
 if(dtux.gt.icl) then
260 call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
 dtux = dtux - icl
 if(dtux.gt.icl) go to 260
 endif
nux = 1
 ixt = 1
dep(m) = st11
go to 500
endif
call UPDATE(nce, k, ig, ir, z, dep, M, icl,
  ncy)
call UPDAT1(nux, niq, nua)
ixt = 1
go to 389
endif
nux = 1
dtux = dep(m) - depm
go to 500
endif
290 if(jvar.eq.0) then
dep(m) = z(k) + ig
else
dep(m) = dphy(k) + ig
endif

if(niq.eq.1) dep(m) = dep(m) + dtiq
if(nux.eq.1) dep(m) = dep(m) + dtux
if(tuz(jz2).gt.ig.and.jpd2.eq.0) then
if(ijk.eq.1.and.ncy.eq.1) go to 389
if(nux.eq.1.and.tuz(jz2).lt.st11)go to 389
dep(m) = dep(m) + dtuz(jz2)
endif
389 continue
ELSEIF(arr(m).gt.ircl) then
if(nq.ge.iveh1+1) then
iq = iq + 1
qs(iq) = arrd(m1)
```

```
if(st11.gt.ircl) then
if(jib.eq.0.and.jpd1.eq.0) then
 jib = 1
 tuz(jz1) = st11
 dtuz(jz1) = tuz(jz1) - irclp
endif
nqs = nqs + 1
rqs(nqs) = arrd(m1)
iq = iq - 1
igcl = ig + icl
if(st11.gt.igcl) then
call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
call UPDAT1(nux, niq, nua)
dtux = st11 - ig - 2.05
if(dtux.gt.icl) then
call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
dtux = dtux - icl
endif
nux = 1
ixt = 1
dep(m) = st11
go to 500
endif
endif
endif
call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
call UPDAT1(nux, niq, nua)
ixt = 1
ENDIF
if(nua.eq.1) dep(m) = dep(m) + dtua
ircl = ir + icl*(2-jac)
irclp = ir + icl*(2-jac)
IF(nka(isp).eq.iveh1+2.and.nq.le.iveh1) then
if(m.ge.iveh1+1.and.arrd(m1).ne.suts(isp,
  nka(isp)-1)) then
if(arrd(m1).lt.suts(isp, nka(isp)-2)) then
if(arrd(m-1).eq.0.and.dep(m-1).ne.0) go to 410
if(suts(isp, nka(isp)-1) - arrd(m1).lt.jdq) go
  to 410
iqs = iqs + 1
qsi(iqs) = arrd(m1)
endif
410 continue
if(dep(m).lt.suts(isp, nka(isp)-1)) then
nua = 1
if(suts(isp, nka(isp)-1).gt.ircl) then
if(jib.eq.0.and.jpd1.eq.0) then
 jib = 1
 tuz(jz1) = st11
 dtuz(jz1) = tuz(jz1) - irclp
endif
if(suts(isp, nka(isp)-1) - arrd(m1).lt.jdq) go
  to 420
nqs = nqs + 1
rqs(nqs) = arrd(m1)
if(iqs.gt.0) iqs = iqs - 1
igcl = ig + icl
if(st11.gt.igcl) then
call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
call UPDAT1(nux, niq, nua)
dtux = st11 - ig - 2.05
nux = 1
ixt = 1
dep(m) = st11
go to 500
endif
420 continue
endif
dtua = suts(isp, nka(isp)-1) - dep(m)
dep(m) = suts(isp, nka(isp)-1)
endif
endif
ENDIF

ircl = ir + icl*(2-jac)
if(k.gt.2.and.arr(m-1).eq.dep(m-1).and.arr(m-
  1).gt.1.and.
  arr(m).gt.ig.and.arr(m).le.ircl) then
dep(m) = arr(m)
endif

490 if(arr(m).ge.dep(m)) dep(m) = arr(m)

m10 = m - 1
call findep(dep, m10)
if(dep(m).lt.dep(m10)+0.7) then
dep(m) = dep(m10) + 0.7
endif

if(dep(m).gt.ircl) then
call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
call UPDAT1(nux, niq, nua)
ixt = 1
endif

igcl = ig + icl
ircl = ir + icl*(2-jac)

if(jgt.eq.1.and.st11.gt.dep(m)) then
if(st11.gt.ircl) then
if(jib.eq.0.and.jpd1.eq.0) then
jib = 1
tuz(jz1) = st11
dtuz(jz1) = tuz(jz1) - ircl
endif
if(dep(m).gt.igcl) then
call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
call UPDAT1(nux, niq, nua)
dtux = st11 - ig - 2.05
nux = 1
ixt = 1
dep(m) = st11
go to 500
endif
call UPDATE(nce, k, ig, ir, z, dep, M, icl,
  ncy)
call UPDAT1(nux, niq, nua)
ixt = 1
dep(m) = st11
go to 500
endif
dep(m) = st11
endif
```

```
if(jgt.eq.1.and.nq.ge.iveh1+1.and.st(jz1).gt.ig
  cl) then
if(jib.eq.0.and.jpd1.eq.0) then
tuz(jz1) = st(jz1)
dtuz(jz1) = tuz(jz1) - ircl
endif
endif

500 continue

return
end
*******
subroutine FINDEV(m, depd, iveh1, lks, st11)
c calculates when the (m-(iveh1+1))th vehicle
  in the downstream link starts moving forward
  (iveh1 = link capacity in number of
  vehicles).
dimension depd(0:1500)
lk1 = 0
if(m.gt.iveh1) then
do 100 lk = 1,99
if(depd(m-lk).eq.0.or.depd(m-
  lk).ne.0.and.depd(m-lk).eq.depd(m-lk-1))
  then
lk1 = lk1 + 1
endif
if(lk-lk1.eq.iveh1+1) then
lks = lk
go to 110
elseif(m-lk.eq.1) then
st11 = 0.
go to 120
endif
100 continue
110 st11 = depd(m-lks) - 2.04 + (iveh1+1)*1.1
endif
120 return
end
*******
subroutine FINDST(m, depd, iveh1, kv, st11)
c calculates when the (m-(kv+1))th vehicle in
  the downstream link starts moving forward
dimension depd(0:1500)
lk1 = 0
if(m.gt.iveh1) then
do 100 lk = 1,99
if(depd(m-lk).eq.0.or.depd(m-
  lk).ne.0.and.depd(m-lk).eq.depd(m-lk-1))
  then
lk1 = lk1 + 1
endif
if(lk-lk1.eq.kv+1) then
lks = lk
go to 110
elseif(m-lk.eq.1) then
st11 = 0.
go to 120
endif
100 continue
110 st11 = depd(m-lks) - 2.04 + (iveh1+1)*1.1
endif
120 return
end
*******
subroutine FINDEX(m, depd, iveh1, lks, st, j1)
c calculates when the (m-iveh1)th vehicle in
  the downstream link starts moving forward
dimension depd(0:1500), st(15)
lk1 = 0
if(m.ge.iveh1) then
do 100 lk = 1,99
if(depd(m-lk).eq.0.or.depd(m-
  lk).ne.0.and.depd(m-lk).eq.depd(m-lk-1))
  then
lk1 = lk1 + 1
endif
if(lk-lk1.eq.iveh1) then
lks = lk
go to 110
elseif(m-lk.eq.1) then
st(j1) = 0.
go to 120
endif
100 continue
110 st(j1) = depd(m-lks) - 2.04 + (iveh1)*1.1
endif
120 return
end
*******
subroutine ARRNTE(arr, dep, arrp, depp, m, nce,
  dist, speed, nq, iveh1, nka, isp, suts, ki,
  jac, jdum, jvar)
c calculates arrival time of a vehicle when
  there are no turn-in movements
dimension dep(0:1500), depp(0:1500),
  arr(0:1500), arrp(0:1500)
dimension ki(0:1500), nka(100), suts(0:100,
  0:60)
wid = 40.
if(jac.eq.2) wid = 60.
if(nce.eq.0) then
ipo = 19
call NOQUE(arr, dep, depp, 0, iveh1, ki, m, 0,
  nq, 0, 0, 0, ipo, 0)
endif
if(arr(m-1).eq.dep(m-1).and.arr(m-1).ne.0.) nce
  = 0

if(m.ge.2) nce = nq

dcs = dist+wid-20.*nce

if(jvar.eq.1) then
call ranspd(jdum, nce, arrp, depp, dcs, speed,
  icx)
else
call cspeed(m, nce, arrp, depp, speed, dcs,
  icx)
endif

arr(m) = depp(m) + dcs/speed

if(20.*nce.gt.dist) arr(m) = depp(m) +
  wid/speed

intv = int(iveh1*0.3)
nisp = nka(isp)
if(nce.ne.0) then
atx = depp(m) + (wid+dist-(nisp-1)*20.)/speed
```

```
if(nisp.le.iveh1.and.atx.lt.suts(isp, nka(isp)-
  1).and.arr(m-1).ne.suts(isp, nka(isp)-1))
  then
arr(m) = atx
endif
endif
if(nisp.eq.iveh1+1.and.depp(m).lt.suts(isp,
  nka(isp)-1). and.nq.ge.intv.and.arr(m-
  1).ne.suts(isp, nka(isp)-1)) then
arr(m) = depp(m) + wid/speed
endif
nce = nce + 1

if(arr(m).le.arr(m-1).and.arr(m).gt.0.) then
arr(m) = arr(m-1) + 1.0
endif
m10 = m - 1
call findep(arr, m10)
if(arr(m).lt.arr(m10)+0.5) then
arr(m) = arr(m10) + 0.5
endif
return
end
*******
subroutine CROSST(kc, ku3, ku4, tdc, carrr,
  arrra, nvol, lgic, noff, icl, jdq, ip, jp,
  speed, ntg, mult, jvol, tuz, dtuz, mq1,
  mq11, klt, krt, jwh1, jwh2, isps, nkas,
  suts, isps1, nkas1, suts1, dist, iveh, ismt,
  irs, igs, irs1, igs1, igf, kfa, ma, st, st1,
  nca, j1, iqis2, qs2, iqss, qsi, nqss, rqs2,
  iqts, qst, iqtis, qsti, nqsts, rqst, ki2,
  ki3, j2, iqis21, qs21, iqss1, qsi1, nqss1,
  rqs21, iqts1, qst1, iqtis1, qsti1, nqsts1,
  rqst1, ki21, ki31, carr1, cdep1, carr2,
  cdep2, carr3, cdep3, darr1, ddep1, darr2,
  ddep2, darr3, ddep3, isp1, nka1, sutc1,
  nqa1, adep, jdum, dy1, dy2, dy3, dy4, dy5,
  dy6, jvar, ispec)
c processes vehicles in the cross street
dimension carr1(0:1500), cdep1(0:1500),
  carr2(0:1500)
dimension cdep2(0:1500), carr3(0:1500),
  cdep3(0:1500)
dimension darr1(0:1500), ddep1(0:1500),
  darr2(0:1500)
dimension ddep2(0:1500), darr3(0:1500),
  ddep3(0:1500)
dimension carrr(0:500), arrra(0:500),
  adep(0:1500)
dimension ki2(0:1500), ki3(0:1500),
  ki21(0:1500), ki31(0:1500)
dimension dist(20), iveh(20), noff(20)
dimension irs(20), igs(20), irs1(20), igs1(20),
  igf(20)
dimension qs2(100), qst(100), qsti(100),
  rqst(100)
dimension qs21(100), qst1(100), qsti1(100),
  rqst1(100)
dimension zz(0:100), zsut(0:100)
dimension suts(0:100, 0:60), suts1(0:100, 0:60)
dimension nkas(100), nkas1(100), qsi(100),
  qsi1(100)
dimension jgn2(100), jgn3(100), rqs2(100),
  rqs21(100)
dimension nka1(100), tuz(15), dtuz(15)
dimension kfa(10), ma(15), st(15), st1(15),
  sutc1(0:100, 0:60)
dimension dy1(0:60), dy2(0:60), dy3(0:60)
dimension dy4(0:60), dy5(0:60), dy6(0:60)
read(50)ixts, ncs, nes, ngs, ks1, ks2, ks3, is,
  nns,nls, iups, iqis3, ncys1, ncys2, ncys3,
  mq2, mq3, iqes,iqeas, niis, niqs, nuas,
  nuxs, ixts1, ncs1, nes1, ngs1, ks11,ks21,
  ks31, is1, nns1, jos1, nrs, iups1, iqis31,
  ncys11, ncys21,ncys31, mq21, mq31, iqes1,
  iqeas1, niis1, niqs1
read(50)nuas1, nuxs1, ijk, ijk1, kfi, nui,
  nui1,imqs1, joc1, nuzi, nuzi1
 read(50)dtiqs, dtuas, dtuxs, cparr1, cpdep1,
  cparr2, cpdep2,dtiqs1, dtuas1, dtuxs1, dtui,
  dtui1, dtuzi, dtuzi1, dparr1,dpdep1, dparr2,
  dpdep2
rewind(50)
jac = 2
if(kfa(j1).eq.0) then
nui = 0
ixts = 0
lric = icl - lgic
if(mult.eq.0) then
write(ku3, 121) kc
121 format(/' ****** CROSS STREET # ', i1, '
  ******'/)
write(ku3, 122)
122 format(' *** Left lane of the cross street
  ***'/)
call HEAD(ku3, icl, lgic, lric, noff(2),
  noff(3))
endif
*****************************************
*****Left lane of the cross street *****
*****************************************
call CTIME(kc, igs, irs, noff, lgic, icl, ip,
  jp)
call INIT(carr1, cdep1, carr2, cdep2, carr3,
  cdep3, ncs, nes, ngs, ks1, ks2, ks3, is,
  nns, m)
call INIT1(jos, nls, iups, iqis2, iqis3, ncys1,
  ncys2, ncys3, mq1, mq2, mq3, iqts, iqes,
  iqeas, iqtis, nqsts)
call INIT2(niis, isps, iqss, nqss, niqs, dtiqs,
  nuas, dtuas, nuxs, dtuxs)
call SIGADJ(irs, igs, lric, lgic, icl, dist,
  speed, jac, kc,carrr, jdum, jvar)
endif

if(kfa(j1).eq.0.or.kfa(j2).eq.0.and.kfi.ne.1)
  then
do 120 ij = 1,100
nkas(ij) = 1
nkas1(ij) = 1
120 continue
ijk = 0
ijk1 = 0
c ijk: 0; ig(2)<igs(2),1; otherwise (ijk1 for
  cross street)

if(igf(kc).gt.igs(2)) then
 ijk = 1
ijk1 = 1
```

```
endif
jgn2(1) = igs(2)
jgn3(1) = igs(3)
do 135 ka = 2,100
jgn2(ka) = jgn2(ka-1) + icl
jgn3(ka) = jgn3(ka-1) + icl
135 continue
call DPHDWY(zz, zsut)
do 145 im = 1,100
do 142 jm = 0,60,1
suts(im, jm) = jgn3(im) + zsut(jm)
suts1(im, jm) = jgn3(im) + zsut(jm)
142 continue
145 continue
if(kfa(j1).eq.0) kfa(j1) = 1
kfi = 1
endif

if(klt.ne.1) go to 1000

C--- Arrival time at intersection 1 ------
C
do 100 m = ma(j1),ma(j1)+ntg
if(igs(2).ge.ismt) jvol = jvol + 1

if(ixts.eq.1) go to 150
ki2(m) = is
ki3(m) = 0

IF(nls.eq.0) then
carr1(m) = tdc * (m-is)
C
C--- Departure time at intersection 1 ----
C
call DEPTQ(ks1, igs(1), irs(1), zz, cdep1, m,
  icl, carr1, ncs,carr2, cdep2, ki2, mq2,
  iveh(1), ncys1, nui, dtui, jac, kc, jdum,
  dy1, jvar)
C
C--- Arrival time at intersection 2 ------
C
call ARRNT(carr2, cdep2, carr1, cdep1, m, ncs,
  dist(1), speed, mq2, jac,jdum, jvar)
C
C--- Departure time at intersection 2 ----
C
jwh = jwh1
call DEPQLS(ks2, igs(2), irs(2), zz, cdep2, m,
  icl, carr2, nes, carr3, cdep3, iqis2, qs2,
  ki3, mq1, ncys2, iveh(1), zsut, nkas, isps,
  nuas, dtuas, suts, iqss, qsi, nqss, rqs2,
  niqs, dtiqs, jdq, nuxs, dtuxs, icx, jwh,
  jac, ixts, tuz,dtuz, ijk, j1, 15, st, st1,
  jdum, dy2, jvar)
if(ixts.eq.1.and.ijk.eq.1) then
ixts = 0
ijk = 0
endif
C
C--- Arrival time at intersection 3 ------
C
if(ixts.eq.1) go to 110
150 if(ixts.eq.1) ixts = 0

ELSEIF(iups.eq.1) then
carr1(m) = cparr1
cdep1(m) = cpdep1
carr2(m) = cparr2
cdep2(m) = cpdep2
nls = 0
iups = 0
iqes = 0
iqeas = 0
ELSE
carr1(m) = cparr1
cdep1(m) = cpdep1
carr2(m) = cparr2
cdep2(m) = cpdep2
ENDIF

icx = 1300
if(kc.eq.2) icw = 11
152 call ARRQLA(carr3, cdep3, carr2, cdep2, m,
  nes, dist(2), speed, at3, carrr, is, nns,
  nls, iups, ki3, mq3, iqts, qst, iqes, iqeas,
  iveh(1), icx, nkas, isps, suts, niis, iqtis,
  qsti, niqs, dtiqs,nqsts, rqst, jdq, arnp,
  icw, jac, jdum, jvar)
icx = 0
icw = 0
C
C--- Departure time at intersection 3 ----
C
call DEPART(ks3, igs(3), irs(3), zz, cdep3, m,
  icl, carr3, ngs,ncys3, jac, 0, jdum, dy3,
  jvar)
mprod = 1
call UPDNKA(ks3, igs(3), irs(3), zz, cdep3, m,
  jdq, lgic, icl, carr3, ngs, mq3, 0, ncys3,
  zsut, iveh(1), nkas, isps, suts,mprod, jac)

155 continue

if(nls.ne.0) then
cparr1 = carr1(m)
cpdep1 = cdep1(m)
cparr2 = carr2(m)
cpdep2 = cdep2(m)
carr1(m) = 0.
cdep1(m) = 0.
carr2(m) = 0.
cdep2(m) = 0.
endif

if(mult.eq.0) then
call FRINT(carr1, cdep1, carr2, cdep2, carr3,
  cdep3, ku3, m, is,jos, mq3)
endif

if(ncys3.lt.isps) then
do 160 ik = isps,isps
nkas(ik) = nkas(ik) + 1
160 continue
else
do 161 ik = isps,ncys3
nkas(ik) = nkas(ik) + 1
161 continue
endif

if(nls.ne.0) then
```

```
carr1(m) = cparr1
cdep1(m) = cpdep1
carr2(m) = cparr2
cdep2(m) = cpdep2
endif

100 continue
110 ma(j1) = m
if(igs(2).ge.ismt) jvol = jvol - 1

1000 if(krt.ne.1) go to 1100

*****************************************
****Right lane of the cross street *****
*****************************************

if(kfa(j2).eq.0) then
ixts1 = 0
joc1 = 0
imqs1 = 0
lric = icl - lgic
if(mult.eq.0) then
write(ku4, 121) kc
write(ku4, 221)
221 format(' *** Right lane of the cross street
  ***'/)
call HEAD(ku4, icl, lgic, lric, noff(2),
  noff(3))
endif
C------
call CTIME(kc, igs1, irs1, noff, lgic, icl, ip,
  jp)
call INIT(darr1, ddep1, darr2, ddep2, darr3,
  ddep3, ncs1, nes1, ngs1, ks11, ks21, ks31,
  is1, nns1, m)
call INIT1(jos1, nrs, iups1, iqis21, iqis31,
  ncys11, ncys21, ncys31, mq11, mq21, mq31,
  iqts1, iqes1, iqeas1,iqtis1, nqsts1)
call INIT2(niis1, isps1, iqss1, nqss1, niqs1,
  dtiqs1, nuas1, dtuas1, nuxs1, dtuxs1)
call SIGADJ(irs1, igs1, lric, lgic, icl, dist,
  speed, jac,kc, carrr, jdum, jvar)

call DPHDWY(zz, zsut)
igsu4 = igs1(2)
jn = 0
nca1 = nca
if(jdq.eq.11) nca1 = 3*nca
kfa(j2) = 1
endif
C
C--- Arrival time at intersection 1 ------
C
do 200 m = ma(j2),ma(j2)+ntg
if(igs1(2).ge.ismt) jvol = jvol + 1
if(m.eq.1) st0 = 0.0

if(ixts1.eq.1) go to 250
ki21(m) = 0
ki31(m) = jos1

darr1(m) = tdc * (m-is1)
C
C--- Departure time at intersection 1 ----
C
c if(kc.eq.1.and.m.ge.18) then
c endif
call DEPTQ(ks11, igs1(1), irs1(1), zz, ddep1,
  m, icl, darr1, ncs1, darr2, ddep2, ki21,
  mq21, iveh(1), ncys11, nui1, dtui1, jac, kc,
  jdum,dy4, jvar)
if(imqs1.eq.1) imqs1 = 0
C
C--- Arrival time at intersection 2 ------
C
call ARRNT(darr2, ddep2, darr1, ddep1, m, ncs1,
  dist(1), speed, mq21,jac, jdum, jvar)
C
C--- Departure time at intersection 2 ----
C
jwh = jwh2
jz2 = 1
if(kc.eq.2) jz2 = 2
if(kc.eq.1) icx = 54
if(kc.eq.3) jz2 = 11
call DEPQLS(ks21, igs1(2), irs1(2), zz, ddep2,
  m, icl, darr2, nes1, darr3, ddep3, iqis21,
  qs21, ki31, mq11, ncys21, iveh(1), zsut,
  nkas1, isps1, nuas1, dtuas1, suts1, iqss1,
  qsi1, nqss1, rqs21, niqs1, dtiqs1, jdq,
  nuxs1, dtuxs1, icx,jwh, jac, ixts1, tuz,
  dtuz, ijk1, j2, jz2, st, st1, jdum, dy5,
  jvar)
if(ixts1.eq.1) joc1 = 0

if(ixts1.eq.1.and.ijk1.eq.1) then
ixts1 = 0
ijk1 = 0
endif
C
C--- Arrival time at intersection 3 ------
C
if(ixts1.eq.1) go to 210
250 if(ixts1.eq.1) then
jz2 = 1
if(kc.eq.2) jz2 = 2
if(kc.eq.3) jz2 = 11
ixts1 = 0

if(kc.ne.3.and.m.ge.25) then
if(tuz(jz2).lt.ddep2(m).and.tuz(jz2).gt.ddep2(m
  )-icl) then
tuz(jz2) = tuz(jz2) + icl
endif
endif

if(tuz(jz2).gt.igs1(2)) then
dtuz(jz2) = tuz(jz2) - igs1(2)
ddep2(m) = ddep2(m) + dtuz(jz2)
if(ddep2(m).ge.irs1(2)) then
240 call UPDATE(nes1, ks21, igs1(2), irs1(2),
  zz, ddep2, m, icl,ncys21)
nkas1(isps1) = 0
isps1 = isps1 + 1
ixts1 = 1
go to 210
endif
endif
endif
```

```
if(ijk1.eq.0.and.((nca.eq.1.and.(ks21.eq.2.or.
  jdq.eq.11.and.(ks21.eq.4.or.ks21.eq.5))).or.
  (nca.eq.2.and.(ks21.eq.2.or.ks21.eq.3.or.jdq
  .eq.11.and.
  (ks21.eq.4.or.ks21.eq.5.or.ks21.eq.6.or.ks21
  .eq.7)))) ) then
jz2 = 1
if(kc.eq.2) jz2 = 2
if(kc.eq.3) jz2 = 11

if(nqa1.ge.iveh(1).and.st1(jz2)-1.1.gt.irs1(2))
  then
call UPDATE(nes1, ks21, igs1(2), irs1(2), zz,
  ddep2, m, icl,ncys21)
go to 252
endif
jos1 = jos1 +1
arrra(jos1) = ddep2(m)

if(kc.ne.3.and.ma(jz2).ge.iveh(1)) then
ijoc = iveh(1)-(joc1+1)
call FINDST(ma(jz2), adep, iveh(1), ijoc, st0)

if(jdq.eq.7.and.(nqa1.eq.iveh(1).or.joc1+nqa1.e
  q.iveh(1)).or.jdq.eq.11.and.st0.gt.ddep2(m))
  then
ssc = st0
ssd = ssc - ddep2(m)

if(ssc.gt.irs1(2)) then
 imqs1 = 1
 if(ssc.gt.igs1(2)+icl) then
 tuz(jz2) = ssc
 endif
elseif(ssd.gt.0) then
 niqs1 = 1
 dtiqs1 = ssd
endif
 endif
endif
joc1 = joc1 + 1
go to 255
endif

if(kc.eq.3) icx = 34
252 call ARRNT(darr3, ddep3, darr2, ddep2, m,
  nes1, dist(2), speed, mq11,jac, jdum, jvar)
 icx = 0
C
C--- Departure time at intersection 3 ----
C

call DEPART(ks31, igs1(3), irs1(3), zz, ddep3,
  m, icl, darr3, ngs1, ncys31, jac, 0, jdum,
  dy6, jvar)
mprod = 1
call UPDNKA(ks31, igs1(3), irs1(3), zz, ddep3,
  m, jdq, lgic, icl, darr3, ngs1, mq11, 0,
  ncys31, zsut, iveh(1), nkas1, isps1,suts1,
  mprod, jac)

255 continue

ki31p = jos1

if(mult.eq.0) then
call FRINT(darr1, ddep1, darr2, ddep2, darr3,
  ddep3, ku4, m,is1, jos1, mq31)
endif

if(imqs1.eq.1) then
call UPDATE(nes1, ks21, igs1(2), irs1(2), zz,
  ddep2, m, icl,ncys21)
niqs1 = 0
ks21 = ks21 - 1
joc1 = 0
endif

if(ki31p.ne.ki31(m)) go to 262
if(ncys31.lt.isps1) then
do 260 ik = isps1,isps1
nkas1(ik) = nkas1(ik) + 1
260 continue
else
do 261 ik = isps1,ncys31
nkas1(ik) = nkas1(ik) + 1
261 continue
endif

262 if(imqs1.eq.1) go to 210

200 continue
210 ma(j2) = m
if(igs1(2).ge.ismt) jvol = jvol - 1
if(imqs1.eq.1) then
ma(j2) = m + 1
if(igs1(2).ge.ismt) jvol = jvol + 1
endif

1100 write(50)ixts, ncs, nes, ngs, ks1, ks2,
  ks3, is, nns, nls, iups, iqis3, ncys1,
  ncys2, ncys3, mq2, mq3, iqes, iqeas, niis,
  niqs, nuas, nuxs, ixts1, ncs1, nes1, ngs1,
  ks11, ks21, ks31, is1, nns1, jos1, nrs,
  iups1, iqis31, ncys11, ncys21, ncys31, mq21,
  mq31, iqes1, iqeas1, niis1, niqs1
write(50)nuas1, nuxs1, ijk, ijk1, kfi, nui,
  nui1, imqs1, joc1, nuzi, nuzi1
write(50)dtiqs, dtuas, dtuxs, cparr1, cpdep1,
  cparr2, cpdep2, dtiqs1, dtuas1, dtuxs1,
  dtui, dtui1, dtuzi, dtuzi1, dparr1,dpdep1,
  dparr2, dpdep2
rewind(50)
return
end
*******
subroutine DEPTQ(k, ig, ir, z, dep, m, icl,
  arr, nce, arrd, depd, ki, nq, iveh1, ncy,
  nui, dtui, jac, kc, jdum, dphy, jvar)
c calculates vehicle departure times and
  includes an option that delays departure
  when a queue spillback occurs
dimension z(0:100), dep(0:1500), arr(0:1500)
dimension zsut(0:100), depd(0:1500),
  arrd(0:1500), ki(0:1500)
dimension dphy(0:60)
if(m.eq.1) then
nq = 0
st11 = 0
endif
```

```
k = k + 1
call DPHDWY(z, zsut)
if(jvar.eq.1.and.k.eq.1.or.k.eq.2) then
call nordev(jdum, dphy, icx)
endif

ircl = ir+icl*(2-jac)
call FINDEV(m, depd, iveh1, lks, st11)
ipo = 9
call NOQUE(arrd, depd, 0, 0, iveh1, ki, m, m1,
  nq, 0, 0, 0, ipo, 0)
if(arr(m).le.ircl) then
if(nq.eq.iveh1+1) then
if(jvar.eq.0) then
depm = z(k) + ig
else
depm = dphy(k) + ig
endif
nui = 1
dtui = st11 - depm
dep(m) = st11
if(st11.gt.ircl) then
igcl = ig + icl
if(dep(m).gt.igcl) then
 call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
dtui = st11 - ig - 2.05
 if(dtui.gt.icl) then
260 call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
 dtui = dtui - icl
dep(m) = st11
 if(dtui.gt.icl) go to 260
 endif
endif
endif
go to 100
endif

if(jvar.eq.0) then
dep(m) = z(k) + ig
else
dep(m) = dphy(k) + ig
endif
if(dep(m).gt.ircl)then
call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
nui = 0
endif
if(arr(m).ge.dep(m)) dep(m) = arr(m)
if(nui.eq.1) dep(m) = dep(m) + dtui
100 continue
elseif(arr(m).gt.ircl) then
call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
nui = 0
if(arr(m).ge.dep(m)) dep(m) = arr(m)
endif

m10 = m - 1
call findep(dep, m10)
if(dep(m).lt.dep(m10)+0.7) then
dep(m) = dep(m10) + 0.7
endif
ircl = ir+icl*(2-jac)

if(k.gt.2.and.arr(m-1).eq.dep(m-1).and.
  arr(m).gt.ig.and.arr(m).le.ircl) then
dep(m) = arr(m)
endif

if(dep(m).gt.ircl) then
call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
nui = 0
endif
return
end
*******
subroutine PRN(arr1, dep1, arr2, dep2, arr3,
  dep3, ku, m, j, jo, nq2, nq3, nq9)
c prints arrival and departure times at each
  intersection
dimension arr1(0:1500), dep1(0:1500),
  arr2(0:1500)
dimension dep2(0:1500), arr3(0:1500),
  dep3(0:1500)
write(ku, 100) m, (m-j), j, jo, arr1(m),
  dep1(m), nq2, arr2(m), dep2(m), nq3, nq9,
  arr3(m), dep3(m)
100 format( 4I4,2(1X, 2F8.1, I3), 1X, I3,2F7.1)
return
end
*******
subroutine PRNT(arr1, dep1, arr2, dep2, arr3,
  dep3, ku, m, j,jo, nq9)
c prints arrival and departure times at each
  intersection
dimension arr1(0:1500), dep1(0:1500),
  arr2(0:1500)
dimension dep2(0:1500), arr3(0:1500),
  dep3(0:1500)
write(ku, 100) m, (m-j), j, jo, arr1(m),
  dep1(m), arr2(m), dep2(m), nq9, arr3(m),
  dep3(m)
100 format(4I4, 2(1X, 2F8.1), I3, 1X, 2F8.1)
return
end
*******
subroutine HEAD(j, icl, lgi, lri, noff2, noff3)
write(j, 100) icl, lgi, lri, noff2, noff3
100 format(3x, 'C=', i3, 'sec, ', ' g=', i2,
  'sec, ', ' r=', i2, 'sec, ', ' off2=', i2,
  'sec, ', ' off3=',i2, 'sec'/)
write(j, 200)
200 format(' Σ thruturn*** Int #1 ***','*** Int
  #2 ***', '*** Int #3 ***'/
  'inoutA.timeD.time',' A.time D.time',','
  A.time D.time'/)
 return
end
*******
subroutine INPUT(icl, dist, iveh, speed, ni)
c stores data in appropriate format and
  locations for later use by other subroutines
dimension dist(20), iveh(20)
iveh(1) = int(dist(1)/20)
iveh(2) = int(dist(2)/20)
speed = 18.1
ni = 3
```

```
return
end
*******
subroutine INIT(arr1, dep1, arr2, dep2, arr3,
  dep3, nc, ne, ng, k1, k2, k3, i, n, m)
c initialize computational arrays
dimension arr1(0:1500), dep1(0:1500),
  arr2(0:1500)
dimension dep2(0:1500), arr3(0:1500),
  dep3(0:1500)
nc = 0
ne = 0
ng = 0
k1 = 0
k2 = 0
k3 = 0
i = 0
n = 1
do 100 m = 0, 1500
 arr1(m) = 0.
 dep1(m) = 0.
 arr2(m) = 0.
 dep2(m) = 0.
 arr3(m) = 0.
100 dep3(m) = 0.
return
end
*******
subroutine INIT1(jo, nnn, iup, iqi2, iqi3,
  ncy1, ncy2, ncy3, nq2, nq3, nq9, iqt, iqe,
  iqea, iqti, nqst)
c initialize computational arrays
jo = 0
nnn = 0
iup = 0
iqi2 = 0
iqi3 = 0
ncy1 = 1
ncy2 = 1
ncy3 = 1
nq2 = 0
nq3 = 0
nq9 = 0
iqt = 0
iqe = 0
iqea = 0
iqti = 0
nqst = 0
return
end
*******
subroutine INIT2(nii, isp, iqs, nqs, niq, dtiq,
  nua, dtua, nux,dtux)
c initialize computational arrays
nii = 0
isp = 1
iqs = 0
nqs = 0
niq = 0
dtiq = 0.
nua = 0
dtua = 0.
nux = 0
dtux = 0.
return
end
*******
subroutine SIGNAL(j1, speed, icl, ni, ir, ig,
  noff, lgi)
dimension ir(20), ig(20), noff(20)
c calculates arterial traffic signal timing
ir(1) = 0
ig(1) = icl - lgi
ir1 = ir(1)
ig1 = ig(1)
do 100 n = 1,ni-1
ir(n+1) = ir(1) + noff(n+1)
ig(n+1) = ig(1) + noff(n+1)
100 continue
if(j1.eq.3.or.j1.eq.4) then
ir(1) = ir(ni)
ig(1) = ig(ni)
if(ir(1).gt.0) then
ig(1) = ig(1) - icl
ir(1) = ir(1) - icl
endif
ir(2) = ir(2)
ig(2) = ig(2)
ir(3) = ir1
ig(3) = ig1
endif
return
end
*******
subroutine SIGNA1(jc, ni, ir, ig, ir1, ig1)
dimension ir(20), ig(20), ir1(20), ig1(20)
if(jc.eq.1) then
do 100 n = 1,ni
ir1(n) = ir(n)
ig1(n) = ig(n)
100 continue
else
do 200 n = 1,ni
ir1(n) = ir(4-n)
ig1(n) = ig(4-n)
200 continue
endif
return
end
*******
subroutine CTIME(kc, ig, ir, noff, lgi, icl,
  ip, jp)
c calculates cross street traffic signal timing
dimension ir(20), ig(20), noff(20)
ig(2) = noff(kc)
if(ig(2).ge.icl) ig(2) = ig(2) - icl
ir(2) = ig(2) + lgi
ig(1) = ig(2) - ip
ir(1) = ir(2) - ip
ig(3) = ig(1) + jp
ir(3) = ir(1) + jp
return
end
*******
subroutine SIGADJ(ir, ig, lri, lgi, icl, dist,
  speed, jac, kc,carrr, jdum, jvar)
c adjusts traffic signal timing obtained by
  SIGNAL
dimension ir(20), ig(20), dist(20),
  carrr(0:500)
```

```
wid = 40.
if(jac.eq.2) wid = 60.

dcs = dist(1) + wid
nce = 0
speed = -6.1525 + 15.268*alog10(dcs)

if(ig(1)+lgi.lt.0.or.jac.eq.2.and.ir(1).le.7.2)
  then
ir(1) = ir(1) + icl
ig(1) = ig(1) + icl
endif
a1 = 7.2
if(jac.eq.1) a1 = lri + 2.04
if(kc.eq.1) then
a1 = ig(1) + 2.04
if(a1.lt.7.2) a1 = 7.2
endif
arr2f = a1 + (dist(1)+wid)/speed
ircl2 = ir(2) - icl
if(jac.eq.1) ircl2 = ir(2)
if(kc.eq.1) ircl2 = ig(2)

if(arr2f.gt.ig(2)+lgi) then
ir(2) = ir(2) + icl
ig(2) = ig(2) + icl
elseif(arr2f.lt.ircl2) then
ir(2) = ir(2) - icl
ig(2) = ig(2) - icl
endif

if(arr2f.lt.ig(2)) dep2f = ig(2) + 2.04
if(arr2f.ge.ig(2)) dep2f = arr2f
arr3f = dep2f + (dist(2)+wid)/speed
ircl3 = ir(3) - icl
if(jac.eq.1) ircl3 = ir(3)

if(arr3f.gt.ig(3)+lgi) then
ig(3) = ig(3) + icl
ir(3) = ir(3) + icl
elseif(arr3f.lt.ircl3) then
ig(3) = ig(3) - icl
ir(3) = ir(3) - icl
endif
if(jac.eq.2.and.ig(3).gt.carrr(1)+icl) then
ig(3) = ig(3) - icl
ir(3) = ir(3) - icl
endif
return
end
*******
subroutine UPDATE(nce, k, ig, ir, z, dep, m,
  icl, ncy)
c updates signal timing
dimension z(0:100),dep(0:1500),zsut(0:100)
nce = 0
k = 1
ig = ig + icl
ir = ir + icl
call DPHDWY(z, zsut)
dep(m) = z(k) + ig
ncy = ncy + 1
return
end
*******
subroutine UPDAT1(nux, niq, nua)
nux = 0
niq = 0
nua = 0
return
end
*******
subroutine PRNQS(ku, nint, icl, lgi, jiq, qs,
  jncy, jiqs, qsi, jnrqs, rqs, prqs, pqs,
  pqsn, ptqs, ismt, smtj, mult)
c calculates and prints number of queue
  spillbacks caused by through traffic
dimension qs(100), qsi(100), rqs(100)
if(mult.eq.0) then
smpd = smtj - ismt
write(ku, 111) nint
111 format(/' **** QS due to thru traffic at
  Int.#', i2,' ****'/)
write(ku, 113) ismt
113 format('Simulation starts at t =', i4, '
  sec')
write(ku, 115) smtj
115 format('Simulation ends at t =', f6.0, '
  sec'/)
write(ku, 117) jncy
117 format(2x, 'No. of cycle:', i3/)

do 100 iq = 1, jiq
write(ku, 110) iq, qs(iq)
110 format(i8, 8x, f8.1)
100 continue
if(jiq.eq.0) then
write(ku, 120) jiq
120 format(2x, 'No. of QSi-full:', i3/)
else
call CNNQS(ismt, smtj, qs, jiq)
write(ku, 120) jiq
endif

do 200 jq = 1, jiqs
write(ku, 210) jq, qsi(jq)
210 format(i8, 8x, f8.1)
200 continue
if(jiqs.eq.0) then
write(ku, 220) jiqs
220 format(2x, 'No. of QSi-n.full:',i3/)
else
call CNNQS(ismt, smtj, qsi, jiqs)
write(ku, 220) jiqs
endif
c
do 300 ia = 1, jnrqs
write(ku, 310) ia, rqs(ia)
310 format(i8, 8x, f8.1)
300 continue
if(jnrqs.eq.0) then
write(ku, 320) jnrqs
320 format(2x, 'No. of QS-full/n.full:', i3/)
else
call CNNQS(ismt, smtj, rqs, jnrqs)
write(ku, 320) jnrqs
endif
endif
```

```
if(mult.eq.1) then
if(jiq.ne.0) then
call CNNQS(ismt, smtj, qs, jiq)
endif
if(jiqs.ne.0) then
call CNNQS(ismt, smtj, qsi, jiqs)
endif
if(jnrqs.ne.0) then
call CNNQS(ismt, smtj, rqs, jnrqs)
endif
endif

bncy = jncy
prqs = jnrqs/bncy
pqs = jiq/bncy
pqsn = jiqs/bncy
jtotal = jiq + jiqs + jnrqs
ptqs = jtotal/bncy

if(mult.eq.0) then
write(ku, 340) prqs
340 format('No.of QS-f/n.f per cycle:', f5.2, '
  / cycle'/)
write(ku, 360) jtotal
360 format(2x, 'No. of QS(i)-total:', i3/)
write(ku, 380) ptqs
380 format('No.of QS(i)-total per cycle:',
  f5.2, ' / cycle'/)
write(ku, 460) smpd
460 format(2x, 'Simulation Period:', f6.0, '
  sec'/)
endif
return
end
*******
subroutine CNNQS(ismt, smtj, wqs, jnnqs)
c counts numbers of queue spillbacks
dimension wqs(100)
nnqs = 0
do 100 ii = 1,jnnqs
if(wqs(ii).gt.ismt.and.wqs(ii).lt.smtj) then
nnqs = nnqs + 1
endif
100 continue
jnnqs = nnqs
return
end
*******
subroutine PRNQST(ku, nint, jiqt, qst, jiqti,
  qsti, jncy, jnrqst, rqst, prqst, pqst,
  pqstn, ptqst, ismt, smtj, mult)
c calculates and prints number of queue
  spillbacks caused by turning-in vehicles
dimension qst(100), qsti(100), rqst(100)
if(mult.eq.0) then
write(ku, 111) nint
111 format(/' **** QS due to turn-in traffic at
  Int. #', i2,' ****'/)
write(ku, 117) jncy
117 format(2x, 'No. of cycle:', i3/)

do 100 iq = 1, jiqt
write(ku, 110) iq, qst(iq)
110 format(i8, 8x, f8.1)
100 continue

if(jiqt.eq.0) then
write(ku, 120) jiqt
120 format(2x,'No. of QSi-full:', i3/)
else
call CNNQS(ismt, smtj, qst, jiqt)
write(ku, 120) jiqt
endif

do 200 ia = 1, jiqti
write(ku, 210) ia, qsti(ia)
210 format(i8, 8x, f8.1)
200 continue
if(jiqti.eq.0) then
write(ku, 220) jiqti
220 format(2x, 'No. of QSi-n.full:', i3/)
else
call CNNQS(ismt, smtj, qsti, jiqti)
write(ku, 220) jiqti
endif

do 300 iat = 1, jnrqst
write(ku, 310) iat, rqst(iat)
310 format(i8, 8x, f8.1)
300 continue
if(jnrqst.eq.0) then
write(ku, 320) jnrqst
320 format(2x, 'No. of QS-full/n.full:', i3/)
else
call CNNQS(ismt, smtj, rqst, jnrqst)
write(ku, 320) jnrqst
endif
endif

if(mult.eq.1) then
if(jiqt.ne.0) then
call CNNQS(ismt, smtj, qst, jiqt)
endif
if(jiqti.ne.0) then
call CNNQS(ismt, smtj, qsti, jiqti)
endif
if(jnrqst.ne.0) then
call CNNQS(ismt, smtj, rqst, jnrqst)
endif
endif

bncy = jncy
prqst = jnrqst/bncy
pqst = jiqt/bncy
pqstn = jiqti/bncy
jtotal = jiqt + jiqti + jnrqst
ptqst = jtotal/bncy

if(mult.eq.0) then
write(ku, 340) prqst
340 format('No.of QS-f/n.f per cycle:', f5.2, '
  / cycle'/)
write(ku, 360) jtotal
360 format(2x, 'No. of QS(i)-total:', i3/)
write(ku, 380) ptqst
380 format('No.of QS(i)-total per cycle:',
  f5.2, ' / cycle'/)
endif
return
end
*******
```

```
subroutine PRNOP1(ku, jc, ln, incre, ioff1,
  ioff2, joff1, joff2, pt)
c prints summarized outputs (number of queue
  spillbacks per cycle)
dimension pt(0:15, 0:15)
if(jc.eq.1) then
write(ku, 10) ln
10 format(/4X, 'arterial lane no.', i1/)
elseif(jc.eq.2) then
write(ku, 20) ln
20 format(/4X, 'cross street no.', i1/)
elseif(jc.eq.3) then
write(ku, 30)
30 format(/4X, 'arterial total'/)
elseif(jc.eq.4) then
write(ku, 40)
40 format(/4X, 'cross street total'/)
elseif(jc.eq.5) then
write(ku, 50)
50 format(/4X, 'No.of QS per cycle: Total'/)
elseif(jc.eq.6) then
write(ku, 60)
60 format(/4X, 'Simulation Period: Total'/)
endif
write(ku, 200) (jp*incre,jp = joff1, joff2)
200 format(4x, 'off3:', i2, 14(3x, i3))
write(ku, 300)
300 format(2x, 'off2')
do 500 ip = ioff1,ioff2
if(jc.ne.6) then
write(ku, 400) ip*incre, (pt(ip, jp),jp=joff1,
  joff2)
400 format(3x, i3, 15f6.2)
else
write(ku, 410) ip*incre, (pt(ip, jp),jp=joff1,
  joff2)
410 format(3x, i3, 15f6.0)
endif
500 continue
return
end
*******
subroutine PRNOP2(ku, ln, incre, ioff1, ioff2,
  joff1, joff2, mt)
c prints summarized outputs (number of vehicles
  simulated)
dimension mt(0:15, 0:15)
if(ln.eq.99) then
write(ku, 100)
100 format(/4X, 'No. of vehicles (warmup)'/)
go to 15
endif
write(ku, 10) ln
10 format(/4X, 'lane no.'i2/)
15 write(ku, 20) (jp*incre,jp = joff1, joff2)
20 format(4x, 'off3:', i2, 14(3x, i3))
write(ku, 30)
30 format(2x, 'off2')
do 50 ip = ioff1,ioff2
write(ku, 40) ip*incre, (mt(ip, jp),jp = joff1,
  joff2)
40 format(3x, i3, 15i6)
50 continue
return
end
*******
subroutine cspeed(m, nce, arrp, depp, speed,
  dcs, icx)
c calculates average overall speeds
dimension depp(0:1500), arrp(0:1500)
if(nce.gt.0) then
speed = 13.033 + 0.026584*dcs
elseif(arrp(m).ne.depp(m)) then
speed = -6.1525 + 15.268*alog10(dcs)
else
speed = 42.77
endif
return
end
*******
subroutine ranspd(jdum, nce, arrp, depp, dcs,
  speed, icx)
c generates random average overall speeds
dimension rn(100), vnor(100), depp(0:1500),
  arrp(0:1500)
idum = jdum - 100
jdum = jdum - 100
if(nce.gt.0) then
s = 2.85
elseif(arrp(m).ne.depp(m)) then
s = 3.28
else
xbar = 42.77
s = 6.31
endif
ni = 1
nj = 1

call rannum (idum, rn, ni)
call normal (rn, vnor, nj, icx)

if(nce.gt.0) then
speed = s*vnor(1) + 13.033 + 0.026584*dcs
elseif(arrp(m).ne.depp(m)) then
speed = s*vnor(1) - 6.1525 + 15.268*alog10(dcs)
else
speed = s*vnor(1) + xbar
endif

return
end
*******
subroutine nordev(jdum, dphy, icx)
c generates normal random deviates with a mean
  m and a standard deviation s
dimension rn(100), vnor(100), hdwy(100),
  dphy(0:60)
idum = jdum - 100
jdum = jdum - 100
s = 0.43
ni = 100
nj = 60
dphy(0) = 0.
call rannum (idum, rn, ni)
call normal (rn, vnor, nj, icx)
do 11 j = 1,nj
if(j.eq.1) xbar = 2.04
if(j.eq.2) xbar = 2.46
if(j.eq.3) xbar = 2.12
if(j.ge.4) xbar = 1.82
```

```
hdwy(j) = s*vnor(j) + xbar
dphy(j) = dphy(j-1) + hdwy(j)
11 continue
return
end
*******
subroutine rannum(idum, rn, ni)
c generates uniform random deviates between 0.0
  and 1.0
dimension r(97),rn(100)
parameter
  (m1=259200,ia1=7141,ic1=54773,rm1=1./m1)
parameter
  (m2=134456,ia2=8121,ic2=28411,rm2=1./m2)
parameter (m3=243000,ia3=4561,ic3=51349)
data iff /0/
do 10 i = 1,ni
if (idum.lt.0.or.iff.eq.0) then
iff = 1
ix1 = mod(ic1-idum, m1)
ix1 = mod(ia1*ix1+ic1, m1)
ix2 = mod(ix1, m2)
ix1 = mod(ia1*ix1+ic1, m1)
ix3 = mod(ix1, m3)
do 11 j = 1,97
ix1 = mod(ia1*ix1+ic1, m1)
ix2 = mod(ia2*ix2+ic2, m2)
r(j) = (float(ix1)+float(ix2)*rm2)*rm1
11 continue
idum = 1
endif
ix1 = mod(ia1*ix1+ic1, m1)
ix2 = mod(ia2*ix2+ic2, m2)
ix3 = mod(ia3*ix3+ic3, m3)
j = 1 + (97*ix3)/m3
if (j.gt.97.or.j.lt.1) pause 'stop'
rn(i) = r(j)
r(j) = (float(ix1)+float(ix2)*rm2)*rm1
10 continue
return
end
*******
subroutine normal (rn, vnor, nj, icx)
c generates normally distributed deviates with
  zero mean and unit variance
dimension rn(100),vnor(100)
data iset/0/
i = 0
iov = 0

do 50 k = 1,nj
vnor(k) = 0.0
50 continue

do 100 j = 1,nj
if (iset.eq.0) then
i = i + 2
10v1 = 2.*rn(i-1) - 1.
v2 = 2.*rn(i) - 1.
r = v1**2 + v2**2
if (r.ge.1) then
i = i + 2
if(i.gt.100) then
i = 0
iov = 1
go to 100
endif
go to 10
endif
fac = sqrt(-2.*log(r)/r)
gset = v1*fac
gasdev = v2*fac
jj = j
if(iov.eq.1) jj = jj - 1
vnor(jj) = gasdev
iset = 1
else
gasdev = gset
jj = j
if(iov.eq.1) jj = jj - 1
vnor(jj) = gasdev
iset = 0
endif
100 continue
return
end
```

Appendix B

Source Code (Two-way Operation Version):

Traffic Simulation Model For Oversaturated Arterial Networks

```
program
c traffic simulation model for oversaturated
  arterial networks
c two-way arterial operation version
dimension arr1(0:2000), dep1(0:2000),
  arr2(0:2000)
dimension dep2(0:2000), arr3(0:2000),
  dep3(0:2000)
dimension arr11(0:2000), dep11(0:2000),
  arr21(0:2000)
dimension dep21(0:2000), arr31(0:2000),
  dep31(0:2000)
dimension zz(0:100), zsut(0:100), igf(20)
dimension arrr2(0:500), arrl3(0:500),
  arrr4(0:500)
dimension carrl1(0:500), carrr2(0:500),
  carrl3(0:500)
dimension ki2(0:2000), ki3(0:2000),
  ki21(0:2000), ki31(0:2000)
dimension dist(20), iveh(20), ir(20), ig(20),
  ir1(20), ig1(20)
dimension pta1(0:15, 0:15), pta2(0:15, 0:15),
  pta3(0:15, 0:15)
dimension pta4(0:15, 0:15), ptc1(0:15, 0:15),
  ptc2(0:15, 0:15)
dimension ptc3(0:15, 0:15), ptat(0:15, 0:15),
  ptct(0:15, 0:15)
dimension ptst(0:15, 0:15), spst(0:15, 0:15)
dimension mar(0:15, 0:15), mal(0:15, 0:15),
  mwu(0:15, 0:15)
dimension marb(0:15, 0:15), malb(0:15, 0:15)
dimension qs2(80), qs20(80), qs21(80), qs31(80)
dimension qst(80), qst1(80), qsti(80),
  qsti1(80)
dimension noff(20), jgn1(100), jgn2(100),
  jgn3(100)
dimension nka(100), nka0(100), nka1(100),
  nka2(100)
dimension qsi(80), qsi0(80), qsi1(80), qsi2(80)
dimension suts(0:100, 0:60), suts0(0:100, 0:60)
dimension suts1(0:100, 0:60), suts2(0:100,
  0:60)
dimension rqst(80), rqst1(80), dep(10)
dimension rqs2(80), rqs20(80), rqs21(80),
  rqs31(80)
dimension tuz(15), dtuz(15), wuz(15, 80),
  ib(15)
dimension kfa(10), ma(10), st(15)

dimension arrb1(0:2000), depb1(0:2000),
  arrb2(0:2000)
dimension depb2(0:2000), arrb3(0:2000),
  depb3(0:2000)
dimension arrb11(0:2000), depb11(0:2000),
  arrb21(0:2000)
dimension depb21(0:2000), arrb31(0:2000),
  depb31(0:2000)
dimension arrrb2(0:500), arrlb3(0:500),
  arrrb4(0:500)
dimension darrl1(0:500), darrr2(0:500),
  darrl3(0:500)
dimension kib2(0:2000), kib3(0:2000),
  kib21(0:2000)
dimension irb(20), igb(20), irb1(20), igb1(20),
  kib31(0:2000)
dimension qsb2(80), qsb20(80), qsb21(80),
  qsb31(80)
dimension qstb(80), qstb1(80), qstib(80),
  qstib1(80)
dimension nkab(100), nkab0(100), nkab1(100),
  nkab2(100)
dimension qsib(80), qsib0(80), qsib1(80),
  qsib2(80)
dimension sutsb(0:100, 0:60), sutsb0(0:100,
  0:60)
dimension sutsb1(0:100, 0:60), sutsb2(0:100,
  0:60)
dimension rqstb(80), rqstb1(80)
dimension rqsb2(80), rqsb20(80), rqsb21(80),
  rqsb31(80)

dimension mcl1(0:15, 0:15), mcr1(0:15, 0:15)
dimension carr1(0:2000), cdep1(0:2000),
  carr2(0:2000)
dimension cdep2(0:2000), carr3(0:2000),
  cdep3(0:2000)
dimension darr1(0:2000), ddep1(0:2000),
  darr2(0:2000)
dimension ddep2(0:2000), darr3(0:2000),
  ddep3(0:2000)
dimension kir2(0:2000), kir3(0:2000),
  kir21(0:2000)
dimension irr(20), igr(20), irr1(20), igr1(20),
  kir31(0:2000)
dimension nkar(100), nkar1(100)
dimension sutsr(0:100, 0:60), sutsr1(0:100,
  0:60)
dimension qsr2(80), qsr21(80), qstr(80),
  qstr1(80)
dimension qsir(80), qsir1(80), qstir(80),
  qstir1(80)
dimension rqsr2(80), rqsr21(80), rqstr(80),
  rqstr1(80)

dimension mcl2(0:15, 0:15), mcr2(0:15, 0:15)
dimension earr1(0:2000), edep1(0:2000),
  earr2(0:2000)
dimension edep2(0:2000), earr3(0:2000),
  edep3(0:2000)
dimension farr1(0:2000), fdep1(0:2000),
  farr2(0:2000)
dimension fdep2(0:2000), farr3(0:2000),
  fdep3(0:2000)
dimension kis2(0:2000), kis3(0:2000),
  kis21(0:2000)
dimension irs(20), igs(20), irs1(20), igs1(20),
  kis31(0:2000)
dimension nkas(100), nkas1(100)
dimension sutss(0:100, 0:60), sutss1(0:100,
  0:60)
dimension qss2(80), qss21(80), qsts(80),
  qsts1(80)
dimension qsis(80), qsis1(80), qstis(80),
  qstis1(80)
dimension rqss2(80), rqss21(80), rqsts(80),
  rqsts1(80)

dimension mcl3(0:15, 0:15), mcr3(0:15, 0:15)
dimension garr1(0:2000), gdep1(0:2000),
  garr2(0:2000)
dimension gdep2(0:2000), garr3(0:2000),
  gdep3(0:2000)
dimension harr1(0:2000), hdep1(0:2000),
  harr2(0:2000)
```

```
dimension hdep2(0:2000), harr3(0:2000),
  hdep3(0:2000)
dimension kit2(0:2000), kit3(0:2000),
  kit21(0:2000)
dimension irt(20), igt(20), irt1(20), igt1(20),
  kit31(0:2000)
dimension nkat(100), nkat1(100)
dimension sutst(0:100, 0:60), sutst1(0:100,
  0:60)
dimension qst2(80), qst21(80), qstt(80),
  qstt1(80)
dimension qsit(80), qsit1(80), qstit(80),
  qstit1(80)
dimension rqst2(80), rqst21(80), rqstt(80),
  rqstt1(80)
dimension mj(100), mj1(100), mj2(100), bj(30),
  bj1(30), bj2(30)
dimension dy1(0:60), dy2(0:60), dy3(0:60)
dimension dy4(0:60), dy5(0:60), dy6(0:60)
dimension dya1(0:60), dya2(0:60), dya3(0:60)
dimension dya4(0:60), dya5(0:60), dya6(0:60)
dimension dy11(0:60), dy12(0:60), dy13(0:60)
dimension dy14(0:60), dy15(0:60), dy16(0:60)
dimension dy21(0:60), dy22(0:60), dy23(0:60)
dimension dy24(0:60), dy25(0:60), dy26(0:60)
dimension dy31(0:60), dy32(0:60), dy33(0:60)
dimension dy34(0:60), dy35(0:60), dy36(0:60)
C-----------------------------------------
open(4, file='inputo', status='old',
  form='formatted')
read(4, *) mult, jcyc, jvar
read(4, *) nvol, ncwu, tdc, nac, nca
read(4, *) dist(1), dist(2), wida, widc, avsh
read(4, *) icl, lgi, idxn, incg, ipt, jdq
read(4, *) noff(1), ioff1, ioff2, joff1, joff2,
  incre

c input variables
c
c mult: number of cases (0; single case,  1;
  multiple cases for optimization)
c jcyc: cycle length optimization (0; no,  1;
  yes)
c jvar: variability of parameters (0; no,  1;
  yes)
c nvol: number of vehicles to be simulated
c ncwu: number of signal cycles for warm-up
  time
c tdc: cross street traffic demand (arriving
  headway)
c nac: number of turning vehicles per cycle
  from arterial to cross street(CS)
c nca: number of turning vehicles per cycle
  from CS to arterial
c dist(1): link length between the 1st and 2nd
  intersections [ft]
c dist(2): link length between the 2nd and 3rd
  intersections [ft]
c wida: width of arterial [ft]
c widc: width of cross streets [ft]
c avsh: average vehicle space headway [ft]
c icl: cycle length [sec]
c lgi: arterial green interval [sec]
c idxn: difference between maximum and minimum
  arterial greens [sec]
c incg: increment of idxn (for optimization)
  [sec]
c ipt: protected left turn phase duration (only
  for two-way operation) [sec]
c jdq: minimum delay to determine a queue
  spillback [sec]
c noff(1): offset 1 [sec]
c ioff1: minimum offset 2 (0,  for
  optimization)
c ioff2: minimum offset 2 (C,  for
  optimization)
c joff1: minimum offset 3 (0,  for
  optimization)
c joff2: minimum offset 3 (C,  for
  optimization)
c incre: increment of offsets (for
  optimization)

open(unit=10, file='y1d', status='unknown',
  form='formatted')
open(unit=11, file='y2i', status='unknown',
  form='formatted')
open(unit=12, file='y3', status='unknown',
  form='formatted')
open(unit=13, file='y4', status='unknown',
  form='formatted')
open(unit=20, file='y5p', status='unknown',
  form='formatted')
open(unit=21, file='y6j', status='unknown',
  form='formatted')
open(unit=30, file='y7', status='unknown',
  form='formatted')
open(unit=31, file='y8', status='unknown',
  form='formatted')
open(unit=40, file='y9', status='unknown',
  form='formatted')
open(unit=41, file='y10', status='unknown',
  form='formatted')
open(unit=50, file='sc', status='unknown',
  form='unformatted')
ku1 = 10
ku2 = 11
ku3 = 12
ku4 = 13
ku5 = 20
ku6 = 21
ku7 = 30
ku8 = 31
ku9 = 40
ku10 = 41
ku15 = 50
C------
lri = icl - lgi

noff(1) = 0
icl = lgi + lri
nac1 = nac
ncas = nca
if(jdq.eq.11) then
  nac1 = 3*nac
  ncas = 3*nca
endif
if(mult.eq.1) then
  write(ku1, 120) icl, lgi-ipt, ipt, lri,
  dist(1)
  write(ku2, 120) icl, lgi-ipt, ipt, lri,
  dist(1)
  write(ku5, 120) icl, lgi-ipt, ipt, lri,
  dist(1)
```

```
  write(ku6, 120) icl, lgi-ipt, ipt, lri,
  dist(1)
120  format(/2x, 'C=', i3, 'sec.', ' g=', i2, '
  l=', i2, ' r=', i2, ',  link length =',
  f5.0, 'ft.')
  write(ku1, 125) nac1, ncas
  write(ku2, 125) nac1, ncas
  write(ku5, 125) nac1, ncas
  write(ku6, 125) nac1, ncas
125  format(2x, 'turn-in: ', '(art to cro) =',
  i2, ' veh/cycle, ', ' (cro to art) =', i2, '
  veh/cycle')
endif

do 400 ip = ioff1,  ioff2
do 410 jp = joff1,  joff2
do 420 kp = lgi,  lgi+idxn,  incg

noff(2) = ip*incre
noff(3) = jp*incre
if(noff(2).ge.icl) noff(2) = noff(2) - icl
if(noff(3).ge.icl) noff(3) = noff(3) - icl
ipz = noff(2)
jpz = noff(3)

lgi = kp
if(jcyc.eq.1) then
  lri = kp/incg
  icl = lgi + lri
else
  lri = icl - kp
endif

if(mult.eq.0) then
  lgit = lgi - ipt
  call HEAD(ku1, icl, lgit, ipt, lri, noff(2),
  noff(3))
  call HEAD(ku2, icl, lgit, ipt, lri, noff(2),
  noff(3))
  call HEAD(ku3, icl, lgit, ipt, lri, noff(2),
  noff(3))
  call HEAD(ku4, icl, lgit, ipt, lri, noff(2),
  noff(3))
endif
do 130 ij = 1,  100
  nka(ij) = 1
  nka0(ij) = 1
  nka1(ij) = 1
  nka2(ij) = 1
  nkab(ij) = 1
  nkab0(ij) = 1
  nkab1(ij) = 1
  nkab2(ij) = 1
130 continue
do 140 km = 0,  500
  carrl1(km) = 0.
  carrr2(km) = 0.
  carrl3(km) = 0.
  arrr2(km) = 0.
  arrl3(km) = 0.
  arrr4(km) = 0.
  darrl1(km) = 0.
  darrr2(km) = 0.
  darrl3(km) = 0.
  arrrb2(km) = 0.
  arrlb3(km) = 0.
  arrrb4(km) = 0.
140 continue

***** Arterial,  A direction  *****************
***** Right lane

call INPUT(icl, dist, iveh, speed, ni)
ixt11 = 0
ixt12 = 0
ixt13 = 0
ixt21 = 0
ixt22 = 0
ixt23 = 0
ixt31 = 0
ixt32 = 0
ixt33 = 0
ixt41 = 0
ixt42 = 0
ixt43 = 0
ntg = 99
jvol = 0
kskip = 0
ksk = 0

do 150 j = 1,  10
  kfa(j) = 0
  ma(j) = 1
150 continue
do 160 j = 1,  15
 ib(j) = 1
 tuz(j) = 0.0
 dtuz(j) = 0.0
160 continue

nibi = 0
nibj = 0
nibib = 0
nibjb = 0
imq1 = 0
imq2 = 0
imq3 = 0
imqb1 = 0
imqb2 = 0
imqb3 = 0
mqc1 = 0
mqc2 = 0
mqc3 = 0
mqc11 = 0
mqc21 = 0
mqc31 = 0
jac = 1
jor = 0
jor1 = 0
jor2 = 0
jorb = 0
jorb1 = 0
jorb2 = 0
jdum = -302357

c------ Initialization ------------------------
c
if(kfa(1).eq.0) then
call SIGNAL(1, speed, icl, ni, ir, ig, noff,
  lgi)
call INIT(arr1, dep1, arr2, dep2, arr3, dep3,
  nc, ne, ng, k1, k2, k3, i, n, m)
call INIT1(jo, nr, iup, iqi2, iqi3, ncy1, ncy2,
  ncy3, nq2, nq3, nq2i, ind1, iqt, iqe, iqea,
  iqti, nqst)
```

```
call INIT2(nii, isp, iqs, nqs, niq, dtiq, nua,
  dtua, nux, dtux)
call INIT2(nii0, isp0, iqs0, nqs0, niq0, dtiq0,
  nua0, dtua0, nux0, dtux0)
call SIGADJ(ir, ig, lri, lgi, icl, dist, speed,
  jac, kc, arrr2, jdum, jvar)

call SIGNA1(1, ni, ir, ig, ir1, ig1)
call SIGNA1(2, ni, ir, ig, irb, igb)
if(irb(1).gt.0) then
  igb(1) = igb(1) - icl
  irb(1) = irb(1) - icl
endif
call SIGNA1(1, ni, irb, igb, irb1, igb1)

ismt = ig(2) + ncwu * icl

igsu1 = ig1(1)
igsu2 = igb(2)
igsu3 = ig1(3)
call DPHDWY(zz, zsut)
jn = 0
165do 170 kt = 1,  nac1
  jn = jn + 1
  carrl1(jn) = igsu1 + (lgi-ipt) + zz(kt)
  darrr2(jn) = igsu2 + (lgi-ipt) + zz(kt)
  carrl3(jn) = igsu3 + (lgi-ipt) + zz(kt)
170 continue
igsu1 = igsu1 + icl
igsu2 = igsu2 + icl
igsu3 = igsu3 + icl
if(jn.lt.470) go to 165

igf(1) = ig(1)
igf(2) = ig(2)
igf(3) = igb(1)

jgn1(1) = ig(1)
jgn2(1) = ig(2)
jgn3(1) = ig(3)

do 190 k = 2,  100
  jgn1(k) = jgn1(k-1) + icl
  jgn2(k) = jgn2(k-1) + icl
  jgn3(k) = jgn3(k-1) + icl
190 continue
call DPHDWY(zz, zsut)

do 200 im = 1,  100
  do 210 jm = 0,  60,  1
    suts(im, jm) = jgn2(im) + zsut(jm)
    suts0(im, jm) = jgn3(im) + zsut(jm)
    suts1(im, jm) = jgn2(im) + zsut(jm)
    suts2(im, jm) = jgn3(im) + zsut(jm)
    sutsb(im, jm) = jgn2(im) + zsut(jm)
    sutsb0(im, jm) = jgn1(im) + zsut(jm)
    sutsb1(im, jm) = jgn2(im) + zsut(jm)
    sutsb2(im, jm) = jgn1(im) + zsut(jm)
210  continue
200 continue

kfa(1) = 1
endif
C
smtj = 0.
norm1 = 1
norm2 = 1
norm3 = 1
norm4 = 1
jxt11 = 1
jxt12 = 1
jxt13 = 1
jxt21 = 1
jxt22 = 1
jxt23 = 1
jxt31 = 1
jxt32 = 1
jxt33 = 1
jxt41 = 1
jxt42 = 1
jxt43 = 1
krx1 = 0
krx2 = 0
klx3 = 0
ibs5 = 1
kibs5 = 0
ibs7 = 1
kibs7 = 0
ibs9 = 1
kibs9 = 0
1000 continue
jac = 1
klt1 = 0
krt1 = 0
klt2 = 0
krt2 = 0
klt3 = 0
krt3 = 0
kltb1 = 0
krtb1 = 0
kltb2 = 0
krtb2 = 0
kltb3 = 0
krtb3 = 0

if(ig(2).gt.200.and.ig(2).gt.igs(2)) then
  norm2 = 1
  norm1 = 0
endif
if(igb1(3).gt.200.and.igb1(3).gt.igr(2)) then
  norm3 = 1
  norm4 = 0
endif

 if(norm1.eq.1) then
call RIGHTL(1, nvol, dist, iveh, speed, jdq,
  mult, icl, lgi, ku1, arrr2, carrr2, m, m1,
  ma, ntg, jvol, imq1, ki2, ki3, jo, nr, at2,
  arr1, dep1, arr2, dep2, arr3, dep3, i, k1,
  k2, k3, ig, ir, nc, ne, ng, iqi2, iqi3, qs2,
  qs20, nq2, nq3, ncy1, ncy2, ncy3, nka, nka0,
  isp, isp0, nua, nua0, dtua, dtua0, suts,
  suts0, iqs, iqs0, qsi, qsi0, nqs, nqs0,
  rqs2, rqs20, niq, niq0, dtiq, dtiq0, nux,
  nux0, dtux, dtux0, jac, tuz, dtuz, wuz, ib,
  10, 11, 12, nq2i, nuzj1, dtuzj1, n, iup,
  ind1, iqt, qst, iqe, iqea, nii, nii0, iqti,
  qsti, nqst, rqst, jor, mqc21, sutss1, isps1,
  nkas1, ismt, nibi, nibj, ixt11, ixt12,
  ixt13, jxt11, jxt12, jxt13, icy, st, ipt,
  nac, noff, krt2, krx1, irs, ibs7, kibs7,
  jdum, dy1, dy2, dy3, jvar)
if(krx2.eq.1) krt2 = 0
if(m.ge.157) then
endif
endif
```

```
*****   Arterial,  A direction   ********
*****   Left lane

if(kfa(2).eq.0) then
jo2 = 0
call INIT(arr11, dep11, arr21, dep21, arr31,
  dep31, nc1, ne1, ng1, k11, k21, k31, ii, nn,
  m)
call INIT1(jo1, nl, iup1, iqi21, iqi31, ncy11,
  ncy21, ncy31, nql2, nql3, nql3i, ind11,
  iqt1, iqe1, iqea1, iqti1, nqst1)
call INIT2(nii1, isp1, iqs2, nqs2, niq1, dtiq1,
  nua1, dtua1, nux1, dtux1)
call INIT2(nii2, isp2, iqs3, nqs3, niq2, dtiq2,
  nua2, dtua2, nux2, dtux2)
nux3 = 0
dtux3 = 0.
kfa(2) = 1
endif

 if(norm2.eq.1) then
call LEFTL(2, nvol, dist, iveh, speed, jdq,
  mult, icl, lgi, ku2, arrl3, carrl1, m, ma,
  ntg, jvol, imq2, imq3, ki21, ki31, 0, 0,
  ixt21, ixt22, ixt23, jxt21, jxt22, jxt23,
  jo1, nl, at3, arr11, dep11, arr21, dep21,
  arr31, dep31, ii, k11, k21, k31, ig1, ir1,
  nc1, ne1, ng1, iqi21, iqi31, qs21, qs31,
  nql2, nql3, ncy11, ncy21, ncy31, nka1, nka2,
  isp1, isp2, nua1, nua2, dtua1, dtua2, suts1,
  suts2, iqs2, iqs3, qsi1, qsi2, nqs2, nqs3,
  rqs21, rqs31, niq1, niq2, dtiq1, dtiq2,
  nux1, nux2, dtux1, dtux2, jac, tuz, dtuz,
  wuz, ib, 20, 21, 22, nql3i, nuzj2, dtuzj2,
  nn, iup1, ind11, iqt1, qst1, iqe1, iqea1,
  nii2, iqti1, qsti1, nqst1, rqst1, jor1,
  mqc1, mqc3, sutsr, ispr, nkar, ismt, carrl3,
  jo2, nuzj3, dtuzj3, jor2, sutst, ispt, nkat,
  icz, st, ipt, nac, noff, krx2, klx3, nux3,
  dtux3, irr, irt, ibs5, kibs5, ibs9, kibs9,
  jdum, dy4, dy5, dy6, jvar)
endif

*****   Arterial,  B direction   **********
*****   Right lane

if(kfa(3).eq.0) then
call INIT(arrb1, depb1, arrb2, depb2, arrb3,
  depb3, ncb, neb, ngb, kb1, kb2, kb3, i1b,
  nb, m)
call INIT1(job, nrb, iupb, iqib2, iqib3, ncyb1,
  ncyb2, ncyb3, nqb2, nqb3, nqb2i, indb1,
  iqtb, iqeb, iqeab, iqtib, nqstb)
call INIT2(niib, ispb, iqsb, nqsb, niqb, dtiqb,
  nuab, dtuab, nuxb, dtuxb)
call INIT2(niib0, ispb0, iqsb0, nqsb0, niqb0,
  dtiqb0, nuab0, dtuab0, nuxb0, dtuxb0)
kfa(3) = 1
endif

 if(norm3.eq.1) then
call RIGHTL(3, nvol, dist, iveh, speed, jdq,
  mult, icl, lgi, ku3, arrrb2, darrr2, m, m2,
  ma, ntg, jvol, imqb1, kib2, kib3, job, nrb,
  atb2, arrb1, depb1, arrb2, depb2, arrb3,
  depb3, i1b, kb1, kb2, kb3, igb, irb, ncb,
  neb, ngb, iqib2, iqib3, qsb2, qsb20, nqb2,
  nqb3, ncyb1, ncyb2, ncyb3, nkab, nkab0,
  ispb, ispb0, nuab, nuab0, dtuab, dtuab0,
  sutsb, sutsb0, iqsb, iqsb0, qsib, qsib0,
  nqsb, nqsb0, rqsb2, rqsb20, niqb, niqb0,
  dtiqb, dtiqb0, nuxb, nuxb0, dtuxb, dtuxb0,
  jac, tuz, dtuz, wuz, ib, 13, 14, 15, nqb2i,
  nuzjb1, btuzj1, nb, iupb, indb1, iqtb, qstb,
  iqeb, iqeab, niib, niib0, iqtib, qstib,
  nqstb, rqstb, jorb, mqc2, sutss, isps, nkas,
  ismt, nibib, nibjb, ixt31, ixt32, ixt33,
  jxt31, jxt32, jxt33, icy, st, ipt, nac,
  noff, krtb1, krx1, irs, ibs7, kibs7, jdum,
  dya1, dya2, dya3, jvar)
endif

*****   Arterial,  B direction   **********
*****   Left lane

if(kfa(4).eq.0) then
job2 = 0
call INIT(arrb11, depb11, arrb21, depb21,
  arrb31, depb31, ncb1, neb1, ngb1, kb11,
  kb21, kb31, iib, nnb, m)
call INIT1(job1, nlb, iupb1, iqib21, iqib31,
  ncyb11, ncyb21, ncyb31, nqlb2, nqlb3,
  nqlb3i, indb11, iqtb1, iqeb1, iqeab1,
  iqtib1, nqstb1)
call INIT2(niib1, ispb1, iqsb2, nqsb2, niqb1,
  dtiqb1, nuab1, dtuab1, nuxb1, dtuxb1)
call INIT2(niib2, ispb2, iqsb3, nqsb3, niqb2,
  dtiqb2, nuab2, dtuab2, nuxb2, dtuxb2)
nuxb3 = 0
dtuxb3 = 0.
kfa(4) = 1
endif

if(norm4.eq.1) then
call LEFTL(4, nvol, dist, iveh, speed, jdq,
  mult, icl, lgi, ku4, arrlb3, darrl1, m, ma,
  ntg, jvol, imqb2, imqb3, kib21, kib31,
  krtb2, krtb3, ixt41, ixt42, ixt43, jxt41,
  jxt42, jxt43, job1, nlb, atb3, arrb11,
  depb11, arrb21, depb21, arrb31, depb31, iib,
  kb11, kb21, kb31, igb1, irb1, ncb1, neb1,
  ngb1, iqib21, iqib31, qsb21, qsb31, nqlb2,
  nqlb3, ncyb11, ncyb21, ncyb31, nkab1, nkab2,
  ispb1, ispb2, nuab1, nuab2, dtuab1, dtuab2,
  sutsb1, sutsb2, iqsb2, iqsb3, qsib1, qsib2,
  nqsb2, nqsb3, rqsb21, rqsb31, niqb1, niqb2,
  dtiqb1, dtiqb2, nuxb1, nuxb2, dtuxb1,
  dtuxb2, jac, tuz, dtuz, wuz, ib, 23, 24, 25,
  nqlb3i, nuzjb2, dtuzjb2, nnb, iupb1, indb11,
  iqtb1, qstb1, iqeb1, iqeab1, niib2, iqtib1,
  qstib1, nqstb1, rqstb1, jorb1, mqc31, mqc11,
  sutst1, ispt1, nkat1, ismt, darrl3, job2,
  nuzjb3, dtuzjb3, jorb2, sutsr1, ispr1,
  nkar1, icz, st, ipt, nac, noff, krx2, klx3,
  nuxb3, dtuxb3, irr, irt, ibs5, kibs5, ibs9,
  kibs9, jdum, dya4, dya5, dya6, jvar)
endif

*****   1st Cross Street   *************

norm1 = 0
norm2 = 0
norm3 = 0
norm4 = 0
```

```
 if(ixt33.eq.1.and.jxt33.eq.1.or.
   ixt43.eq.1.and.jxt43.eq.1.or.imqb3.eq.1)
   then
  if(ixt33.eq.1.and.jxt33.eq.1) then
    klt1 = 1
    jxt33 = 0
  endif
  if(imqb3.eq.1) krt1 = 1
  if(ixt43.eq.1.and.jxt43.eq.1) then
    krt1 = 1
    jxt43 = 0
  endif
 write(ku15)(mj(imj), imj=1, 48)
 write(ku15)(mj(imj), imj=49, 61)
 write(ku15)(bj(ibj), ibj=1, 18)
 rewind(ku15)
 if(krx1.eq.1) krt1 = 0
 call CROSST(1, ku5, ku6, tdc, carr11, darr13,
   arrrb4, arrr2, nvol, lri, noff, icl, jdq,
   ipz, jpz, speed, ntg, mult, jvol, tuz, dtuz,
   wuz, ib, mqc1, mqc11, klt1, krt1, 30, 31,
   32, 33, ispr, nkar, sutsr, ispr1, nkar1,
   sutsr1, dist, iveh, ismt, irr, igr, irr1,
   igr1, igf, kfa, ma, st, ipt, nca, 5, iqir2,
   qsr2, iqsr, qsir, nqsr, rqsr2, iqtr, qstr,
   iqtir, qstir, nqstr, rqstr, kir2, kir3, 6,
   iqir21, qsr21, iqsr1, qsir1, nqsr1, rqsr21,
   iqtr1, qstr1, iqtir1, qstir1, nqstr1,
   rqstr1, kir21, kir31, carr1, cdep1, carr2,
   cdep2, carr3, cdep3, darr1, ddep1, darr2,
   ddep2, darr3, ddep3, isp, nka, suts, 0, isp,
   nka, suts, nq2i, jdum, dy11, dy12, dy13,
   dy14, dy15, dy16, jvar, ig, igb1)
 read(ku15)(mj(imj), imj=1, 48)
 read(ku15)(mj(imj), imj=49, 61)
 read(ku15)(bj(ibj), ibj=1, 18)
 rewind(ku15)
 endif

 *****   2nd Cross Street   ********

 if(ixt22.eq.1.and.jxt22.eq.1.or.imq1.eq.1.or.kr
   t2.eq.1.or.
   ixt12.eq.1.and.jxt12.eq.1.and.krx2.ne.1)
   then
  if(ixt22.eq.1.and.jxt22.eq.1) then
    klt2 = 1
    jxt22 = 0
  endif
  if(imq1.eq.1.and.krx2.ne.1) krt2 = 1
  if(ixt12.eq.1.and.jxt12.eq.1) then
    krt2 = 1
    jxt12 = 0
  endif
 write(ku15)(mj1(imj), imj=1, 48)
 write(ku15)(mj1(imj), imj=49, 61)
 write(ku15)(bj1(ibj), ibj=1, 18)
 rewind(ku15)
 if(krx2.eq.1) krt2 = 0
 call CROSST(2, ku7, ku8, tdc, darrr2, carrr2,
   arrl3, arrlb3, nvol, lri, noff, icl, jdq,
   ipz, jpz, speed, ntg, mult, jvol, tuz, dtuz,
   wuz, ib, mqc2, mqc21, klt2, krt2, 40, 41,
   42, 43, isps, nkas, sutss, isps1, nkas1,
   sutss1, dist, iveh, ismt, irs, igs, irs1,
   igs1, igf, kfa, ma, st, ipt, nca, 7, iqis2,
   qss2, iqss, qsis, nqss, rqss2, iqts, qsts,
   iqtis, qstis, nqsts, rqsts, kis2, kis3, 8,
   iqis21, qss21, iqss1, qsis1, nqss1, rqss21,
   iqts1, qsts1, iqtis1, qstis1, nqsts1,
   rqsts1, kis21, kis31, earr1, edep1, earr2,
   edep2, earr3, edep3, farr1, fdep1, farr2,
   fdep2, farr3, fdep3, isp2, nka2, suts2,
   nql3i, ispb2, nkab2, sutsb2, nqlb3i, jdum,
   dy21, dy22, dy23, dy24, dy25, dy26, jvar,
   ig1, igb1)
 read(ku15)(mj1(imj), imj=1, 48)
 read(ku15)(mj1(imj), imj=49, 61)
 read(ku15)(bj1(ibj), ibj=1, 18)
 rewind(ku15)
 endif

 *****   3rd Cross Street   ********

 minv = min(ig(2), ig(3), ig1(1), ig1(2),
   ig1(3), igb(1), igb(2),  igb(3), igb1(3))

 mpd1 = (minv-ig(1))*(minv-ig(2))*(minv-ig(3))
 mpd2 = (minv-ig1(1))*(minv-ig1(2))*(minv-
   ig1(3))
 mpd3 = (minv-igb(1))*(minv-igb(2))*(minv-
   igb(3))
 mpd4 = (minv-igb1(1))*(minv-igb1(2))*(minv-
   igb1(3))
 if(mpd1.eq.0) norm1 = 1
 if(mpd2.eq.0) norm2 = 1
 if(mpd3.eq.0) norm3 = 1
 if(mpd4.eq.0) norm4 = 1

 if(igr1(2).gt.ig(1)+icl) norm1 = 1
 if(ixt31.eq.1.and.jxt31.eq.1.and.klx3.eq.0.or.i
   mqb2.eq.1
   .or.ixt41.eq.1.and.jxt41.eq.1.or.krtb3.eq.1)
   then
  if(ixt31.eq.1.and.jxt31.eq.1.and.klx3.eq.0)
   then
    klt3 = 1
    jxt31 = 0
  endif
  if(imqb2.eq.1.or.krtb3.eq.1) krt3 = 1
  if(ixt41.eq.1.and.jxt41.eq.1) then
    krt3 = 1
    jxt41 = 0
  endif
 write(ku15)(mj2(imj), imj=1, 48)
 write(ku15)(mj2(imj), imj=49, 61)
 write(ku15)(bj2(ibj), ibj=1, 18)
 rewind(ku15)
 if(klx3.eq.1) klt3 = 0
 call CROSST(3, ku9, ku10, tdc, carr13, darr11,
   arrrb2, arrr4, nvol, lri, noff, icl, jdq,
   ipz, jpz, speed, ntg, mult, jvol, tuz, dtuz,
   wuz, ib, mqc3, mqc31, klt3, krt3, 50, 51,
   52, 53, ispt, nkat, sutst, ispt1, nkat1,
   sutst1, dist, iveh, ismt, irt, igt, irt1,
   igt1, igf, kfa, ma, st, ipt, nca, 9, iqit2,
   qst2, iqst, qsit, nqsu, rqst2, iqtt, qstt,
   iqtit, qstit, nqstt, rqstt, kit2, kit3, 10,
   iqit21, qst21, iqst1, qsit1, nqsu1, rqst21,
   iqtt1, qstt1, iqtit1, qstit1, nqstt1,
   rqstt1, kit21, kit31, garr1, gdep1, garr2,
   gdep2, garr3, gdep3, harr1, hdep1, harr2,
   hdep2, harr3, hdep3, ispb, nkab, sutsb,
   nqb2i, ispb, nkab, sutsb, 0, jdum, dy31,
   dy32, dy33, dy34, dy35, dy36, jvar, ig,
   igb1)
```

```
read(ku15)(mj2(imj), imj=1, 48)
read(ku15)(mj2(imj), imj=49, 61)
read(ku15)(bj2(ibj), ibj=1, 18)
rewind(ku15)
endif

if(jvol.ge.nvol+400.and.kskip.ne.1) then

  ma10 = ma(1) - 1
  mb10 = ma(2) - 1
  ma11 = ma(3) - 1
  mb11 = ma(4) - 1
  mc10 = ma(5) - 1
  md10 = ma(6) - 1
  mc11 = ma(7) - 1
  md11 = ma(8) - 1
  mc12 = ma(9) - 1
  md12 = ma(10) - 1

  call NOVWUP(dep1, dep2, dep3, ismt, nv1,
  ma10)
  call NOVWUP(dep11, dep21, dep31, ismt, nv2,
  mb10)
  call NOVWUP(depb1, depb2, depb3, ismt, nv3,
  ma11)
  call NOVWUP(depb11, depb21, depb31, ismt,
  nv4, mb11)
  call NOVWUP(cdep1, cdep2, cdep3, ismt, nv5,
  mc10)
  call NOVWUP(ddep1, ddep2, ddep3, ismt, nv6,
  md10)
  call NOVWUP(edep1, edep2, edep3, ismt, nv7,
  mc11)
  call NOVWUP(fdep1, fdep2, fdep3, ismt, nv8,
  md11)
  call NOVWUP(gdep1, gdep2, gdep3, ismt, nv9,
  mc12)
  call NOVWUP(hdep1, hdep2, hdep3, ismt, nv10,
  md12)
  nwu = nv1+nv2+nv3+nv4+nv5+nv6+nv7+nv8+nv9
  +nv10

  call findep(dep2, ma10)
  call findep(dep21, mb10)
  call findep(depb2, ma11)
  call findep(depb21, mb11)
  call findep(cdep2, mc10)
  call findep(ddep2, md10)
  call findep(edep2, mc11)
  call findep(fdep2, md11)
  call findep(gdep2, mc12)
  call findep(hdep2, md12)
  dep(1) = dep2(ma10)
  dep(2) = dep21(mb10)
  dep(3) = depb2(ma11)
  dep(4) = depb21(mb11)
  dep(5) = cdep2(mc10)
  dep(6) = ddep2(md10)
  dep(7) = edep2(mc11)
  dep(8) = fdep2(md11)
  dep(9) = gdep2(mc12)
  dep(10) = hdep2(md12)
  call piksrt(10, dep)
  jmin = 1

if(dep(8).gt.dep(7)+1000.or.ksk.ge.4.and.jmin.e
  q.7) jmin = 8
if(dep(7).gt.dep(6)+1000.or.ksk.ge.4.and.jmin.e
  q.6) jmin = 7
if(dep(6).gt.dep(5)+1000.or.ksk.ge.4.and.jmin.e
  q.5) jmin = 6
if(dep(5).gt.dep(4)+1000.or.ksk.ge.4.and.jmin.e
  q.4) jmin = 5
if(dep(4).gt.dep(3)+1000.or.ksk.ge.4.and.jmin.e
  q.3) jmin = 4
if(dep(3).gt.dep(2)+1000.or.ksk.ge.4.and.jmin.e
  q.2) jmin = 3
if(dep(2).gt.dep(1)+1000.or.ksk.ge.4.and.jmin.e
  q.1) jmin = 2

220  depmin = dep(jmin)
  diffd = abs(depmin - depmp)
  if(diffd.le.0.01) ksk = ksk + 1

  if(diffd.gt.0.01) then
    call NOVSIM(dep2, ma10, depmin, nov1)
    call NOVSIM(dep21, mb10, depmin, nov2)
    call NOVSIM(depb2, ma11, depmin, nov3)
    call NOVSIM(depb21, mb11, depmin, nov4)
    call NOVSIM(cdep2, mc10, depmin, nov5)
    call NOVSIM(ddep2, md10, depmin, nov6)
    call NOVSIM(edep2, mc11, depmin, nov7)
    call NOVSIM(fdep2, md11, depmin, nov8)
    call NOVSIM(gdep2, mc12, depmin, nov9)
    call NOVSIM(hdep2, md12, depmin, nov10)
    novt = nov1+nov2+nov3+nov4+nov5+nov6+nov7
  +nov8+nov9+nov10

    if(novt.ge.nvol+nwu) then
      kskip = 1
      go to 250
    endif
  endif
  depmp = depmin
endif

250 if(kskip.eq.1) then
  do 260 time = depmin,  depmin-1000,  -1.0
    call NOVSIM(dep2, ma10, time, nov1)
    call NOVSIM(dep21, mb10, time, nov2)
    call NOVSIM(depb2, ma11, time, nov3)
    call NOVSIM(depb21, mb11, time, nov4)
    call NOVSIM(cdep2, mc10, time, nov5)
    call NOVSIM(ddep2, md10, time, nov6)
    call NOVSIM(edep2, mc11, time, nov7)
    call NOVSIM(fdep2, md11, time, nov8)
    call NOVSIM(gdep2, mc12, time, nov9)
    call NOVSIM(hdep2, md12, time, nov10)
    nsim = nov1+nov2+nov3+nov4+nov5+nov6+nov7
  +nov8+nov9+nov10-nwu
    if(nsim.le.nvol) go to 270
260  continue
endif
if(jvol.lt.nvol+5000) go to 1000

270smtj = time
nos1 = nov1 - nv1
if(nos1.lt.0) nos1 = 0
nos2 = nov2 - nv2
nos3 = nov3 - nv3
nos4 = nov4 - nv4
if(nos4.lt.0) nos4 = 0
nos5 = nov5 - nv5
nos6 = nov6 - nv6
if(nos6.lt.0) nos6 = 0
```

```
nos7 = nov7 - nv7
nos8 = nov8 - nv8
if(nos8.lt.0) nos8 = 0
nos9 = nov9 - nv9
if(nos9.lt.0) nos9 = 0
nos10 = nov10 - nv10

****** Print queue spillback statistics

**arterial,  right lane (A dirction)
smpd = smtj - ismt
jcy = int(smpd/icl) + 1
call PRNQS(ku1, 2, icl, lgi, iqi2, qs2, jcy,
  iqs, qsi, nqs, rqs2, prqs, pqs, pqsn, ptqs,
  ismt, smtj, mult)
call PRNQST(ku1, 2, iqt, qst, iqti, qsti, jcy,
  nqst, rqst, prqst, pqst, pqstn, ptqst, ismt,
  smtj, mult)
call PRNQS(ku1, 3, icl, lgi, iqi3, qs20, jcy,
  iqs0, qsi0, nqs0, rqs20, prqs1, pqs1, pqsn1,
  ptqs1, ismt, smtj, mult)

**arterial,  left lane (A dirction)
call PRNQS(ku2, 2, icl, lgi, iqi21, qs21, jcy,
  iqs2, qsi1, nqs2, rqs21, prqs2, pqs2, pqsn2,
  ptqs2, ismt, smtj, mult)
call PRNQS(ku2, 3, icl, lgi, iqi31, qs31, jcy,
  iqs3, qsi2, nqs3, rqs31, prqs3, pqs3, pqsn3,
  ptqs3, ismt, smtj, mult)
call PRNQST(ku2, 3, iqt1, qst1, iqti1, qsti1,
  jcy, nqst1, rqst1, prqst3, pqst3, pqstn3,
  ptqst3, ismt, smtj, mult)

**arterial,  right lane (B dirction)
call PRNQS(ku3, 2, icl, lgi, iqib2, qsb2, jcy,
  iqsb, qsib, nqsb, rqsb2, brqs, bqs, bqsn,
  btqs, ismt, smtj, mult)
call PRNQST(ku3, 2, iqtb, qstb, iqtib, qstib,
  jcy, nqstb, rqstb, brqst, bqst, bqstn,
  btqst, ismt, smtj, mult)
call PRNQS(ku3, 3, icl, lgi, iqib3, qsb20, jcy,
  iqsb0, qsib0, nqsb0, rqsb20, brqs1, bqs1,
  bqsn1, btqs1, ismt, smtj, mult)

**arterial,  left lane (B dirction)
call PRNQS(ku4, 2, icl, lgi, iqib21, qsb21,
  jcy, iqsb2, qsib1, nqsb2, rqsb21, brqs2,
  bqs2, bqsn2, btqs2, ismt, smtj, mult)
call PRNQS(ku4, 3, icl, lgi, iqib31, qsb31,
  jcy, iqsb3, qsib2, nqsb3, rqsb31, brqs3,
  bqs3, bqsn3, btqs3, ismt, smtj, mult)
call PRNQST(ku4, 3, iqtb1, qstb1, iqtib1,
  qstib1, jcy, nqstb1, rqstb1, brqst3, bqst3,
  bqstn3, btqst3, ismt, smtj, mult)

**cross street 1,  left lane
call PRNQS(ku5, 3, icl, lri, iqir2, qsr2, jcy,
  iqsr, qsir, nqsr, rqsr2, prqa, pqa, pqna,
  ptqa, ismt, smtj, mult)
call PRNQST(ku5, 3, iqtr, qstr, iqtir, qstir,
  jcy, nqstr, rqstr, prqta, pqta, pqtna,
  ptqta, ismt, smtj, mult)

**cross street 1,  right lane
call PRNQS(ku6, 3, icl, lri, iqir21, qsr21,
  jcy, iqsr1, qsir1, nqsr1, rqsr21, prqa1,
  pqa1, pqna1, ptqa1, ismt, smtj, mult)
call PRNQST(ku6, 3, iqtr1, qstr1, iqtir1,
  qstir1, jcy, nqstr1, rqstr1, prqta1, pqta1,
  pqtna1, ptqta1, ismt, smtj, mult)

**cross street 2,  left lane
call PRNQS(ku7, 3, icl, lri, iqis2, qss2, jcy,
  iqss, qsis, nqss, rqss2, prqb, pqb, pqnb,
  ptqb, ismt, smtj, mult)
call PRNQST(ku7, 3, iqts, qsts, iqtis, qstis,
  jcy, nqsts, rqsts, prqtb, pqtb, pqtnb,
  ptqtb, ismt, smtj, mult)

**cross street 2,  right lane
call PRNQS(ku8, 3, icl, lri, iqis21, qss21,
  jcy, iqss1, qsis1, nqss1, rqss21, ptqb1,
  pqb1, pqnb1, ptqb1, ismt, smtj, mult)
call PRNQST(ku8, 3, iqts1, qsts1, iqtis1,
  qstis1, jcy, nqsts1, rqsts1, prqtb1, pqtb1,
  pqtnb1, ptqtb1, ismt, smtj, mult)

**cross street 3,  left lane
call PRNQS(ku9, 3, icl, lri, iqit2, qst2, jcy,
  iqst, qsit, nqsu, rqst2, prqc, pqc, pqnc,
  ptqc, ismt, smtj, mult)
call PRNQST(ku9, 3, iqtt, qstt, iqtit, qstit,
  jcy, nqstt, rqstt, prqtc, pqtc, pqtnc,
  ptqtc, ismt, smtj, mult)

**cross street 3,  right lane
call PRNQS(ku10, 3, icl, lri, iqit21, qst21,
  jcy, iqst1, qsit1, nqsu1, rqst21, prqc1,
  pqc1, pqnc1, ptqc1, ismt, smtj, mult)
call PRNQST(ku10, 3, iqtt1, qstt1, iqtit1,
  qstit1, jcy, nqstt1, rqstt1, prqtc1, pqtc1,
  pqtnc1, ptqtc1, ismt, smtj, mult)

*****
if(mult.eq.1) then

c   write(*, 310) ip*incre, jp*incre
  write(ku1, 310) ip*incre, jp*incre
310  format(/' offset2=', i3, ',  offset3=',
  i3)

  pta1(ip, jp) = ptqs + ptqst + ptqs1
  pta2(ip, jp) = ptqs2 + ptqs3 + ptqst3
  pta3(ip, jp) = btqs + btqst + btqs1
  pta4(ip, jp) = btqs2 + btqs3 + btqst3
  ptc1(ip, jp) = ptqa + ptqta + ptqa1 + ptqta1
  ptc2(ip, jp) = ptqb + ptqtb + ptqb1 + ptqtb1
  ptc3(ip, jp) = ptqc + ptqtc + ptqc1 + ptqtc1
  ptat(ip, jp) = pta1(ip, jp) + pta2(ip, jp) +
  pta3(ip, jp) + pta4(ip, jp)
  ptct(ip, jp) = ptc1(ip, jp) + ptc2(ip, jp) +
  ptc3(ip, jp)
  ptst(ip, jp) = ptat(ip, jp) + ptct(ip, jp)
  spst(ip, jp) = smpd
  nwu(ip, jp) = nwu

  nar(ip, jp) = nos1
  nal(ip, jp) = nos2
  narb(ip, jp) = nos3
  nalb(ip, jp) = nos4
  ncl1(ip, jp) = nos5
  ncr1(ip, jp) = nos6
  ncl2(ip, jp) = nos7
  ncr2(ip, jp) = nos8
  ncl3(ip, jp) = nos9
```

```
  mcr3(ip, jp) = nos10

  write(ku1, 320) ptat(ip, jp), ptct(ip, jp),
  ptst(ip, jp), smpd
320  format(' P(A)=', f5.2, '  P(C)=', f5.2, '
  P(T)=', f5.2,    '  Pd=', f6.0)
endif
*****
if(mult.eq.0) then
  write(ku1, 330) smpd
330  format(2x, 'Simulation Period(Total):',
  f6.0, ' sec')
  write(ku1, 340) nv1, nv2, nv3, nv4, nv5, nv6,
  nv7, nv8, nv9, nv10
  write(ku1, 350) nos1, nos2, nos3, nos4, nos5,
  nos6, nos7, nos8,    nos9, nos10
340  format(/2x, 'Wi(A1R,  A1L,  A2R,  A2L,
  C1L,  C1R,  C2L,  C2R, ',    ' C3L, ', '
  C3R) :', /4x, 9(i4, ', '), i4)
350  format(/2x, 'Si(A1R,  A1L,  A2R,  A2L,
  C1L,  C1R,  C2L,  C2R, ',    ' C3L, ', '
  C3R) :', /4x, 9(i4, ', '), i4)
  jtot = nwu + nsim
  write(ku1, 355) nwu
355  format(/2x, 'V (warmup) : ', i4, '
  vehicles')
  write(ku1, 357) nsim
357  format(/2x, 'V (simulation) : ', i4, '
  vehicles')
  write(ku1, 360) jtot
360  format(/2x, 'V (total) : ', i4, '
  vehicles')
endif

420 continue
410 continue
400 continue

*****  Print summarized outputs

if(mult.eq.1) then
write(ku5, 510)
510 format(/2X, 'No.of queue spillback per
  cycle')
call PRNOP1(ku5, 1, 1, incre, ioff1, ioff2,
  joff1, joff2, pta1)
call PRNOP1(ku5, 1, 2, incre, ioff1, ioff2,
  joff1, joff2, pta2)
call PRNOP1(ku5, 1, 3, incre, ioff1, ioff2,
  joff1, joff2, pta3)
call PRNOP1(ku5, 1, 4, incre, ioff1, ioff2,
  joff1, joff2, pta4)
call PRNOP1(ku5, 2, 1, incre, ioff1, ioff2,
  joff1, joff2, ptc1)
call PRNOP1(ku5, 2, 2, incre, ioff1, ioff2,
  joff1, joff2, ptc2)
call PRNOP1(ku5, 2, 3, incre, ioff1, ioff2,
  joff1, joff2, ptc3)
call PRNOP1(ku5, 3, 9, incre, ioff1, ioff2,
  joff1, joff2, ptat)
call PRNOP1(ku5, 4, 9, incre, ioff1, ioff2,
  joff1, joff2, ptct)
call PRNOP1(ku5, 5, 9, incre, ioff1, ioff2,
  joff1, joff2, ptst)
call PRNOP1(ku2, 6, 9, incre, ioff1, ioff2,
  joff1, joff2, spst)
call PRNOP1(ku2, 5, 9, incre, ioff1, ioff2,
  joff1, joff2, ptst)

write(ku6, 520)
520 format(/2X, 'No.of vehicles simulated')
call PRNOP2(ku6, 1, incre, ioff1, ioff2, joff1,
  joff2, mar)
call PRNOP2(ku6, 2, incre, ioff1, ioff2, joff1,
  joff2, mal)
call PRNOP2(ku6, 3, incre, ioff1, ioff2, joff1,
  joff2, marb)
call PRNOP2(ku6, 4, incre, ioff1, ioff2, joff1,
  joff2, malb)
call PRNOP2(ku6, 5, incre, ioff1, ioff2, joff1,
  joff2, mcl1)
call PRNOP2(ku6, 6, incre, ioff1, ioff2, joff1,
  joff2, mcr1)
call PRNOP2(ku6, 7, incre, ioff1, ioff2, joff1,
  joff2, mcl2)
call PRNOP2(ku6, 8, incre, ioff1, ioff2, joff1,
  joff2, mcr2)
call PRNOP2(ku6, 9, incre, ioff1, ioff2, joff1,
  joff2, mcl3)
call PRNOP2(ku6, 10, incre, ioff1, ioff2,
  joff1, joff2, mcr3)
call PRNOP2(ku2, 99, incre, ioff1, ioff2,
  joff1, joff2, mwu)
endif

stop
end

************************************************
******  SUBROUTINES  ***************************
************************************************

subroutine RIGHTL(j1, nvol, dist, iveh, speed,
  jdq, mult, icl, lgi, ku1, arrr2, carrr2, m,
  m1, ma, ntg, jvol, imq1, ki2, ki3, jo, nr,
  at2, arr1, dep1, arr2, dep2, arr3, dep3, i,
  k1, k2, k3, ig, ir, nc, ne, ng, iqi2, iqi3,
  qs2, qs20, nq2, nq3, ncy1, ncy2, ncy3, nka,
  nka0, isp, isp0, nua, nua0, dtua, dtua0,
  suts, suts0, iqs, iqs0, qsi, qsi0, nqs,
  nqs0, rqs2, rqs20, niq, niq0, dtiq, dtiq0,
  nux, nux0, dtux, dtux0, jac, tuz, dtuz, wuz,
  ib, jwh1, jwh2, jwh3, nq2i, nuzj1, dtuzj1,
  n, iup, ind1, iqt, qst, iqe, iqea, nii,
  nii0, iqti, qsti, nqst, rqst, jor, mqc2,
  sutss, isps, nkas, ismt, nibi, nibj, ixt1,
  ixt2, ixt3, jxt1, jxt2, jxt3, icy, st, ipt,
  nac, noff, krt2, krx1, irs, ibs7, kibs7,
  jdum, dy1, dy2, dy3, jvar)
dimension arr1(0:2000), dep1(0:2000),
  arr2(0:2000)
dimension dep2(0:2000), arr3(0:2000),
  dep3(0:2000)
dimension arrr2(0:500), carrr2(0:500)
dimension ki2(0:2000), ki3(0:2000), zz(0:100),
  zsut(0:100)
dimension dist(20), iveh(20), ir(20), ig(20),
  irs(20)
dimension qs2(80), qs20(80), qst(80), qsti(80)
dimension nka(100), nka0(100), nkas(100)
dimension qsi(80), qsi0(80), rqst(80)
dimension suts(0:100, 0:60), suts0(0:100, 0:60)
dimension rqs2(80), rqs20(80), sutss(0:100,
  0:60)
dimension tuz(15), dtuz(15), wuz(15, 80),
  ib(15)
```

```
dimension ma(10), st(15), noff(20)
dimension dy1(0:60), dy2(0:60), dy3(0:60)

C------ Arrival time at intersection 1 --------
C
do 100 m = ma(j1),  ma(j1)+ntg
if(ig(2).ge.ismt) jvol = jvol + 1

if(ixt1.eq.1) go to 145
if(ixt2.eq.1.and.imq1.ne.1) go to 150
if(ixt3.eq.1) go to 151

ki2(m) = 0
ki3(m) = jo
IF(nr.eq.0) then
    arr1(m) = 2.0 * (m-i)
C
C------ Departure time at intersection 1  -----
C
if(j1.eq.1) icx = 41
jwh = jwh1
call DEPQLS(k1, ig(1), ir(1), zz, dep1, m, icl,
  arr1, nc, arr2, dep2, iqi2, qs2, ki2, nq2,
  ncy1, iveh(1), zsut, nka, isp, nua, dtua,
  suts, iqs, qsi, nqs, rqs2, niq, dtiq, jdq,
  nux, dtux, icx, jwh, jac, ixt1, tuz, dtuz,
  wuz, ib, 0, j1, 14, 15, st, ipt, jdum, dy1,
  jvar)
icx = 0
if(imq1.eq.1) then
  imq1 = 0
  jor = 0
  call UPDATE(ne, k2, ig(2), ir(2), zz, dep2,
  m, icl, ncy2)
  k2 = k2 - 1
endif

if(ixt1.eq.1) jxt1 = 1
if(ixt1.eq.1) go to 110
145 if(ixt1.eq.1) ixt1 = 0
C
C------ Arrival time at intersection 2 --------
C
ELSEIF(iup.eq.1) then
  arr1(m) = parr1
  dep1(m) = pdep1
  nr = 0
  iup = 0
  iqe = 0
  iqea = 0
ELSE
  arr1(m) = parr1
  dep1(m) = pdep1
ENDIF
if(imq1.eq.1) then
  imq1 = 0
  jor = 0
  call UPDATE(ne, k2, ig(2), ir(2), zz, dep2,
  m, icl, ncy2)
  k2 = k2 - 1
endif
if(j1.eq.1) icx = 89
call ARRQLA(arr2, dep2, dep1, m, nc, dist(1),
  speed, at2, arrr2, i, n, nr, iup, ki2, nq2i,
  ind1, iqt, qst, iqe, iqea, iveh(1), icx,
  nka, isp, suts, nii, iqti, qsti, niq, dtiq,
  nqst, rqst, jdq, arnp, icw, jac, jdum, jvar)
icx = 0

if(dep1(m).gt.ir(1)+icl) then
  call UPDATE(nc, k1, ig(1), ir(1), zz, dep1,
  m, icl, ncy1)
  ixt1 = 1
  niq = 0
  go to 110
endif
C
C------ Departure time at intersection 2 ------
C
jwh = jwh2
jz2 = 14
jz3 = 15
if(j1.eq.1) then
  jz2 = 8
  jz3 = 7
endif
call DEPQLS(k2, ig(2), ir(2), zz, dep2, m, icl,
  arr2, ne, arr3, dep3, iqi3, qs20, ki3, nq3,
  ncy2, iveh(1), zsut, nka0, isp0, nua0,
  dtua0, suts0, iqs0, qsi0, nqs0, rqs20, niq0,
  dtiq0, jdq, nux0, dtux0, icx, jwh, jac,
  ixt2, tuz, dtuz, wuz, ib, 0, 13, jz2, jz3,
  st, ipt, jdum, dy2, jvar)
icx = 0
if(ixt2.eq.1) then
  jor = 0
endif

mprod = (nua0-1)*(niq0-1)
call UPNKA(k2, ig(2), ir(2), zz, dep2, m, jdq,
  lgi, icl, arr2, ne, nq2i, nq3, ncy2, zsut,
  iveh(1), nka, isp, suts, mprod, jac, tuz,
  dtuz, nibi, nibj)
C
C------ Arrival time at intersection 3 --------
C
if(ixt2.eq.1) jxt2 = 1
if(ixt2.eq.1) go to 110
150 if(ixt2.eq.1) then
  ixt2 = 0
  if(j1.eq.1) krx1 = 0


  tuz7 = tuz(7)

  if(j1.eq.1.and.tuz7.gt.ig(2).and.ig(2)+icl.l
  t.irs(2)) then
    tuz7 = wuz(7, ibs7)
    if(dep2(m).gt.wuz(7,
  ibs7).and.dep2(m).lt.wuz(7, ibs7+1))    then
      ibs7 = ibs7 + 1
      tuz7 = wuz(7, ibs7)
    endif
    if(tuz7.gt.ig(2)+icl) go to 120
    kibs7 = 1
  endif


  if(j1.eq.1.and.(tuz7.gt.ig(2).or.tuz(8).gt.i
  g(2))) then
    jz = 7
    if(tuz7.lt.tuz(8)) jz = 8
    if(jz.eq.7) then
      ibs7 = ibs7 + 1
      kibs7 = 0
    endif
    dtuz(jz) = tuz7 - ig(2)
```

```
      dep2(m) = tuz7 + 2.04
      if(jz.eq.8) dtuz(jz) = tuz(8) - ig(2)
      if(jz.eq.8) dep2(m) = tuz(8) + 2.04
      dslgi = lgi - dtuz(jz)
      jveh = iveh(1) * 2
      if(dslgi.lt.jveh) nibj = 1
      ircl1 = ir(2)+icl-ipt
      if(dep2(m).ge.ircl1) then
        call UPDATE(ne, k2, ig(2), ir(2), zz,
    dep2, m, icl, ncy2)
        call UPDATE(ng, k3, ig(3), ir(3), zz,
    dep3, m, icl, ncy3)
        nka(isp) = 0
        isp = isp + 1
        nka0(isp0) = 0
        isp0 = isp0 + 1
        ixt2 = 1
        krt2 = 1
        krx1 = 1
        go to 110
      endif
    endif
  endif

120 if(nr.eq.0) then
if((nac.eq.1.and.(k2.eq.2.or.jdq.eq.11.and.
   (k2.eq.4.or.k2.eq.5))).or.(nac.eq.2.and.
   (k2.eq.2.or.k2.eq.3.or.jdq.eq.11.and.(k2.eq.
   5.or.k2.eq.6.or.k2.eq.8.or.k2.eq.9)))) then
   jo = jo + 1
   if(j1.eq.1) then
    carrr2(jo) = dep2(m)
    if(mqc2.eq.iveh(1).or.jor+mqc2.eq.iveh(1))
   then
     sss = st(8)
     ssd = sss - dep2(m)
     if(sss.ge.ir(2)+icl) then
       imq1 = 1
     elseif(ssd.gt.0) then
       nux0 = 1
       dtux0 = ssd
     endif
     if(ssd.gt.lgi-20) nibi = 1
    endif
   endif
   jor = jor + 1
   go to 155
endif
endif

if(dep2(m).lt.arr2(m)) dep2(m) = arr2(m)

call ARRNT(arr3, dep3, dep2, m, ne, dist(2),
   speed, nq3, jac,  jdum, jvar)
C
C------ Departure time at intersection 3 ------
C
jwh = jwh3
call DEPART(k3, ig(3), ir(3), zz, dep3, m, icl,
   arr3, ng, ncy3, jac, ixt3, jwh, ipt, jdum,
   dy3, jvar)
mprod = 1
if(j1.eq.1) then
endif
call UPDNKA(k3, ig(3), ir(3), zz, dep3, m, jdq,
   lgi, icl, arr3,  ng, nq3, 0, ncy3, zsut,
   iveh(1), nka0, isp0, suts0, mprod, jac)

if(ixt3.eq.1) jxt3 = 1
if(ixt3.eq.1) go to 110
151 if(ixt3.eq.1) ixt3 = 0
C
C------ Print arrival and departure time at
   each intersection ---------
C
155 if(nr.ne.0) then
   parr1 = arr1(m)
   pdep1 = dep1(m)
   arr1(m) = 0.
   dep1(m) = 0.
endif

ki3p = jo

if(dep3(m).lt.arr3(m)) dep3(m) = arr3(m)

if(mult.eq.0) then
call FRINT(arr1, dep1, arr2, dep2, arr3, dep3,
   ku1, m, i, jo, nq2i)
endif
if(ncy2.lt.isp) then
   do 160 ik = isp,  isp
     nka(ik) = nka(ik) + 1
160  continue
else
   do 161 ik = isp,  ncy2
     nka(ik) = nka(ik) + 1
161  continue
endif

if(ki3p.ne.ki3(m)) go to 164
if(ncy3.lt.isp0) then
   do 162 ik = isp0,  isp0
     nka0(ik) = nka0(ik) + 1
162  continue
else
   do 163 ik = isp0,  ncy3
     nka0(ik) = nka0(ik) + 1
163  continue
endif
164 continue

if(nr.ne.0) then
   arr1(m) = parr1
   dep1(m) = pdep1
endif

if(imq1.eq.1) go to 110

100 continue
110ma(j1) = m
if(ig(2).ge.ismt) jvol = jvol - 1
if(imq1.eq.1.and.ixt1.ne.1.and.ixt2.ne.1.and.ix
   t3.ne.1) then
   ma(j1) = m + 1
   if(ig(2).ge.ismt) jvol = jvol + 1
endif
return
end
*******
subroutine LEFTL(j1, nvol, dist, iveh, speed,
   jdq, mult, icl, lgi, ku2, arrl3, carrl1, m,
   ma, ntg, jvol, imq2, imq3, ki21, ki31, krt2,
   krt3, ixt1, ixt2, ixt3, jxt1, jxt2, jxt3,
   jo1, nl, at3, arr11, dep11, arr21, dep21,
   arr31, dep31, ii, k11, k21, k31, ig1, ir1,
```

```
   nc1, ne1, ng1, iqi21, iqi31, qs21, qs31,
   nql2, nql3, ncy11, ncy21, ncy31, nka1, nka2,
   isp1, isp2, nua1, nua2, dtua1, dtua2, suts1,
   suts2, iqs2, iqs3, qsi1, qsi2, nqs2, nqs3,
   rqs21, rqs31, niq1, niq2, dtiq1, dtiq2,
   nux1, nux2, dtux1, dtux2, jac, tuz, dtuz,
   wuz, ib, jwh1, jwh2, jwh3, nql3i, nuzj2,
   dtuzj2, nn, iup1, ind11, iqt1, qst1, iqe1,
   iqea1, nii2, iqti1, qsti1, nqst1, rqst1,
   jor1, mqc1, mqc2, sutsr, ispr, nkar, ismt,
   carrl3, jo2, nuzj3, dtuzj3, jor2, sutst,
   ispt, nkat, icz, st, ipt, nac, noff, krx2,
   klx3, nux3, dtux3, irr, irt, ibs5, kibs5,
   ibs9, kibs9, jdum, dy1, dy2, dy3, jvar)
dimension arr11(0:2000), dep11(0:2000),
  arr21(0:2000)
dimension dep21(0:2000), arr31(0:2000),
  dep31(0:2000)
dimension arrl3(0:500), carrl1(0:500)
dimension ki21(0:2000), ki31(0:2000),
  zz(0:100), zsut(0:100)
dimension dist(20), iveh(20), ir1(20), ig1(20),
  irr(20), irt(20)
dimension qs21(80), qs31(80), qst1(80),
  qsti1(80)
dimension nka1(100), nka2(100), nkar(100),
  nkat(100)
dimension qsi1(80), qsi2(80), rqst1(80)
dimension suts1(0:100, 0:60), suts2(0:100,
  0:60)
dimension sutsr(0:100, 0:60), sutst(0:100,
  0:60)
dimension rqs21(80), rqs31(80), carrl3(0:500)
dimension tuz(15), dtuz(15), wuz(15, 80),
  ib(15)
dimension ma(10), st(15), noff(20)
 dimension dy1(0:60), dy2(0:60), dy3(0:60)

C------ Arrival time at intersection 1 --------
C
do 200 m = ma(j1),  ma(j1)+ntg
if(ig1(2).ge.ismt) jvol = jvol + 1

if(ixt1.eq.1.and.imq2.ne.1) go to 245
if(ixt2.eq.1) go to 250
if(ixt3.eq.1.and.imq3.ne.1) go to 251

ki21(m) = ii + jo1
ki31(m) = jo1
C
IF(nl.eq.0) then
    arr11(m) = 2.0 * (m-ii)
C
C------ Departure time at intersection 1  -----
C
if(imq2.eq.1) then
  imq2 = 0
  call UPDATE(nc1, k11, ig1(1), ir1(1), zz,
  dep11, m, icl, ncy11)
  k11 = k11 - 1
  jor1 = 0
endif

jwh = jwh1
jz2 = 14
jz3 = 15
if(j1.eq.4) then
  jz2 = 10
  jz3 = 9
endif
call DEPQLS(k11, ig1(1), ir1(1), zz, dep11, m,
  icl, arr11, nc1, arr21, dep21, iqi21, qs21,
  ki21, nql2, ncy11, iveh(1), zsut, nka1,
  isp1, nua1, dtua1, suts1, iqs2, qsi1, nqs2,
  rqs21, niq1, dtiq1, jdq, nux1, dtux1, icx,
  jwh, jac, ixt1, tuz, dtuz, wuz, ib, 0, 13,
  jz2, jz3, st, ipt, jdum, dy1, jvar)
if(ixt1.eq.1) then
  jor1 = 0
  jxt1 = 1
endif

if(ixt1.eq.1) then
if(imq3.eq.1) then
  imq3 = 0
  call UPDATE(ng1, k31, ig1(3), ir1(3), zz,
  dep31, m, icl, ncy31)
  k31 = k31 - 1
  jor2 = 0
endif
go to 210
endif

if(ixt1.eq.1) go to 210
245 if(ixt1.eq.1) then

  ixt1 = 0
  tuz9 = tuz(9)

  if(j1.eq.4.and.tuz9.gt.ig1(1).and.ig1(1)+icl
  .lt.irt(2))then
    tuz9 = wuz(9, ibs9)
    if(dep11(m).gt.wuz(9,
  ibs9).and.dep11(m).lt.wuz(9, ibs9+1))
  then
      ibs9 = ibs9 + 1
      tuz9 = wuz(9, ibs9)
    endif
    if(tuz9.gt.ig1(1)+icl) go to 220
    kibs9 = 1
  endif

  if(j1.eq.4.and.(tuz9.gt.ig1(1).or.tuz(10).gt
  .ig1(1))) then
    jz = 9
    if(tuz9.lt.tuz(10)) jz = 10
    if(jz.eq.9) then
      ibs9 = ibs9 + 1
      kibs9 = 0
    endif
    dep11(m) = tuz9 + 2.04
    if(jz.eq.10) dep11(m) = tuz(10) + 2.04
    ircl1 = ir1(1)+icl-ipt
    ircl3 = ir1(3)+icl-ipt
    if(dep11(m).ge.ircl1) then


  if(j1.eq.4.and.(tuz(5).gt.ig1(3).or.tuz(6).g
  t.ig1(3))) then
    jz = 5
    if(jz.eq.5) then
      ibs5 = ibs5 + 1
      kibs5 = 0
    endif
    if(tuz(5).lt.tuz(6)) jz = 6
    dep31(m) = tuz(5) + 2.04
```

```
    if(jz.eq.6) dep31(m) = tuz(6) + 2.04
    ircls = ir1(3)+icl-ipt
    if(dep31(m).ge.ircl3) then
      krx2 = 1
      klx3 = 1
    endif
  endif

     call UPDATE(nc1, k11, ig1(1), ir1(1), zz,
  dep11, m, icl, ncy11)
     call UPDATE(ne1, k21, ig1(2), ir1(2), zz,
  dep21, m, icl, ncy21)
     call UPDATE(ng1, k31, ig1(3), ir1(3), zz,
  dep31, m, icl, ncy31)
     nka1(isp1) = 0
     isp1 = isp1 + 1
     nka2(isp2) = 0
     isp2 = isp2 + 1
     ixt1 = 1
     jxt1 = 1
     krt3 = 1
     go to 210
    endif
  endif
endif
C
C------ Arrival time at intersection 2 --------
C
220 if((nac.eq.1.and.(k11.eq.2.or.jdq.eq.11.
  and.(k11.eq.4.or.k11.eq.5))).or.(nac.eq.2.an
  d. (k11.eq.2.or.k11.eq.3.or.jdq.eq.11.and.
  (k11.eq.5.or.k11.eq.6.or.k11.eq.8.or.k11.eq.
  9)))) then
  jo1 = jo1 + 1
  if(j1.eq.4) then
   carr11(jo1) = dep11(m)
   if(mqc1.eq.iveh(1).or.jor1+mqc1.eq.iveh(1))
  then
    sss = st(10)
    ssd = sss - dep11(m)
    if(sss.ge.ir1(1)+icl) then
      imq2 = 1
    elseif(ssd.gt.0) then
      nux1 = 1
      dtux1 = ssd
    endif
   endif
  endif
  jor1 = jor1 + 1
  go to 255
endif
*******
115call ARRNTE(arr21, dep21, dep11, m, nc1,
  dist(1), speed, nq12, iveh(1), nka1, isp1,
  suts1, ki21, jac, jdum, jvar)
C
C------ Departure time at intersection 2 ------
  C
jwh = jwh2
if(j1.eq.4) icx = 3
call DEPQLS(k21, ig1(2), ir1(2), zz, dep21, m,
  icl, arr21, ne1, arr31, dep31, iqi31, qs31,
  ki31, nq13, ncy21, iveh(1), zsut, nka2,
  isp2, nua2, dtua2, suts2, iqs3, qsi2, nqs3,
  rqs31, niq2, dtiq2, jdq, nux2, dtux2, icx,
  jwh, jac, ixt2, tuz, dtuz, wuz, ib, 0, j1,
  14, 15, st, ipt, jdum, dy2, jvar)
icx = 0

mprod = (nua2-1)*(nii2-1)*(niq2-1)
call UPDNKA(k21, ig1(2), ir1(2), zz, dep21, m,
  jdq, lgi, icl, arr21, ne1, nq12, nq13,
  ncy21, zsut, iveh(1), nka1, isp1, suts1,
  mprod, jac)

if(imq3.eq.1) then
  imq3 = 0
  call UPDATE(ng1, k31, ig1(3), ir1(3), zz,
  dep31, m, icl, ncy31)
  k31 = k31 - 1
endif
C
C------ Arrival time at intersection 3 -------
C
if(ixt2.eq.1) jxt2 = 1
if(ixt2.eq.1) go to 210
250 if(ixt2.eq.1) ixt2 = 0

ELSEIF(iup1.eq.1) then
  arr11(m) = parr11
  dep11(m) = pdep11
  arr21(m) = parr21
  dep21(m) = pdep21
  n1 = 0
  iup1 = 0
  iqe1 = 0
  iqea1 = 0
ELSE
  arr11(m) = parr11
  dep11(m) = pdep11
  arr21(m) = parr21
  dep21(m) = pdep21
ENDIF

if(imq3.eq.1) then
  imq3 = 0
  call UPDATE(ng1, k31, ig1(3), ir1(3), zz,
  dep31, m, icl, ncy31)
  k31 = k31 - 1
endif

if(dep21(m).lt.arr21(m)) dep21(m) = arr21(m)

if(j1.eq.2) icx = 36
call ARRQLA(arr31, dep31, dep21, m, ne1,
  dist(2), speed, at3, arr13, ii, nn, n1,
  iup1, ki31, nq13i, ind11, iqt1, qst1, iqe1,
  iqea1, iveh(1), icx, nka2, isp2, suts2,
  nii2, iqti1, qsti1, niq2, dtiq2, nqst1,
  rqst1, jdq, amp, icw, jac, jdum, jvar)
icx = 0
if(ind11.eq.1) then
  arr11(m) = parr11
  dep11(m) = pdep11
  arr21(m) = parr21
  dep21(m) = pdep21
  n1 = 0
  iup1 = 0
     iqea1 = 0
  ind11 = 0
endif
C
C------ Departure time at intersection 3 ------
  C
jwh = jwh3
if(j1.eq.4) icx = 28
```

```
      call DPARTA(k31, ig1(3), ir1(3), zz, dep31, m,
     &  icl, arr31, ng1, ncy31, jac, ixt3, jwh, ipt,
     &  tuz, dtuz, nux3, dtux3, jdum, dy3, jvar,
     &  icx)
      icx = 0
      mprod = 1
      call UPDNKA(k31, ig1(3), ir1(3), zz, dep31, m,
     &  jdq, lgi, icl, arr31, ng1, nql3i, 0, ncy31,
     &  zsut, iveh(1), nka2, isp2, suts2,  mprod,
     &  jac)

      if(ixt3.eq.1) jor2 = 0

      if(ixt3.eq.1) jxt3 = 1
      if(ixt3.eq.1) go to 210
      251 if(ixt3.eq.1) then
        ixt3 = 0
           if(j1.eq.4) then
            krx2 = 0
            klx3 = 0
           endif
        tuz5 = tuz(5)



        if(j1.eq.4.and.tuz5.gt.ig1(3).and.ig1(3)+icl
        .lt.irr(2))then
          tuz5 = wuz(5, ibs5)
          if(dep31(m).gt.wuz(5,
        ibs5).and.dep31(m).lt.wuz(5, ibs5+1))
        then
           ibs5 = ibs5 + 1
           tuz5 = wuz(5, ibs5)
         endif
         if(tuz5.gt.ig1(3)+icl) go to 230
         kibs5 = 1
        endif


        if(j1.eq.4.and.(tuz5.gt.ig1(3).or.tuz(6).gt.
        ig1(3))) then
          jz = 5
          if(tuz5.lt.tuz(6)) jz = 6
          if(jz.eq.5) then
            ibs5 = ibs5 + 1
            kibs5 = 0
          endif
          dep31(m) = tuz5 + 2.04
          if(jz.eq.6) dep31(m) = tuz(6) + 2.04
          ircl1 = ir1(3)+icl-ipt
          if(dep31(m).ge.ircl1) then
            call UPDATE(ng1, k31, ig1(3), ir1(3), zz,
        dep31, m, icl, ncy31)
           ixt3 = 1
           krx2 = 1
           klx3 = 1
           krt3 = 1
           go to 210
         endif
        endif
      endif

      230 if(j1.eq.4) then
      if((nac.eq.1.and.(k31.eq.2.or.jdq.eq.11.and.
        (k31.eq.4.or.k31.eq.5))).or.(nac.eq.2.and.
        (k31.eq.2.or.k31.eq.3.or.jdq.eq.11.and.(k31.
        eq.5.or.k31.eq.6.or.k31.eq.8.or.k31.eq.9))))
        then
          jo2 = jo2 + 1
          carrl3(jo2) = dep31(m)
          if(mqc2.eq.iveh(1).or.jor2+mqc2.eq.iveh(1))
          then
            sss = st(6)
            ssd = sss - dep31(m)
            if(sss.ge.ir1(3)+icl) then
              imq3 = 1
            elseif(ssd.gt.0) then
              nux3 = 1
              dtux3 = ssd
            endif
          endif
          jor2 = jor2 + 1
        endif
        endif
        C
        C------ Print arrival and departure time at
          each intersection ---------
        C
        255 if(nl.ne.0) then
          parr11 = arr11(m)
          pdep11 = dep11(m)
          parr21 = arr21(m)
          pdep21 = dep21(m)
          arr11(m) = 0.
          dep11(m) = 0.
          arr21(m) = 0.
          dep21(m) = 0.
        endif
        ki21p = ii + jo1
        ki31p = jo1
        if(dep31(m).lt.arr31(m)) dep31(m) = arr31(m)
        C
        if(mult.eq.0) then
        call PRN(arr11, dep11, arr21, dep21, arr31,
          dep31, ku2, m, ii,  jo1, nql2, nql3, nql3i)
        endif
        if(ki21p.ne.ki21(m)) go to 261
        do 260 ik = isp1,  ncy21
          nka1(ik) = nka1(ik) + 1
        260 continue
        261 continue

        if(ki31p.ne.ki31(m)) go to 266
        do 265 ik = isp2,  ncy31
          nka2(ik) = nka2(ik) + 1
        265 continue
        266 continue

        if(nl.ne.0) then
          arr11(m) = parr11
          dep11(m) = pdep11
          arr21(m) = parr21
          dep21(m) = pdep21
        endif

        if(imq2.eq.1) go to 210
        if(imq3.eq.1) go to 210
        200 continue
        210ma(j1) = m
        if(j1.eq.4.and.m.ge.176) then
        endif
        if(ig1(2).ge.ismt) jvol = jvol - 1
        if(imq2.eq.1.or.imq3.eq.1) then
          ma(j1) = m + 1
          if(ig1(2).ge.ismt) jvol = jvol + 1
        endif
```

```
return
end
*******
subroutine findep (dep, mx1)
dimension dep(0:2000)
100 if(dep(mx1).eq.0) then
mx1 = mx1 - 1
if(mx1.le.0) then
  mx1 = 0
  go to 200
endif
go to 100
endif
200return
end
*******
subroutine NOVSIM(dep, mx1, time, novs)
dimension dep(0:2000)
if(mx1.eq.0) then
  novs = 0
  go to 200
endif
do 100 ij = mx1,  1,  -1
if(time.gt.dep(ij).and.dep(ij).gt.0.1) then
  novs = ij
  go to 200
endif
if(dep(ij).gt.time.and.dep(ij-1).le.time.and.
  dep(ij-1).gt.0.1) then
  novs = ij - 1
  if(ij.eq.0) go to 200
  ijs = ij
250  if(dep(ijs).lt.0.01) then
    ijs = ijs - 1
    novs = ijs - 1
    go to 250
  endif
  go to 200
endif
100 continue
200return
end
*******
subroutine NOVWUP(dep1, dep2, dep3, iwp, novs,
  mx1)
dimension dep1(0:2000), dep2(0:2000),
  dep3(0:2000)
if(mx1.eq.0) then
novs = 0
go to 200
endif
do 100 ij = 1,  600
  if(dep1(ij).lt.0.01.and.dep2(ij).lt.0.01.and.
  dep3(ij).lt.0.01) then
    novs = ij - 2
    go to 200
  endif
  if(dep2(ij).gt.iwp) then
    novs = ij - 1
    go to 200
  endif
100 continue
200return
end
*******
subroutine piksrt(n, dep)
dimension dep(10)
do 12 j = 2,  n
  a = dep(j)
  do 11 i = j-1,  1,  -1
    if(dep(i).le.a) go to 10
    dep(i+1) = dep(i)
11  continue
  i = 0
10  dep(i+1) = a
12 continue
return
end
*******
subroutine ARRQLA(arr, dep, depp, m, nce, dist,
  speed, at, arrr, j, n, nnn, iup, ki, nq,
  ind1, iqt, qst, iqe, iqea, iveh1, icx, nka,
  isp, suts, nii, iqti, qsti, niq, dtiq, nqst,
  rqst, jdq, arrrmp, icw, jac, jdum, jvar)
dimension dep(0:2000), depp(0:2000),
  arr(0:2000)
dimension arrr(0:500), ki(0:2000), qst(80),
  qsti(80)
dimension nka(100), suts(0:100, 0:60), rqst(80)
wid = 40.
if(jac.eq.2) wid = 60.
arrrmp = arrr(n)
if(arr(m-1).eq.dep(m-1).and.arr(m-1).ne.0.) nce
  = 0
ipo = 1
call NOQUE(arr, dep, 0, arrr, iveh1, ki, m, m1,
  nq, n, iqe, iqea, ipo, 0)
ipo = 0

call FINDEV(m, dep, iveh1, lks, st1)
jsubs = iveh1 - 1
call FINDEV(m, dep, jsubs, lks, st2)

rlo = nq/(dist/20)
if(m.eq.1) rlo = 0.0
if(rlo.gt.1.0) rlo = 1.0

if(jvar.eq.1) then
call rspeed(jdum, rlo, speed, icx)
else
  if(rlo.gt.0.001) then
    speed = 23.365 - 9.0025*rlo
  else
    speed = 30.9
  endif
endif
IF(nq.ge.iveh1+1.and.arr(m-1).eq.arrr(n-1))
  then
  iqt = iqt + 1
  qst(iqt) = arrr(n-1)
  dids = abs(depp(m)-st1)
  if(dids.lt.0.001) then
    iqt = iqt - 1
  elseif(depp(m).lt.st1) then
    niq = 1
    nqst = nqst + 1
    iqt = iqt - 1
    rqst(nqst) = arrr(n-1)
    dtiq = st1 - depp(m)
    depp(m) = st1
  endif

  if(arrr(n).eq.0.) go to 110
  iqea = 1
  if(arrr(n).lt.arr(m-1)) iqea = 0
```

```
   call NOQUE(arr, dep, 0, arrr, iveh1, ki, m,
   m1, nq, n, iqe, iqea,    ipo, 0)
   if(nq.ge.iveh1+1) then
     iqe = 1
     arrr(n) = st1 + 1.0
     if(arrr(n).ge.depp(m) ) then
       call NEXTTI(n, arrr, iup, nnn, iqe, nii,
   niq)
       go to 150
     endif
   endif
ENDIF

110 if(m.ge.2) nce = nq

at = depp(m) + (dist+wid-20.*nce)/speed
if(20.*nce.gt.dist) at = depp(m) + wid/speed

intv = int(iveh1*0.3)
nisp = nka(isp)
nispt = nka(isp+1)
IF(nce.ne.0) then

   atx = depp(m) + (wid+dist-(nisp-1)*20.)/speed
   atxt = depp(m) + (wid+dist-(nispt-
   1)*20.)/speed

   if(nisp.le.iveh1) then
    if(arr(m-1).ne.0.0) then
     if(arr(m-1).ne.suts(isp, nka(isp)-1).and.
   atx.lt.suts(isp, nka(isp)-1).and.nisp.gt.nq)
   then
at = atx
     elseif(arr(m-1).eq.suts(isp, nka(isp)-
   1).and.     atxt.lt.suts(isp+1, nka(isp+1)-
   1).and.     arr(m-1).ne.suts(isp+1,
   nka(isp+1)-1)) then
at = atxt
     endif
    endif
    if(arr(m-1).eq.0.0.and.nisp.gt.nq) then
     if(arr(m-2).ne.suts(isp, nka(isp)-1)) then
at = atx
     endif
    endif
   endif

   if(nisp.eq.iveh1+1.and.depp(m).lt.suts(isp,
   nka(isp)-1).   and.nq.ge.intv) then
    if(arr(m-1).ne.0.0.and.arr(m-1).ne.suts(isp,
   nka(isp)-1))    then
       at = depp(m) + wid/speed
    elseif(arr(m-1).eq.0.0.and.arr(m-2).ne.
   suts(isp, nka(isp)-1)) then
       at = depp(m) + wid/speed
    endif
   endif

ENDIF

if(niq.eq.1.and.arrr(n).ge.depp(m)) go to 130

if(niq.eq.1.and.arrr(n).ne.0.0) then
   call NEXTTI(n, arrr, iup, nnn, iqe, nii, niq)
   go to 150
endif

130IF(arrr(n).lt.depp(m).and.arrr(n).ne.0.)
   then

   if(nq.ge.iveh1) then
     iqea = 1
     if(arrr(n).lt.arr(m-1)) iqea = 0
     call NOQUE(arr, dep, 0, arrr, iveh1, ki, m,
   m1, nq, n, iqe, iqea,      ipo, 0)
     if(nq.gt.0.and.arrr(n).gt.arr(m-1)) nq = nq
   - 1
   endif

rlo = nq/(dist/20)
if(rlo.gt.1.0) rlo = 1.0
if(jvar.eq.1) then
call rspeed(jdum, rlo, speed, icx)
else
   if(rlo.gt.0.001) then
     speed = 23.365 - 9.0025*rlo
   else
     speed = 30.9
   endif
endif

   if(20.*nq.gt.dist) then
     arrr(n) = arrrmp + wid/speed
   else
     arrr(n) = arrrmp + (dist+wid-20.*nq)/speed
    endif
   if(nce.ne.0.and.nisp.gt.nq) then
     att = arrrmp + (wid+dist-(nisp-
   1)*20.)/speed
     if(nisp.le.iveh1.and.att.lt.suts(isp,
   nka(isp)-1).and.    arr(m-1).ne.suts(isp,
   nka(isp)-1)) then
       if(att.gt.arr(m-1)) then
         arrr(n) = att
       endif
     endif
   endif
   if(nisp.eq.iveh1+1.and.arrrmp.lt.suts(isp,
   nka(isp)-1).   and.nq.ge.intv.and.arr(m-
   1).ne.suts(isp, nka(isp)-1)) then
        arrr(n) = arrrmp + wid/speed
   endif
   if(arrr(n).lt.arrr(n-1).and.nnn.eq.0) then
     arrr(n) = arrr(n-1) + 1.0
   endif
   tt = arrrmp + (dist+wid)/30.9
   if(nnn.eq.0.and.tt.gt.dep(m-1).and.dep(m-
   1).gt.0.1) then
     arrr(n) = tt
   endif

ELSE
   go to 150
ENDIF

if(nii.eq.1.and.arrr(n).lt.st1) then
   arrr(n) = st1
   if(arrr(n).ge.depp(m) ) then
     call NEXTTI(n, arrr, iup, nnn, iqe, nii,
   niq)
     go to 150
   endif
   go to 151
endif
*@
```

```
if(iqe.eq.1) then
  arrr(n) = arrr(n-1) + 1.0
  iqe = 0
  if(arrr(n).ge.depp(m) ) then
    call NEXTTI(n, arrr, iup, nnn, iqe, nii,
  niq)
    go to 150
  endif
endif

151 if(arrr(n).lt.arr(m-1).and.arrr(n).lt.at)
  then
  arrr(n) = arrr(n-1) + 1.0
  if(arrr(n).lt.arr(m-1).and.arrr(n).lt.at)
  then
    call NEXTTI(n, arrr, iup, nnn, iqe, nii,
  niq)
  endif
endif
***
iqea = 1
if(arrr(n).lt.arr(m-1)) iqea = 0
call NOQUE(arr, dep, 0, arrr, iveh1, ki, m, m1,
  nq, n, iqe, iqea, ipo, 0)
if(nq.gt.0.and.arrr(n).gt.arr(m-1)) nq = nq - 1
if(m.ge.2) nce = nq
at = depp(m) + (dist+wid-20.*nce)/speed
if(20.*nce.gt.dist) at = depp(m) + wid/speed
***
if(at.lt.arr(m-1)+1.0) at = arr(m-1) + 1.0

IF(at.ge.arrr(n).and.arr(m-1).le.arrr(n)) then

  if(nka(isp).eq.iveh1+1.and.arrr(n).lt.st2-1.1
  .and.arr(m-1).ne.suts(isp, nka(isp)-1)) then
    if(nq.eq.iveh1+1) go to 160
    if(arrr(n).lt.dep(m-lks+1)) go to 160
    if(suts(isp, nka(isp))-arrr(n).lt.jdq) go
  to 170
    iqti = iqti + 1
    qsti(iqti) = arrr(n)
170    nii = 1
160    if(depp(m).lt.st2.and.arr(m-1).ne.
  suts(isp, nka(isp)-1)) then
      if(suts(isp, nka(isp))-arrr(n).lt.jdq) go
  to 190
      nqst = nqst + 1
      if(iqti.gt.0) iqti = iqti - 1
      rqst(nqst) = arrr(n)
190      niq = 1
      dtiq = suts(isp, nka(isp)) - depp(m)
      depp(m) = suts(isp, nka(isp))
    endif
  endif

  if(nka(isp).eq.iveh1+2.and.arrr(n).lt.st2-1.1
  .and.arr(m-1).ne.st2-1.1) then
    arrr(n) = st2-1.1

    if(st2-1.1.eq.depp(m))then
      call NEXTTI(n, arrr, iup, nnn, iqe, nii,
  niq)
      go to 150
    endif
    nii = 1
    if(arrr(n).ge.depp(m)) then
    call NEXTTI(n, arrr, iup, nnn, iqe, nii,
  niq)
      go to 150
      endif
    endif

    nnn = nnn + 1
    j = j + 1
    nce = nce + 1
    at = depp(m) + (dist+wid-20.*nce)/speed
    if(20.*nce.gt.dist) at = depp(m) + wid/speed

    if(nq.ge.iveh1+1) then
      iqea = 1
      call NOQUE(arr, dep, 0, arrr, iveh1, ki, m,
    m1, nq, n, iqe, iqea,      ipo, 0)
      if(nq.ge.iveh1+1) then
        iqe = 1
        arrr(n) = st1 + 1.0
        if(arrr(n).ge.depp(m)) then
   nnn = nnn - 1
   j = j - 1
   nce = nce - 1
   call NEXTTI(n, arrr, iup, nnn, iqe, nii, niq)
   go to 150
       endif
      endif
    endif
    if(nnn.eq.1) then
      arr(m+1) = at
    else
      arr(m+1) = arr(m)
    endif
    arr(m)   = arrr(n)

    n = n + 1
    diff = arrr(n) - arrrmp
    if(diff.gt.25.0) iup = 1
    if(arrr(n).eq.0.) iup = 1
ELSE
150  arr(m) = at
  nii = 0
  nce = nce + 1
ENDIF

if(arr(m).le.arr(m-1).and.arr(m).gt.0.) then
  arr(m) = arr(m-1) + 1.0
endif

m10 = m - 1
call findep(arr, m10)
if(arr(m).lt.arr(m10)+0.5) then
  arr(m) = arr(m10) + 0.5
endif
C****
ipo = 10
call NOQUE(arr, dep, 0, arrr, iveh1, ki, m, m1,
  nq, n, iqe, iqea, ipo, 0)
ipo = 0

return
end
*******
subroutine NOQUE(arr, dep, depp, arrr, iveh1,
  ki, m, m1, nq, n, iqe, iqea, ipo, jgt)
dimension arr(0:2000), dep(0:2000),
  depp(0:2000)
dimension ki(0:2000), arrr(0:500)
ki(0) = 0
if(m.gt.1) then
```

```
      m1 = m - 1
      time = arr(m1)
      if(ipo.eq.19) time = depp(m)
      na = m1
      if(iqe.eq.1.or.iqea.eq.1) then
        if(ipo.eq.0) then
          time = arrr(n)
          na = m
        endif
      endif
  110 if(arr(m1).eq.0) then
        m1 = m1 - 1
        if(ipo.ne.19) time = arr(m1)
        na = m1
        if(ipo.eq.9.and.nq.ge.iveh1+1) then
          jgt = 1
          go to 100
        endif
        go to 110
      endif
      if(ipo.eq.10) then
        time = arr(m)
        na = m
      endif
      if(ipo.eq.29) then
        time = dep(m)
        na = m - 1
      endif
      if(dep(m1).le.time.and.dep(m1).ne.0) then
        nd = m1
        go to 200
      endif
      do 300 ij = 0,  m1-1
      if(dep(ij+1).gt.time.and.dep(ij).le.time)
      then
        nd = ij
        if(ij.eq.0) go to 200
        ijs = ij
  250   if(dep(ijs).eq.0.) then
          ijs = ijs - 1
          nd = ijs
          go to 250
        endif
      go to 200
      endif
  300 continue
  200 nq = na - ki(na) - (nd - ki(nd))
  100 continue
      endif
      return
      end
      *******
      subroutine NEXTTI(n, arrr, iup, nnn, iqe, nii,
        niq)
      dimension arrr(0:500)
      iloop = 0
      100np = n
      200n = n + 1
      iloop = iloop + 1
      if(iloop.gt.30) then
        write(10, *)'stopped by endless go to (in
        NEXTTI)', 'n, np,    arrr(n), arrr(np)', n,
        np, arrr(n), arrr(np)
        stop
      endif
      if(arrr(n).eq.0.0) go to 300
      if(arrr(n).lt.arrr(np)) go to 200
      diff = arrr(n) - arrr(np)
      if(diff.lt.25.) go to 100
      300iup = 1
      nnn = 0
      iqe = 0
      nii = 0
      niq = niq
      return
      end
      *******
      subroutine DEPART(k, ig, ir, z, dep, m, icl,
        arr, nce, ncy, jac, ixt,  jwh, ipt, jdum,
        dphy, jvar)
      dimension z(0:100),  dep(0:2000), arr(0:2000)
      dimension zsut(0:100), dphy(0:60)
      jpd3 = (jwh-15)*(jwh-25)
      if(jpd3.ne.0) jpd3 = 1
      k = k + 1
      call DPHDWY(z, zsut)
      if(jvar.eq.1.and.k.eq.1.or.k.eq.2) then
      call nordev(jdum, dphy, icx)
      endif

      ircl = ir + icl*(2-jac) - ipt*(1-jpd3)
      if(arr(m).le.ircl) then
        if(jvar.eq.0) then
          dep(m) = z(k) + ig
        else
          dep(m) = dphy(k) + ig
        endif
        if(dep(m).gt.ircl)then
          call UPDATE(nce, k, ig, ir, z, dep, m, icl,
        ncy)
          ixt = 1
        endif
        if(arr(m).ge.dep(m)) dep(m) = arr(m)
      elseif(arr(m).gt.ircl) then
        call UPDATE(nce, k, ig, ir, z, dep, m, icl,
        ncy)
        ixt = 1
        if(arr(m).ge.dep(m)) dep(m) = arr(m)
      endif

      m10 = m - 1
      call findep(dep, m10)
      if(dep(m).lt.dep(m10)+0.7) then
        dep(m) = dep(m10) + 0.7
      endif

      ircl = ir + icl*(2-jac) - ipt*(1-jpd3)
      if(k.gt.2.and.arr(m-1).eq.dep(m-1).and.arr(m-
        1).gt.1.and.
        arr(m).gt.ig.and.arr(m).le.ircl) then
        dep(m) = arr(m)
      endif

      if(dep(m).gt.ircl) then
        call UPDATE(nce, k, ig, ir, z, dep, m, icl,
        ncy)
        ixt = 1
      endif
      return
      end
      *******
      subroutine DPARTA(k, ig, ir, z, dep, m, icl,
        arr, nce, ncy, jac, ixt, jwh, ipt, tuz,
        dtuz, nux, dtux, jdum, dphy, jvar, icx)
      dimension z(0:100), dep(0:2000), arr(0:2000)
```

```
dimension zsut(0:100), tuz(15), dtuz(15),
  dphy(0:60)
jpd3 = (jwh-15)*(jwh-25)
if(jpd3.ne.0) jpd3 = 1
k = k + 1
call DPHDWY(z, zsut)
if(jvar.eq.1.and.k.eq.1.or.k.eq.2) then
call nordev(jdum, dphy, icx)
endif
ircl = ir + icl*(2-jac) - ipt*(1-jpd3)
if(arr(m).le.ircl) then
  if(jvar.eq.0) then
    dep(m) = z(k) + ig
  else
    dep(m) = dphy(k) + ig
  endif
  jz2 = 6
  if(tuz(5).gt.tuz(6)) jz2 = 5
  if(nux.eq.1) dep(m) = dep(m) + dtux
   if(tuz(jz2).gt.ig.and.jwh.eq.25) then
    dep(m) = dep(m) + dtuz(jz2)
  endif

  if(dep(m).gt.ircl)then
    call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
    nux = 0
    ixt = 1
  endif
  if(arr(m).ge.dep(m)) dep(m) = arr(m)
elseif(arr(m).gt.ircl) then
  call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
  nux = 0
  ixt = 1
  if(arr(m).ge.dep(m)) dep(m) = arr(m)
endif

m10 = m - 1
call findep(dep, m10)
if(dep(m).lt.dep(m10)+0.7) then
  dep(m) = dep(m10) + 0.7
endif

ircl = ir + icl*(2-jac) - ipt*(1-jpd3)
if(k.gt.2.and.arr(m-1).eq.dep(m-1).and.arr(m-
  1).gt.1.and.
  arr(m).gt.ig.and.arr(m).le.ircl) then
  dep(m) = arr(m)
endif

if(dep(m).gt.ircl) then
  call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
  nux = 0
  ixt = 1
endif
return
end
*******
subroutine UPDNKA(k, ig, ir, z, dep, m, jdq,
  lgi, icl, arr, nce, nq, nqd, ncy, zsut,
  iveh1, nka, isp, suts, mprod, jac)
dimension z(0:100),  dep(0:2000), arr(0:2000)
dimension zsut(0:100)
dimension nka(100), suts(0:100, 0:60)
ircl = ir+icl*(2-jac)
if(nqd.ge.iveh1+1) go to 100
if(mprod.eq.0) go to 100
if(nq.eq.0.and.arr(m).gt.ig.and.arr(m).le.ircl)
  then
  dep(m) = arr(m)
endif
100intv = int(iveh1*0.3)

if(suts(isp, nka(isp)).lt.arr(m)) then
  suts(isp, nka(isp)) = arr(m)
endif
if(suts(isp, nka(isp)).gt.dep(m)) then
  suts(isp, nka(isp)) = dep(m)
endif
if(jac.eq.2) then
endif
if(nka(isp).ge.intv.and.arr(m).gt.suts(isp,
  nka(isp)-1))then
  suts(isp, nka(isp)) = arr(m)
endif

if(m.gt.1.and.isp.le.2) then
  if(dep(m-1).ne.0.and.arr(m-1).ne.0.and.dep(m-
  1).eq.  arr(m-1).and.dep(m).ne.arr(m)) then
    nka(isp) = 0
    isp = isp + 1
  endif
endif
if((jdq.eq.7.and.lgi.ge.30).or.(jdq.eq.9.and.lg
  i.ge.50).or.(jdq.eq.11.and.lgi.ge.70)) then
  isp = ncy
else
  if(nka(isp).ge.iveh1+6) then
    nka(isp) = 0
    isp = isp + 1
  endif
endif
if(m.le.6.and.ncy.gt.isp) isp = ncy

m10 = m - 1
call findep(dep, m10)
if(dep(m).lt.dep(m10)+0.7) then
  dep(m) = dep(m10) + 0.7
endif

if(dep(m).gt.ircl) then
  call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
endif
return
end
*******
subroutine UPNKA(k, ig, ir, z, dep, m, jdq,
  lgi, icl, arr, nce, nq, nqd, ncy, zsut,
  iveh1, nka, isp, suts, mprod, jac, tuz,
  dtuz, nibi, nibj)
dimension z(0:100), dep(0:2000), arr(0:2000)
dimension zsut(0:100), tuz(15), dtuz(15)
dimension nka(100), suts(0:100, 0:60)
ircl = ir+icl*(2-jac)
if(nqd.ge.iveh1+1) go to 100
if(mprod.eq.0) go to 100
if(nq.eq.0.and.arr(m).gt.ig.and.arr(m).le.ircl)
  then
  dep(m) = arr(m)
endif
100intv = int(iveh1*0.3)

if(suts(isp, nka(isp)).lt.arr(m)) then
```

```
  suts(isp, nka(isp)) = arr(m)
endif
if(suts(isp, nka(isp)).gt.dep(m)) then
  suts(isp, nka(isp)) = dep(m)
endif
if(nka(isp).ge.intv.and.arr(m).gt.suts(isp,
  nka(isp)-1))then
  suts(isp, nka(isp)) = arr(m)
endif

if(m.gt.1.and.isp.le.2) then
  if(dep(m-1).ne.0.and.arr(m-1).ne.0.and.dep(m-
  1).eq.  arr(m-1).and.dep(m).ne.arr(m)) then
    nka(isp) = 0
    isp = isp + 1
  endif
endif

if(tuz(8).gt.ig) dtuz(8) = tuz(8) - ig
if(tuz(8).gt.ig.and.lgi.ge.40.and.dtuz(8).gt.lg
  i-20.and. nibi.ne.1) then
  nibi = 1
endif

if(nibi.ne.1.and.nibj.ne.1) then
  if((jdq.eq.7.and.lgi.ge.30).or.(jdq.eq.9.and.
  lgi.ge.50).or.(jdq.eq.11.and.lgi.ge.70))
  then
    isp = ncy
  else
    if(nka(isp).ge.iveh1+6) then
      nka(isp) = 0
      isp = isp + 1
    endif
  endif
elseif(nibi.eq.1.or.nibj.eq.1) then
  if(nka(isp).ge.iveh1+3) then
    nka(isp) = 0
    isp = isp + 1
  endif
endif
if(m.le.6.and.ncy.gt.isp) isp = ncy
if(dep(m).gt.ircl) then
  call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
endif
return
end
*******
subroutine DPHDWY(z, zsut)
dimension z(0:100),  zsut(0:100)
z(0) = 0.
z(1) = 2.04
z(2) = 4.50
z(3) = 6.62
do 100 n = 4,  100
   z(n) = 1.34 + 1.82 * n
100 continue
zsut(0) = 0.
zsut(1) = 2.04
zsut(2) = 2.75
zsut(3) = 3.46
do 200 n = 4,  100
   zsut(n) = 1.1 * n
200 continue
return
end
*******
subroutine ARRNT(arr, dep, depp, m, nce, dist,
  speed, nq, jac,  jdum, jvar)
dimension dep(0:2000), depp(0:2000),
  arr(0:2000)
wid = 40.
if(jac.eq.2) wid = 60.
if(arr(m-1).eq.dep(m-1).and.arr(m-1).ne.0.) nce
  = 0

if(m.ge.2) then
  nce = nq
endif
rlo = nq/(dist/20)
if(rlo.gt.1.0) rlo = 1.0
if(jvar.eq.1) then
call rspeed(jdum, rlo, speed, icx)
else
  if(rlo.gt.0.001) then
    speed = 23.365 - 9.0025*rlo
  else
    speed = 30.9
  endif
endif

arr(m) = depp(m) + (dist+wid-20.* nce)/speed
if(20.*nce.gt.dist) arr(m) = depp(m) +
  wid/speed
nce = nce + 1
if(arr(m).le.arr(m-1).and.arr(m).gt.0.) then
  arr(m) = arr(m-1) + 1.0
endif

m10 = m - 1
call findep(arr, m10)
if(arr(m).lt.arr(m10)+0.5) then
  arr(m) = arr(m10) + 0.5
endif
return
end
*******
subroutine DEPQLS(k, ig, ir, z, dep, m, icl,
  arr, nce, arrd, depd, iq, qs, ki, nq, ncy,
  iveh1, zsut, nka, isp, nua, dtua, suts, iqs,
  qsi, nqs, rqs, niq, dtiq, jdq, nux, dtux,
  icx, jwh, jac, ixt, tuz, dtuz, wuz, ib, ijk,
  jz1, jz2, jz3, st, ipt, jdum, dphy, jvar)
dimension dep(0:2000), arr(0:2000),
  depd(0:2000), arrd(0:2000)
dimension ki(0:2000), suts(0:100, 0:60),
  nka(100)
dimension z(0:100), zsut(0:100), dphy(0:60)
dimension qs(80), qsi(80), rqs(80)
dimension tuz(15), dtuz(15), wuz(15, 80),
  ib(15), st(15)
jpd1 = (jwh-10)*(jwh-13)*(jwh-21)*(jwh-
  24)*(jwh-30) *(jwh-32)*(jwh-40)*(jwh-
  42)*(jwh-50)*(jwh-52)
jpd2 = (jwh-11)*(jwh-23)*(jwh-32)*(jwh-
  40)*(jwh-42) *(jwh-50)
jpd3 = (jwh-11)*(jwh-21)*(jwh-13)*(jwh-23)
if(jpd3.ne.0) jpd3 = 1
if(m.eq.1) then
  nua = 0
  ki(0) = 0
endif
k = k + 1
call DPHDWY(z, zsut)
if(jvar.eq.1.and.k.eq.1.or.k.eq.2) then
```

```
call nordev(jdum, dphy, icx)
endif
jgt = 0
ipo = 9
call NOQUE(arrd, depd, 0, 0, iveh1, ki, m, m1,
  nq, 0, 0, 0, ipo, jgt)
call FINDEV(m, depd, iveh1, lks, st11)
call FINDEX(m, depd, iveh1, lks, st, jz1)
call FINDEX(m, depd, iveh1+1, lks, st1, jz1)

   jib = 0
if(jgt.eq.1) go to 290
ircl = ir + icl*(2-jac) - ipt*(1-jpd3)
irclp = ir + icl*(2-jac)

if(m.gt.iveh1+1.and.nq.le.iveh1.and.arrd(m-
  1).ne.0..and. st11.gt.arrd(m-1)+4.)then
  dep(m) = st11
  if(st11.gt.ircl) then
    if(jib.eq.0.and.jpd1.eq.0) then
      jib = 1
      tuz(jz1) = st11
      dtuz(jz1) = tuz(jz1) - irclp
      wuz(jz1, ib(jz1)) = tuz(jz1)
      ib(jz1) = ib(jz1) + 1
    endif
      igcl = ig + icl
      if(dep(m).gt.igcl) then
 call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
 call UPDAT1(nux, niq, nua)
       dtux = st11 - ig - 2.05
       nux = 1
 ixt = 1
       dep(m) = st11
       go to 500
      endif
    call UPDATE(nce, k, ig, ir, z, dep, M, icl,
  ncy)
    call UPDAT1(nux, niq, nua)
    ixt = 1
    go to 500
  endif
  nua = 1
  dtua = st11 - arrd(m-1)
  go to 490
endif

IF(arr(m).le.ircl) then
  if(nq.ge.iveh1+1) then
    if(jvar.eq.0) then
      depm = z(k) + ig
    else
      depm = dphy(k) + ig
    endif
    dep(m) = st11
    iq = iq + 1
    qs(iq) = arrd(m1)
    if(st11.gt.ircl) then
      if(jib.eq.0.and.jpd1.eq.0) then
 jib = 1
 tuz(jz1) = st11
 dtuz(jz1) = tuz(jz1) - irclp
       wuz(jz1, ib(jz1)) = tuz(jz1)
       ib(jz1) = ib(jz1) + 1
      endif
      nqs = nqs + 1
      rqs(nqs) = arrd(m1)
      iq = iq - 1
      igcl = ig + icl
      if(dep(m).gt.igcl) then
 call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
 call UPDAT1(nux, niq, nua)
       dtux = st11 - ig - 2.05
 if(dtux.gt.icl) then
260   call UPDATE(nce, k, ig, ir, z, dep, m,
  icl, ncy)
   dtux = dtux - icl
   if(dtux.gt.icl) go to 260
 endif
       nux = 1
 ixt = 1
       dep(m) = st11
       go to 500
      endif
      call UPDATE(nce, k, ig, ir, z, dep, M,
  icl, ncy)
      call UPDAT1(nux, niq, nua)
      ixt = 1
      go to 389
    endif
    nux = 1
    dtux = dep(m) - depm
    go to 500
  endif
290  if(jvar.eq.0) then
    dep(m) = z(k) + ig
  else
    dep(m) = dphy(k) + ig
  endif

  if(niq.eq.1) dep(m) = dep(m) + dtiq
  if(nux.eq.1) dep(m) = dep(m) + dtux
   if(tuz(jz2).gt.ig.and.jpd2.eq.0) then
    if(ijk.eq.1.and.ncy.eq.1) go to 389
    if(nux.eq.1.and.tuz(jz2).lt.st11)        go
  to 389
    dep(m) = dep(m) + dtuz(jz2)
  endif
389  continue
ELSEIF(arr(m).gt.ircl) then
  if(nq.ge.iveh1+1) then
    iq = iq + 1
    qs(iq) = arrd(m1)
    if(st11.gt.ircl) then
      if(jib.eq.0.and.jpd1.eq.0) then
 jib = 1
 tuz(jz1) = st11
 dtuz(jz1) = tuz(jz1) - irclp
       wuz(jz1, ib(jz1)) = tuz(jz1)
       ib(jz1) = ib(jz1) + 1
      endif
      nqs = nqs + 1
      rqs(nqs) = arrd(m1)
      iq = iq - 1
      igcl = ig + icl
      if(st11.gt.igcl) then
 call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
 call UPDAT1(nux, niq, nua)
       dtux = st11 - ig - 2.05
 if(dtux.gt.icl) then
   call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
   dtux = dtux - icl
```

```
 endif
       nux = 1
 ixt = 1
       dep(m) = st11
       go to 500
     endif
   endif
  endif
  call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
  call UPDAT1(nux, niq, nua)
  ixt = 1
ENDIF
if(nua.eq.1) dep(m) = dep(m) + dtua
ircl = ir + icl*(2-jac) - ipt*(1-jpd3)
irclp = ir + icl*(2-jac)
IF(nka(isp).eq.iveh1+2.and.nq.le.iveh1) then
if(m.ge.iveh1+1.and.arrd(m1).ne.suts(isp,
  nka(isp)-1)) then
  if(arrd(m1).lt.suts(isp, nka(isp)-2)) then
    if(arrd(m-1).eq.0.and.dep(m-1).ne.0) go to
  410
    if(suts(isp, nka(isp)-1) - arrd(m1).lt.jdq)
  go to 410
    iqs = iqs + 1
    qsi(iqs) = arrd(m1)
  endif
410  continue
  if(dep(m).lt.suts(isp, nka(isp)-1)) then
    nua = 1
    if(suts(isp, nka(isp)-1).gt.ircl) then
      if(jib.eq.0.and.jpd1.eq.0) then
 jib = 1
 tuz(jz1) = st11
 dtuz(jz1) = tuz(jz1) - irclp
       wuz(jz1, ib(jz1)) = tuz(jz1)
       ib(jz1) = ib(jz1) + 1
     endif
     if(suts(isp, nka(isp)-1) -
  arrd(m1).lt.jdq) go to 420
     nqs = nqs + 1
     rqs(nqs) = arrd(m1)
     if(iqs.gt.0) iqs = iqs - 1
     igcl = ig + icl
     if(st11.gt.igcl) then
 call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
 call UPDAT1(nux, niq, nua)
       dtux = st11 - ig - 2.05
       nux = 1
 ixt = 1
       dep(m) = st11
       go to 500
     endif
420      continue
    endif
    dtua = suts(isp, nka(isp)-1) - dep(m)
    dep(m) = suts(isp, nka(isp)-1)
  endif
endif
ENDIF

ircl = ir + icl*(2-jac) - ipt*(1-jpd3)
if(k.gt.2.and.arr(m-1).eq.dep(m-1).and.arr(m-
  1).gt.1.and.
  arr(m).gt.ig.and.arr(m).le.ircl) then
  dep(m) = arr(m)
endif

490 if(arr(m).ge.dep(m)) dep(m) = arr(m)
m10 = m - 1
call findep(dep, m10)
if(dep(m).lt.dep(m10)+0.7) then
  dep(m) = dep(m10) + 0.7
endif

if(dep(m).gt.ircl) then
  call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
  call UPDAT1(nux, niq, nua)
  ixt = 1
endif

igcl = ig + icl
ircl = ir + icl*(2-jac) - ipt*(1-jpd3)

if(jgt.eq.1.and.st11.gt.dep(m)) then
  if(st11.gt.ircl) then
    if(jib.eq.0.and.jpd1.eq.0) then
      jib = 1
      tuz(jz1) = st11
      dtuz(jz1) = tuz(jz1) - ircl
    endif
    if(dep(m).gt.igcl) then
      call UPDATE(nce, k, ig, ir, z, dep, m,
 icl, ncy)
      call UPDAT1(nux, niq, nua)
      dtux = st11 - ig - 2.05
      nux = 1
      ixt = 1
      dep(m) = st11
      go to 500
    endif
    call UPDATE(nce, k, ig, ir, z, dep, M, icl,
  ncy)
    call UPDAT1(nux, niq, nua)
    ixt = 1
    if(dep(m).lt.st11) dep(m) = st11
    go to 500
  endif
  dep(m) = st11
endif
if(jgt.eq.1.and.nq.ge.iveh1+1.and.st(jz1).gt.ig
  cl) then
  if(jib.eq.0.and.jpd1.eq.0) then
    tuz(jz1) = st(jz1)
    dtuz(jz1) = tuz(jz1) - ircl
  endif
endif

500 continue
return
end
*******
subroutine FINDEV(m, depd, iveh1, lks, st1)
dimension depd(0:2000)
lk1 = 0
if(m.gt.iveh1) then
do 100 lk = 1,  99
  if(depd(m-lk).eq.0.or.depd(m-lk).ne.0.and.
  depd(m-lk).eq.depd(m-lk-1)) then
    lk1 = lk1 + 1
  endif
  if(lk-lk1.eq.iveh1+1) then
    lks = lk
    go to 110
```

```
  elseif(m-lk.eq.1) then
    st1 = 0.
    go to 120
  endif
100 continue
110st1 = depd(m-lks) - 2.04 + (iveh1+1)*1.1
endif
120return
end
*******
subroutine FINDEX(m, depd, iveh1, lks, st, j1)
dimension depd(0:2000), st(15)
lk1 = 0
if(m.ge.iveh1) then
do 100 lk = 1,  99
  if(depd(m-lk).eq.0.or.depd(m-lk).ne.0.and.
  depd(m-lk).eq.depd(m-lk-1)) then
    lk1 = lk1 + 1
  endif
  if(lk-lk1.eq.iveh1) then
    lks = lk
    go to 110
  elseif(m-lk.eq.1) then
    st(j1) = 0.
    go to 120
  endif
100 continue
110st(j1) = depd(m-lks) - 2.04 + (iveh1)*1.1
endif
120return
end
*******
subroutine ARRNTE(arr, dep, depp, m, nce, dist,
  speed, nq, iveh1, nka, isp, suts, ki, jac,
  jdum, jvar)
dimension dep(0:2000), depp(0:2000),
  arr(0:2000)
dimension ki(0:2000), nka(100), suts(0:100,
  0:60)
wid = 40.
if(jac.eq.2) wid = 60.
if(nce.eq.0) then
  ipo = 19
  call NOQUE(arr, dep, depp, 0, iveh1, ki, m,
  0, nq, 0, 0, 0, ipo, 0)
endif
if(arr(m-1).eq.dep(m-1).and.arr(m-1).ne.0.) nce
  = 0
if(m.ge.2) then
  nce = nq
endif

rlo = nq/(dist/20)
if(rlo.gt.1.0) rlo = 1.0
if(jvar.eq.1) then
call rspeed(jdum, rlo, speed, icx)
else
  if(rlo.gt.0.001) then
    speed = 23.365 - 9.0025*rlo
  else
    speed = 30.9
  endif
endif

arr(m) = depp(m) + (dist+wid-20.*nce)/speed
if(20.*nce.gt.dist) arr(m) = depp(m) +
  wid/speed
intv = int(iveh1*0.3)
nisp = nka(isp)
if(nce.ne.0) then
  atx = depp(m) + (wid+dist-(nisp-1)*20.)/speed
  if(nisp.le.iveh1.and.atx.lt.suts(isp,
  nka(isp)-1).and.  arr(m-1).ne.suts(isp,
  nka(isp)-1)) then
    arr(m) = atx
  endif
endif
if(nisp.eq.iveh1+1.and.depp(m).lt.suts(isp,
  nka(isp)-1). and.nq.ge.intv.and.arr(m-
  1).ne.suts(isp, nka(isp)-1)) then
  arr(m) = depp(m) + wid/speed
endif
nce = nce + 1
if(arr(m).le.arr(m-1).and.arr(m).gt.0.) then
  arr(m) = arr(m-1) + 1.0
endif
m10 = m - 1
call findep(arr, m10)
if(arr(m).lt.arr(m10)+0.5) then
  arr(m) = arr(m10) + 0.5
endif
return
end
*******
subroutine CROSST(kc, ku3, ku4, tdc, carrr,
  darrr, arrra, arrrb, nvol, lgic, noff, icl,
  jdq, ip, jp, speed, ntg, mult, jvol, tuz,
  dtuz, wuz, ib, mq1, mq11, klt, krt, jwh1,
  jwh2, jwh3, jwh4, isps, nkas, suts, isps1,
  nkas1, suts1, dist, iveh, ismt, irs, igs,
  irs1, igs1, igf, kfa, ma, st, ipt, nca, j1,
  iqis2, qs2, iqss, qsi, nqss, rqs2, iqts,
  qst, iqtis, qsti, nqsts, rqst, ki2, ki3, j2,
  iqis21, qs21, iqss1, qsi1, nqss1, rqs21,
  iqts1, qst1, iqtis1, qsti1, nqsts1, rqst1,
  ki21, ki31, carr1, cdep1, carr2, cdep2,
  carr3, cdep3, darr1, ddep1, darr2, ddep2,
  darr3, ddep3, isp, nka, sutc, nqa, isp1,
  nka1, sutc1, nqa1, jdum, dy1, dy2, dy3, dy4,
  dy5, dy6, jvar, ig, igb1)
dimension carr1(0:2000), cdep1(0:2000),
  carr2(0:2000)
dimension cdep2(0:2000), carr3(0:2000),
  cdep3(0:2000)
dimension darr1(0:2000), ddep1(0:2000),
  darr2(0:2000)
dimension ddep2(0:2000), darr3(0:2000),
  ddep3(0:2000)
dimension carrr(0:500), darrr(0:500)
dimension arrra(0:500), arrrb(0:500)
dimension ki2(0:2000), ki3(0:2000),
  ki21(0:2000), ki31(0:2000)
dimension dist(20), iveh(20), noff(20)
dimension irs(20), igs(20), irs1(20), igs1(20),
  igf(20)
dimension ig(20), igb1(20)
dimension qs2(80), qst(80), qsti(80), rqst(80)
dimension qs21(80), qst1(80), qsti1(80),
  rqst1(80)
dimension zz(0:100), zsut(0:100)
dimension suts(0:100, 0:60), suts1(0:100, 0:60)
dimension nkas(100), nkas1(100), qsi(80),
  qsi1(80)
dimension jgn2(100), jgn3(100), rqs2(80),
  rqs21(80)
```

```
dimension nka(100), nka1(100), sutc(0:100,
  0:60)
dimension tuz(15), dtuz(15), wuz(15, 80),
  ib(15)
dimension kfa(10), ma(10), st(15), sutc1(0:100,
  0:60)
dimension dy1(0:60), dy2(0:60), dy3(0:60)
dimension dy4(0:60), dy5(0:60), dy6(0:60)
read(50)ixts, ncs, nes, ngs, ks1, ks2, ks3, is,
  nns, jos, nls, iups, iqis3, ncys1, ncys2,
  ncys3, mq2, mq3, inds1, iqes, iqeas, niis,
  niqs, nuas, nuxs, ixts1, ncs1, nes1, ngs1,
  ks11, ks21, ks31, is1, nns1, jos1, nrs,
  iups1, iqis31, ncys11, ncys21, ncys31, mq21,
  mq31, inds11, iqes1, iqeas1, niis1, niqs1
read(50)nuas1, nuxs1, ijk, ijk1, kfi, nui,
  nui1, imqs, imqs1, joc, joc1, nuzi, nuzi1
read(50)dtiqs, dtuas, dtuxs, cparr1, cpdep1,
  cparr2, cpdep2, dtiqs1, dtuas1, dtuxs1,
  dtui, dtui1, dtuzi, dtuzi1, dparr1, dpdep1,
  dparr2, dpdep2
rewind(50)
jac = 2
if(kfa(j1).eq.0) then
nui = 0
ixts = 0
joc = 0
imqs = 0
lric = icl - lgic
if(mult.eq.0) then
  write(ku3, 121) kc
121  format('   ****** CROSS STREET # ', i1, '
  ******')
  write(ku3, 122)
122  format('   *** Left lane of the cross
  street ***')
  call HEAD(ku3, icl, lgic, 0, lric, noff(2),
  noff(3))
endif
************************************************
******  Left lane of the cross street *********
************************************************
call CTIME(kc, igs, irs, noff, lgic, icl, ip,
  jp)
call INIT(carr1, cdep1, carr2, cdep2, carr3,
  cdep3, ncs, nes, ngs, ks1, ks2, ks3, is,
  nns, m)
call INIT1(jos, nls, iups, iqis2, iqis3, ncys1,
  ncys2, ncys3, mq1, mq2, mq3, inds1, iqts,
  iqes, iqeas, iqtis, nqsts)
call INIT2(niis, isps, iqss, nqss, niqs, dtiqs,
  nuas, dtuas, nuxs, dtuxs)
call SIGADJ(irs, igs, lric, lgic, icl, dist,
  speed, jac, kc,  carrr, jdum, jvar)
endif

if(kfa(j1).eq.0.or.kfa(j2).eq.0.and.kfi.ne.1)
  then
do 120 ij = 1,  100
  nkas(ij) = 1
  nkas1(ij) = 1
120 continue
ijk = 0
ijk1 = 0
if(igf(kc).gt.igs(2)) then
   ijk = 1
  ijk1 = 1
endif

jgn2(1) = igs(2)
jgn3(1) = igs(3)
do 135 ka = 2,  100
  jgn2(ka) = jgn2(ka-1) + icl
  jgn3(ka) = jgn3(ka-1) + icl
135 continue
call DPHDWY(zz, zsut)
do 145 im = 1,  100
  do 142 jm = 0,  60,  1
    suts(im, jm) = jgn3(im) + zsut(jm)
    suts1(im, jm) = jgn3(im) + zsut(jm)
142  continue
145 continue
if(kfa(j1).eq.0) kfa(j1) = 1
kfi = 1
endif

if(klt.ne.1.and.krt.eq.1) go to 1000

C------ Arrival time at intersection 1 -------
C
do 100 m = ma(j1),  ma(j1)+ntg
if(igs(2).ge.ismt) jvol = jvol + 1

if(ixts.eq.1) go to 150
ki2(m) = is
ki3(m) = jos

IF(nls.eq.0) then
    carr1(m) = 7.2 * (m-is)
C
C------ Departure time at intersection 1 ------
C
 call DEPTQ(ks1, igs(1), irs(1), zz, cdep1, m,
  icl, carr1, ncs, carr2, cdep2, ki2, mq2,
  iveh(1), ncys1, nui, dtui, jac, kc, jdum,
  dy1, jvar)
if(imqs.eq.1) then
  imqs = 0
  call UPDATE(nes, ks2, igs(2), irs(2), zz,
  cdep2, m, icl, ncys2)
  niqs = 0
  ks2 = ks2 - 1
  joc = 0
endif
C
C------ Arrival time at intersection 2 --------
C
call ARRNT(carr2, cdep2, cdep1, m, ncs,
  dist(1), speed, mq2, jac,  jdum, jvar)
C
C------ Departure time at intersection 2 ------
C
jwh = jwh1
jz2 = 12
if(kc.eq.2) jz2 = 2
if(kc.eq.3) jz2 = 3
if(kc.eq.1) icx = 82
call DEPQLS(ks2, igs(2), irs(2), zz, cdep2, m,
  icl, carr2, nes, carr3, cdep3, iqis2, qs2,
  ki3, mq1, ncys2, iveh(1), zsut, nkas, isps,
  nuas, dtuas, suts, iqss, qsi, nqss, rqs2,
  niqs, dtiqs, jdq, nuxs, dtuxs, icx, jwh,
  jac, ixts, tuz, dtuz, wuz, ib, ijk, j1, jz2,
  15, st, ipt, jdum, dy2, jvar)
icx = 0
if(ixts.eq.1) joc = 0
if(ixts.eq.1.and.ijk.eq.1) then
```

```
ixts = 0
ijk = 0
endif
C
C------ Arrival time at intersection 3 --------
C
if(ixts.eq.1) go to 110
150 if(ixts.eq.1) then
  jz2 = 12
  if(kc.eq.2) jz2 = 2
  if(kc.eq.3) jz2 = 3
  ixts = 0
if(kc.eq.2) then
if(tuz(jz2).lt.cdep2(m).and.tuz(jz2).gt.cdep2(m
  )-icl. and.ig(2).lt.igs(2)) then
  tuz(jz2) = tuz(jz2) + icl
elseif(tuz(jz2).lt.cdep2(m)-
  icl.and.tuz(jz2).gt. cdep2(m)-
  2*icl.and.ig(2).lt.igs(2)-icl) then
  tuz(jz2) = tuz(jz2) + 2*icl
endif
endif
  if(tuz(jz2).gt.igs(2)) then
    dtuz(jz2) = tuz(jz2) - igs(2)
    cdep2(m) = cdep2(m) + dtuz(jz2)
    if(cdep2(m).ge.irs(2)) then
      call UPDATE(nes, ks2, igs(2), irs(2), zz,
  cdep2, m, icl,        ncys2)
      call UPDATE(ngs, ks3, igs(3), irs(3), zz,
  cdep3, m, icl,        ncys3)
      nkas(isps) = 0
      isps = isps + 1
      ixts = 1
      go to 110
    endif
  endif
endif

if(ijk.eq.0.and.((nca.eq.1.and.(ks2.eq.2.or.
  jdq.eq.11.and.(ks2.eq.4.or.ks2.eq.5))).or.
  (nca.eq.2.and.(ks2.eq.2.or.ks2.eq.3.or.jdq.e
  q.11.and.
  (ks2.eq.4.or.ks2.eq.5.or.ks2.eq.6.or.ks2.eq.
  7))))) then
  jz2 = 12
  if(kc.eq.2) jz2 = 2
  if(kc.eq.3) jz2 = 3
  if(nqa.ge.iveh(1).and.st(jz2)-1.1.gt.irs(2))
  then
    call UPDATE(nes, ks2, igs(2), irs(2), zz,
  cdep2, m, icl,      ncys2)
    go to 152
  endif
  jos = jos +1
  arrra(jos) = cdep2(m)


  if(kc.ne.1.and.(nqa.eq.iveh(1).or.joc+nqa.eq
  .iveh(1).or.
  joc+nka(isp).eq.iveh(1)+1).and.st(jz2)-
  1.1.gt.cdep2(m))   then
    niqs = 1
    ssc = st(jz2)
    if(ssc.gt.irs(2)) then
      imqs = 1
      niqs = 0
    endif
    if(ssc.gt.cdep2(m)) dtiqs = ssc - cdep2(m)
    endif
    joc = joc + 1
    go to 155
  endif

  ELSEIF(iups.eq.1) then
    carr1(m) = cparr1
    cdep1(m) = cpdep1
    carr2(m) = cparr2
    cdep2(m) = cpdep2
    nls = 0
    iups = 0
    iqes = 0
    iqeas = 0
  ELSE
    carr1(m) = cparr1
    cdep1(m) = cpdep1
    carr2(m) = cparr2
    cdep2(m) = cpdep2
  ENDIF

  icx = 1300
  if(kc.eq.1) icw = 10
  152call ARRQLA(carr3, cdep3, cdep2, m, nes,
    dist(2), speed, at3, carrr, is, nns, nls,
    iups, ki3, mq3, inds1, iqts, qst, iqes,
    iqeas, iveh(1), icx, nkas, isps, suts, niis,
    iqtis, qsti, niqs, dtiqs,  nqsts, rqst, jdq,
    amp, icw, jac, jdum, jvar)
  icx = 0
  icw = 0
  C
  C------ Departure time at intersection 3 ------
  C
  jwh = jwh2
  call DEPART(ks3, igs(3), irs(3), zz, cdep3, m,
    icl, carr3, ngs,  ncys3, jac, 0, jwh, ipt,
    jdum, dy3, jvar)
  mprod = 1
  call UPDNKA(ks3, igs(3), irs(3), zz, cdep3, m,
    jdq, lgic, icl, carr3, ngs, mq3, 0, ncys3,
    zsut, iveh(1), nkas, isps, suts,  mprod,
    jac)

  155 continue

  if(nls.ne.0) then
    cparr1 = carr1(m)
    cpdep1 = cdep1(m)
    cparr2 = carr2(m)
    cpdep2 = cdep2(m)
    carr1(m) = 0.
    cdep1(m) = 0.
    carr2(m) = 0.
    cdep2(m) = 0.
  endif

  ki3p = jos

  if(mult.eq.0) then
  call FRINT(carr1, cdep1, carr2, cdep2, carr3,
    cdep3, ku3, m, is,  jos, mq3)
  endif

  if(ki3p.ne.ki3(m)) go to 162
  if(ncys3.lt.isps) then
    do 160 ik = isps,  isps
      nkas(ik) = nkas(ik) + 1
```

```
160  continue
else
  do 161 ik = isps,  ncys3
    nkas(ik) = nkas(ik) + 1
161  continue
endif

162 if(nls.ne.0) then
  carr1(m) = cparr1
  cdep1(m) = cpdep1
  carr2(m) = cparr2
  cdep2(m) = cpdep2
endif

if(imqs.eq.1) go to 110

100 continue
110ma(j1) = m
if(igs(2).ge.ismt) jvol = jvol - 1
if(imqs.eq.1) then
  ma(j1) = m + 1
  if(igs(2).ge.ismt) jvol = jvol + 1
endif

if(krt.ne.1) go to 1100
***********************************************
******    Right lane of the cross street    ****
***********************************************
c
1000 if(kfa(j2).eq.0) then
ixts1 = 0
joc1 = 0
imqs1 = 0
lric = icl - lgic
if(mult.eq.0) then
write(ku4, 121) kc
write(ku4, 221)
221 format('   *** Right lane of the cross
  street ***'/)
call HEAD(ku4, icl, lgic, 0, lric, noff(2),
  noff(3))
endif
C------
call CTIME(kc, igs1, irs1, noff, lgic, icl, ip,
  jp)
call INIT(darr1, ddep1, darr2, ddep2, darr3,
  ddep3, ncs1, nes1, ngs1, ks11, ks21, ks31,
  is1, nns1, m)
call INIT1(jos1, nrs, iups1, iqis21, iqis31,
  ncys11, ncys21, ncys31, mq11, mq21, mq31,
  inds11, iqts1, iqes1, iqeas1,  iqtis1,
  nqsts1)
call INIT2(niis1, isps1, iqss1, nqss1, niqs1,
  dtiqs1, nuas1, dtuas1, nuxs1, dtuxs1)
call SIGADJ(irs1, igs1, lric, lgic, icl, dist,
  speed, jac, kc,  darrr, jdum, jvar)

call DPHDWY(zz, zsut)
igsu4 = igs1(2)
jn = 0
nca1 = nca
if(jdq.eq.11) nca1 = 3*nca
igsu4 = igsu4 + icl
175do 180 kt = 1,  nca1
  jn = jn + 1
  arrrb(jn) = igsu4 + zz(kt) + 2.5
180 continue
igsu4 = igsu4 + icl

if(jn.lt.290) go to 175
kfa(j2) = 1
endif
C
C------ Arrival time at intersection 1 --------
C
do 200 m = ma(j2),  ma(j2)+ntg
if(igs1(2).ge.ismt) jvol = jvol + 1

if(ixts1.eq.1) go to 250
ki21(m) = is1
ki31(m) = jos1

IF(nrs.eq.0) then
    darr1(m) = 7.2 * (m-is1)
C
C------ Departure time at intersection 1 ------
C
call DEPTQ(ks11, igs1(1), irs1(1), zz, ddep1,
  m, icl, darr1, ncs1, darr2, ddep2, ki21,
  mq21, iveh(1), ncys11, nui1, dtui1, jac, kc,
  jdum,  dy4, jvar)
if(imqs1.eq.1) imqs1 = 0
C
C------ Arrival time at intersection 2 --------
C
call ARRNT(darr2, ddep2, ddep1, m, ncs1,
  dist(1), speed, mq21, jac,  jdum, jvar)
C
C------ Departure time at intersection 2 ------
C
jwh = jwh3
jz2 = 1
if(kc.eq.2) jz2 = 4
if(kc.eq.3) jz2 = 11
call DEPQLS(ks21, igs1(2), irs1(2), zz, ddep2,
  m, icl, darr2, nes1, darr3, ddep3, iqis21,
  qs21, ki31, mq11, ncys21, iveh(1), zsut,
  nkas1, isps1, nuas1, dtuas1, suts1, iqss1,
  qsi1, nqss1, rqs21, niqs1, dtiqs1, jdq,
  nuxs1, dtuxs1, icx,  jwh, jac, ixts1, tuz,
  dtuz, wuz, ib, ijk1, j2, jz2, 15, st, ipt,
  jdum, dy5, jvar)
if(ixts1.eq.1) joc1 = 0

if(ixts1.eq.1.and.ijk1.eq.1) then
ixts1 = 0
ijk1 = 0
endif
C
C------ Arrival time at intersection 3 --------
C
if(ixts1.eq.1) go to 210
250 if(ixts1.eq.1) then
  jz2 = 1
  if(kc.eq.2) jz2 = 4
  if(kc.eq.3) jz2 = 11
  ixts1 = 0

if(kc.eq.1) then
if(tuz(jz2).lt.ddep2(m).and.tuz(jz2).gt.ddep2(m
  )-icl. and.ig(1).lt.igs1(2)) then
  tuz(jz2) = tuz(jz2) + icl
elseif(tuz(jz2).lt.ddep2(m)-
  icl.and.tuz(jz2).gt. ddep2(m)-
  2*icl.and.ig(1).lt.igs1(2)-icl) then
  tuz(jz2) = tuz(jz2) + 2*icl
endif
```

```
  endif

  if(tuz(jz2).gt.igs1(2)) then
    dtuz(jz2) = tuz(jz2) - igs1(2)
    ddep2(m) = ddep2(m) + dtuz(jz2)
     if(ddep2(m).ge.irs1(2)) then
240 call UPDATE(nes1, ks21, igs1(2), irs1(2),
  zz, ddep2, m, icl, ncys21)
        nkas1(isps1) = 0
      isps1 = isps1 + 1
      ixts1 = 1
      go to 210
    endif
  endif
endif

if(ijk1.eq.0.and.((nca.eq.1.and.(ks21.eq.2.or.
  jdq.eq.11.and.(ks21.eq.4.or.ks21.eq.5))).or.
  (nca.eq.2.and.(ks21.eq.2.or.ks21.eq.3.or.jdq
  .eq.11.and.
  (ks21.eq.4.or.ks21.eq.5.or.ks21.eq.6.or.ks21
  .eq.7)))) ) then
  jz2 = 1
  if(kc.eq.2) jz2 = 4
  if(kc.eq.3) jz2 = 11

  if(nqa1.ge.iveh(1).and.st(jz2)-
  1.1.gt.irs1(2)) then
    call UPDATE(nes1, ks21, igs1(2), irs1(2),
  zz, ddep2, m, icl,      ncys21)
    go to 252
  endif

  jos1 = jos1 +1

  if(kc.ne.3.and.(nqa1.eq.iveh(1).or.joc1+nqa1
  .eq.iveh(1)
  .or.joc1+nka1(isp1).eq.iveh(1)+1).and.st(jz2
  )-1.1.gt.   ddep2(m)) then
    niqs1 = 1
    ssc1 = st(jz2)
    if(ssc1.gt.irs1(2)) then
      imqs1 = 1
      niqs1 = 0
    endif
    if(ssc1.gt.ddep2(m)) dtiqs1 = ssc1 -
  ddep2(m)
  endif

  joc1 = joc1 + 1
  go to 255
endif

ELSEIF(iups1.eq.1) then
  darr1(m) = dparr1
  ddep1(m) = dpdep1
  darr2(m) = dparr2
  ddep2(m) = dpdep2
  nrs = 0
  iups1 = 0
  iqes1 = 0
  iqeas1 = 0
ELSE
  darr1(m) = dparr1
  ddep1(m) = dpdep1
  darr2(m) = dparr2
  ddep2(m) = dpdep2
ENDIF

if(kc.eq.3) icx = 34
252call ARRQLA(darr3, ddep3, ddep2, m, nes1,
  dist(2), speed, at31, darrr, is1, nns1, nrs,
  iups1, ki31, mq31, inds11, iqts1, qst1,
  iqes1, iqeas1, iveh(1), icx, nkas1, isps1,
  suts1, niis1, iqtis1, qsti1, niqs1, dtiqs1,
  nqsts1, rqst1, jdq, amp, icw, jac, jdum,
  jvar)
      icx = 0
C
C------ Departure time at intersection 3 ------
  C
jwh = jwh4
call DEPART(ks31, igs1(3), irs1(3), zz, ddep3,
  m, icl, darr3, ngs1, ncys31, jac, 0, jwh,
  ipt, jdum, dy6, jvar)
mprod = 1
call UPDNKA(ks31, igs1(3), irs1(3), zz, ddep3,
  m, jdq, lgic, icl, darr3, ngs1, mq31, 0,
  ncys31, zsut, iveh(1), nkas1, isps1, suts1,
  mprod, jac)

255 continue

if(nrs.ne.0) then
  dparr1 = darr1(m)
  dpdep1 = ddep1(m)
  dparr2 = darr2(m)
  dpdep2 = ddep2(m)
  darr1(m) = 0.
  ddep1(m) = 0.
  darr2(m) = 0.
  ddep2(m) = 0.
endif

ki31p = jos1

if(mult.eq.0) then
call FRINT(darr1, ddep1, darr2, ddep2, darr3,
  ddep3, ku4, m,  is1, jos1, mq31)
endif

if(imqs1.eq.1) then
  call UPDATE(nes1, ks21, igs1(2), irs1(2), zz,
  ddep2, m, icl,    ncys21)
  niqs1 = 0
  ks21 = ks21 - 1
  joc1 = 0
endif

if(ki31p.ne.ki31(m)) go to 262
if(ncys31.lt.isps1) then
    do 260 ik = isps1,  isps1
    nkas1(ik) = nkas1(ik) + 1
260  continue
else
  do 261 ik = isps1,  ncys31
    nkas1(ik) = nkas1(ik) + 1
261  continue
endif

262 if(nrs.ne.0) then
  darr1(m) = dparr1
  ddep1(m) = dpdep1
  darr2(m) = dparr2
  ddep2(m) = dpdep2
```

```
endif

if(imqs1.eq.1) go to 210

200 continue
210ma(j2) = m
if(igs1(2).ge.ismt) jvol = jvol - 1
if(imqs1.eq.1) then
  ma(j2) = m + 1
  if(igs1(2).ge.ismt) jvol = jvol + 1
endif

1100 write(50)ixts, ncs, nes, ngs, ks1, ks2,
  ks3, is, nns, jos, nls, iups, iqis3, ncys1,
  ncys2, ncys3, mq2, mq3, inds1, iqes, iqeas,
  niis, niqs, nuas, nuxs, ixts1, ncs1, nes1,
  ngs1, ks11, ks21, ks31, is1, nns1, jos1,
  nrs, iups1, iqis31, ncys11, ncys21, ncys31,
  mq21, mq31, inds11, iqes1, iqeas1, niis1,
  niqs1
write(50)nuas1, nuxs1, ijk, ijk1, kfi, nui,
  nui1, imqs, imqs1, joc, joc1, nuzi, nuzi1
write(50)dtiqs, dtuas, dtuxs, cparr1, cpdep1,
  cparr2, cpdep2, dtiqs1, dtuas1, dtuxs1,
  dtui, dtui1, dtuzi, dtuzi1, dparr1, dpdep1,
  dparr2, dpdep2
rewind(50)
return
end
*******
subroutine DEPTQ(k, ig, ir, z, dep, m, icl,
  arr, nce, arrd, depd, ki, nq, iveh1, ncy,
  nui, dtui, jac, kc, jdum, dphy, jvar)
dimension z(0:100), dep(0:2000), arr(0:2000)
dimension zsut(0:100), depd(0:2000),
  arrd(0:2000), ki(0:2000)
dimension dphy(0:60)
if(m.eq.1) then
  nq = 0
  st1 = 0
endif
k = k + 1
call DPHDWY(z, zsut)
if(jvar.eq.1.and.k.eq.1.or.k.eq.2) then
call nordev(jdum, dphy, icx)
endif

ircl = ir+icl*(2-jac)
call FINDEV(m, depd, iveh1, lks, st1)
ipo = 9
call NOQUE(arrd, depd, 0, 0, iveh1, ki, m, m1,
  nq, 0, 0, 0, ipo, 0)
if(arr(m).le.ircl) then
  if(nq.eq.iveh1+1) then
    if(jvar.eq.0) then
      depm = z(k) + ig
    else
      depm = dphy(k) + ig
    endif
    nui = 1
    dtui = st1 - depm
    dep(m) = st1
    if(st1.gt.ircl) then
      igcl = ig + icl
      if(dep(m).gt.igcl) then
 call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
       dtui = st1 - ig - 2.05
 if(dtui.gt.icl) then
260   call UPDATE(nce, k, ig, ir, z, dep, m,
   icl, ncy)
    dtui = dtui - icl
         dep(m) = st1
    if(dtui.gt.icl) go to 260
 endif
      endif
     endif
     go to 100
   endif

   if(jvar.eq.0) then
     dep(m) = z(k) + ig
   else
     dep(m) = dphy(k) + ig
   endif
   if(dep(m).gt.ircl)then
     call UPDATE(nce, k, ig, ir, z, dep, m, icl,
   ncy)
     nui = 0
   endif
   if(arr(m).ge.dep(m)) dep(m) = arr(m)
   if(nui.eq.1) dep(m) = dep(m) + dtui
100  continue
elseif(arr(m).gt.ircl) then
  call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
  nui = 0
  if(arr(m).ge.dep(m)) dep(m) = arr(m)
endif

m10 = m - 1
call findep(dep, m10)
if(dep(m).lt.dep(m10)+0.7) then
  dep(m) = dep(m10) + 0.7
endif
ircl = ir+icl*(2-jac)

if(k.gt.2.and.arr(m-1).eq.dep(m-1).and.
  arr(m).gt.ig.and.arr(m).le.ircl) then
  dep(m) = arr(m)
endif

if(dep(m).gt.ircl) then
  call UPDATE(nce, k, ig, ir, z, dep, m, icl,
  ncy)
  nui = 0
endif
return
end
*******
subroutine PRN(arr1, dep1, arr2, dep2, arr3,
  dep3, ku, m, j, jo, nq2, nq3, nq9)
dimension arr1(0:2000), dep1(0:2000),
  arr2(0:2000)
dimension dep2(0:2000), arr3(0:2000),
  dep3(0:2000)
write(ku, 100) m, (m-j), j, jo, arr1(m),
  dep1(m), nq2, arr2(m), dep2(m), nq3, nq9,
  arr3(m), dep3(m)
100 format( 4I4,  2(1X, 2F8.1, I3), 1X, I3,
  2F7.1)
return
end
*******
subroutine FRINT(arr1, dep1, arr2, dep2, arr3,
  dep3, ku, m, j,  jo, nq9)
```

```
dimension arr1(0:2000), dep1(0:2000),
  arr2(0:2000)
dimension dep2(0:2000), arr3(0:2000),
  dep3(0:2000)
write(ku, 100) m, (m-j), j, jo, arr1(m),
  dep1(m), arr2(m), dep2(m), nq9, arr3(m),
  dep3(m)
100 format(4I4, 2(1X, 2F8.1), I3, 1X, 2F8.1)
return
end
*******
subroutine HEAD(j, icl, lgi, ipt, lri, noff2,
  noff3)
write(j, 100) icl, lgi, ipt, lri, noff2, noff3
100 format(3x, 'C=', i3, 'sec, ', ' g=', i2,
  'sec, ', ' l=', i2, 'sec, ', ' r=', i2,
  'sec, ', ' off2=', i2, 'sec, ', ' off3=',
  i2, 'sec'/)
write(j, 200)
200 format('   Σ thru  turn       *** Int #1
  ***',          '    *** Int #2 ***',          '
  *** Int #3 ***'/       '          in  out
  A.time  D.time',        '   A.time
  D.time',        '   A.time   D.time'/)
    return
end
*******
subroutine INPUT(icl, dist, iveh, speed, ni)
dimension dist(20), iveh(20)
iveh(1) = int(dist(1)/20)
iveh(2) = int(dist(2)/20)
speed = 18.1
ni = 3
return
end
*******
subroutine INIT(arr1, dep1, arr2, dep2, arr3,
  dep3, nc, ne, ng, k1, k2, k3, i, n, m)
dimension arr1(0:2000), dep1(0:2000),
  arr2(0:2000)
dimension dep2(0:2000), arr3(0:2000),
  dep3(0:2000)
nc = 0
ne = 0
ng = 0
k1 = 0
k2 = 0
k3 = 0
i = 0
n = 1
do 100 m = 0, 2000
 arr1(m) = 0.
 dep1(m) = 0.
 arr2(m) = 0.
 dep2(m) = 0.
 arr3(m) = 0.
100 dep3(m) = 0.
return
end
*******
subroutine INIT1(jo, nnn, iup, iqi2, iqi3,
  ncy1, ncy2, ncy3, nq2, nq3, nq9, ind1, iqt,
  iqe, iqea, iqti, nqst)
jo = 0
nnn = 0
iup = 0
iqi2 = 0
iqi3 = 0
ncy1 = 1
ncy2 = 1
ncy3 = 1
nq2 = 0
nq3 = 0
nq9 = 0
ind1 = 0
iqt = 0
iqe = 0
iqea = 0
iqti = 0
nqst = 0
return
end
*******
subroutine INIT2(nii, isp, iqs, nqs, niq, dtiq,
  nua, dtua, nux,  dtux)
  nii = 0
  isp = 1
  iqs = 0
  nqs = 0
  niq = 0
  dtiq = 0.
  nua = 0
  dtua = 0.
  nux = 0
  dtux = 0.
return
end
*******
subroutine SIGNAL(j1, speed, icl, ni, ir, ig,
  noff, lgi)
dimension ir(20), ig(20), noff(20)
ir(1) = 0
ig(1) = icl - lgi
ir1 = ir(1)
ig1 = ig(1)
do 100 n = 1,  ni-1
  ir(n+1) = ir(1) + noff(n+1)
  ig(n+1) = ig(1) + noff(n+1)
100 continue
if(j1.eq.3.or.j1.eq.4) then
  ir(1) = ir(ni)
  ig(1) = ig(ni)
  if(ir(1).gt.0) then
    ig(1) = ig(1) - icl
    ir(1) = ir(1) - icl
  endif
  ir(2) = ir(2)
  ig(2) = ig(2)
  ir(3) = ir1
  ig(3) = ig1
endif
return
end
*******
subroutine SIGNA1(jc, ni, ir, ig, ir1, ig1)
dimension ir(20), ig(20), ir1(20), ig1(20)
if(jc.eq.1) then
do 100 n = 1,  ni
  ir1(n) = ir(n)
  ig1(n) = ig(n)
100 continue
else
do 200 n = 1,  ni
  ir1(n) = ir(4-n)
  ig1(n) = ig(4-n)
200 continue
```

```
endif
return
end
*******
subroutine CTIME(kc, ig, ir, noff, lgi, icl,
  ip, jp)
dimension ir(20), ig(20), noff(20)
  ig(2) = noff(kc)
  if(ig(2).ge.icl) ig(2) = ig(2) - icl
  ir(2) = ig(2) + lgi
  ig(1) = ig(2) - ip
  ir(1) = ir(2) - ip
  ig(3) = ig(1) + jp
  ir(3) = ir(1) + jp
return
end
*******
subroutine SIGADJ(ir, ig, lri, lgi, icl, dist,
  speed, jac, kc,  carrr, jdum, jvar)
dimension ir(20), ig(20), dist(20),
  carrr(0:500)
rlo = 0.0
if(jvar.eq.1) then
call rspeed(jdum, rlo, speed, icx)
else
  speed = 30.9
endif
wid = 40.
if(jac.eq.2) wid = 60.
if(ig(1)+lgi.lt.0.or.jac.eq.2.and.ir(1).le.7.2)
  then
  ir(1) = ir(1) + icl
  ig(1) = ig(1) + icl
endif
a1 = 7.2
if(jac.eq.1) a1 = lri + 2.04
if(kc.eq.1) then
  a1 = ig(1) + 2.04
  if(a1.lt.7.2) a1 = 7.2
endif
arr2f = a1 + (dist(1)+wid)/speed
ircl2 = ir(2) - icl
if(jac.eq.1) ircl2 = ir(2)
if(kc.eq.1) ircl2 = ig(2)

if(arr2f.gt.ig(2)+lgi) then
  ir(2) = ir(2) + icl
  ig(2) = ig(2) + icl
elseif(arr2f.lt.ircl2) then
  ir(2) = ir(2) - icl
  ig(2) = ig(2) - icl
endif

if(arr2f.lt.ig(2)) dep2f = ig(2) + 2.04
if(arr2f.ge.ig(2)) dep2f = arr2f
arr3f = dep2f + (dist(2)+wid)/speed
ircl3 = ir(3) - icl
if(jac.eq.1) ircl3 = ir(3)

if(arr3f.gt.ig(3)+lgi) then
  ig(3) = ig(3) + icl
  ir(3) = ir(3) + icl
elseif(arr3f.lt.ircl3) then
  ig(3) = ig(3) - icl
  ir(3) = ir(3) - icl
endif
if(jac.eq.2.and.ig(3).gt.carrr(1)+icl) then
  ig(3) = ig(3) - icl
  ir(3) = ir(3) - icl
endif
return
end
*******
subroutine UPDATE(nce, k, ig, ir, z, dep, m,
  icl, ncy)
dimension z(0:100),  dep(0:2000),  zsut(0:100)
  nce = 0
  k = 1
  ig = ig + icl
  ir = ir + icl
  call DPHDWY(z, zsut)
  dep(m) = z(k) + ig
  ncy = ncy + 1
return
end
*******
subroutine UPDAT1(nux, niq, nua)
  nux = 0
  niq = 0
  nua = 0
return
end
*******
subroutine PRNQS(ku, nint, icl, lgi, jiq, qs,
  jncy, jiqs, qsi, jnrqs, rqs, prqs, pqs,
  pqsn, ptqs, ismt, smtj, mult)
dimension qs(80), qsi(80), rqs(80)
if(mult.eq.0) then
  smpd = smtj - ismt
  write(ku, 111) nint
111  format(/' **** QS due to thru traffic at
  Int.#', i2,    ' ****'/)
  write(ku, 113) ismt
113  format('  Simulation starts at t =', i4, '
  sec')
  write(ku, 115) smtj
115  format('  Simulation ends at t =', f6.0, '
  sec'/)
  write(ku, 117) jncy
117  format(2x, 'No. of cycle:', i3/)

  do 100 iq = 1, jiq
    write(ku, 110) iq, qs(iq)
110    format(i8, 8x, f8.1)
100  continue
  if(jiq.eq.0) then
    write(ku, 120) jiq
120    format(2x, 'No. of QSi-full:', i3/)
  else
    call CNNQS(ismt, smtj, qs, jiq)
    write(ku, 120) jiq
  endif

  do 200 jq = 1, jiqs
    write(ku, 210) jq, qsi(jq)
210    format(i8, 8x, f8.1)
200  continue
  if(jiqs.eq.0) then
    write(ku, 220) jiqs
220    format(2x, 'No. of QSi-n.full:',  i3/)
  else
    call CNNQS(ismt, smtj, qsi, jiqs)
    write(ku, 220) jiqs
  endif
c
  do 300 ia = 1, jnrqs
```

```
    write(ku, 310) ia, rqs(ia)
310     format(i8, 8x, f8.1)
300  continue
  if(jnrqs.eq.0) then
    write(ku, 320) jnrqs
320     format(2x, 'No. of QS-full/n.full:',
  i3/)
  else
    call CNNQS(ismt, smtj, rqs, jnrqs)
    write(ku, 320) jnrqs
  endif
endif

if(mult.eq.1) then
    if(jiq.ne.0) then
    call CNNQS(ismt, smtj, qs, jiq)
  endif
  if(jiqs.ne.0) then
    call CNNQS(ismt, smtj, qsi, jiqs)
  endif
  if(jnrqs.ne.0) then
    call CNNQS(ismt, smtj, rqs, jnrqs)
  endif
endif

bncy = jncy
prqs = jnrqs/bncy
pqs = jiq/bncy
pqsn = jiqs/bncy
jtotal = jiq + jiqs + jnrqs
ptqs = jtotal/bncy

if(mult.eq.0) then
  write(ku, 340) prqs
340  format('  No.of QS-f/n.f per cycle:',
  f5.2, ' / cycle'/)
  write(ku, 360) jtotal
360  format(2x, 'No. of QS(i)-total:', i3/)
  write(ku, 380) ptqs
380  format('  No.of QS(i)-total per cycle:',
  f5.2, ' / cycle'/)
  write(ku, 460) smpd
460  format(2x, 'Simulation Period:', f6.0, '
  sec'/)
endif
return
end
*******
subroutine CNNQS(ismt, smtj, wqs, jnnqs)
dimension wqs(80)
nnqs = 0
do 100 ii = 1,  jnnqs
  if(wqs(ii).gt.ismt.and.wqs(ii).lt.smtj) then
    nnqs = nnqs + 1
  endif
100 continue
jnnqs = nnqs
return
end
*******
subroutine PRNQST(ku, nint, jiqt, qst, jiqti,
  qsti, jncy, jnrqst, rqst, prqst, pqst,
  pqstn, ptqst, ismt, smtj, mult)
dimension qst(80), qsti(80), rqst(80)
if(mult.eq.0) then
  write(ku, 111) nint
111  format(/' **** QS due to turn-in traffic
  at Int. #', i2,     ' ****'/)
  write(ku, 117) jncy
117  format(2x, 'No. of cycle:', i3/)

  do 100 iq = 1, jiqt
    write(ku, 110) iq, qst(iq)
110     format(i8, 8x, f8.1)
100  continue
  if(jiqt.eq.0) then
    write(ku, 120) jiqt
120     format(2x,  'No. of QSi-full:', i3/)
  else
    call CNNQS(ismt, smtj, qst, jiqt)
    write(ku, 120) jiqt
  endif

  do 200 ia = 1, jiqti
    write(ku, 210) ia, qsti(ia)
210     format(i8, 8x, f8.1)
200  continue
  if(jiqti.eq.0) then
    write(ku, 220) jiqti
220     format(2x, 'No. of QSi-n.full:', i3/)
  else
    call CNNQS(ismt, smtj, qsti, jiqti)
    write(ku, 220) jiqti
  endif

  do 300 iat = 1, jnrqst
    write(ku, 310) iat, rqst(iat)
310     format(i8, 8x, f8.1)
300  continue
  if(jnrqst.eq.0) then
    write(ku, 320) jnrqst
320     format(2x, 'No. of QS-full/n.full:',
  i3/)
  else
    call CNNQS(ismt, smtj, rqst, jnrqst)
    write(ku, 320) jnrqst
  endif
endif

if(mult.eq.1) then
  if(jiqt.ne.0) then
    call CNNQS(ismt, smtj, qst, jiqt)
  endif
  if(jiqti.ne.0) then
    call CNNQS(ismt, smtj, qsti, jiqti)
  endif
  if(jnrqst.ne.0) then
    call CNNQS(ismt, smtj, rqst, jnrqst)
  endif
endif

bncy = jncy
prqst = jnrqst/bncy
pqst = jiqt/bncy
pqstn = jiqti/bncy
jtotal = jiqt + jiqti + jnrqst
ptqst = jtotal/bncy

if(mult.eq.0) then
  write(ku, 340) prqst
340  format('  No.of QS-f/n.f per cycle:',
  f5.2, ' / cycle'/)
  write(ku, 360) jtotal
360  format(2x, 'No. of QS(i)-total:', i3/)
  write(ku, 380) ptqst
```

```
380  format(' No.of QS(i)-total per cycle:',
  f5.2, ' / cycle'/)
endif
return
end
*******
subroutine PRNOP1(ku, jc, ln, incre, ioff1,
  ioff2, joff1, joff2, pt)
dimension pt(0:15, 0:15)
if(jc.eq.1) then
  write(ku, 10) ln
10  format(/4X, 'arterial lane no.', i1/)
elseif(jc.eq.2) then
  write(ku, 20) ln
20  format(/4X, 'cross street no.', i1/)
elseif(jc.eq.3) then
  write(ku, 30)
30  format(/4X, 'arterial total'/)
elseif(jc.eq.4) then
  write(ku, 40)
40  format(/4X, 'cross street total'/)
elseif(jc.eq.5) then
  write(ku, 50)
50  format(/4X, 'No.of QS per cycle: Total'/)
elseif(jc.eq.6) then
  write(ku, 60)
60  format(/4X, 'Simulation Period: Total'/)
endif
write(ku, 200) (jp*incre,  jp = joff1, joff2)
200 format(4x, 'off3:', i2, 14(3x, i3))
write(ku, 300)
300 format(2x, 'off2')
do 500 ip = ioff1,  ioff2
  if(jc.ne.6) then
    write(ku, 400) ip*incre, (pt(ip, jp),
  jp=joff1, joff2)
400    format(3x, i3, 15f6.2)
  else
    write(ku, 410) ip*incre, (pt(ip, jp),
  jp=joff1, joff2)
410    format(3x, i3, 15f6.0)
  endif
500 continue
return
end
*******
subroutine PRNOP2(ku, ln, incre, ioff1, ioff2,
  joff1, joff2, mt)
dimension mt(0:15, 0:15)
if(ln.eq.99) then
write(ku, 100)
100 format(/4X, 'No. of vehicles (warmup)'/)
go to 15
endif
write(ku, 10) ln
10 format(/4X, 'lane no.'i2/)
15 write(ku, 20) (jp*incre,  jp = joff1, joff2)
20 format(4x, 'off3:', i2, 14(3x, i3))
write(ku, 30)
30 format(2x, 'off2')
do 50 ip = ioff1,  ioff2
  write(ku, 40) ip*incre, (mt(ip, jp),  jp =
  joff1, joff2)
40  format(3x, i3, 15i6)
50 continue
return
end
*******
```

```
subroutine rspeed(jdum, rlo, speed, icx)
dimension rn(100), vnor(100)
idum = jdum - 100
jdum = jdum - 100
if(rlo.gt.0.001) then
  s = 2.57
else
  xbar = 30.9
  s = 3.81
endif
ni = 1
nj = 1

call rannum (idum, rn, ni)
call normal (rn, vnor, nj, icx)

if(rlo.gt.0.001) then
  speed = s*vnor(1) + 23.365 - 9.0025*rlo
else
  speed = s*vnor(1) + xbar
endif

return
end
*******
subroutine nordev(jdum, dphy, icx)
dimension rn(100), vnor(100), hdwy(100),
  dphy(0:60)
idum = jdum - 100
jdum = jdum - 100
s = 0.43
ni = 100
nj = 60
dphy(0) = 0.
call rannum (idum, rn, ni)
call normal (rn, vnor, nj, icx)
do 11 j = 1,  nj
if(j.eq.1) xbar = 2.04
if(j.eq.2) xbar = 2.46
if(j.eq.3) xbar = 2.12
if(j.ge.4) xbar = 1.82
hdwy(j) = s*vnor(j) + xbar
dphy(j) = dphy(j-1) + hdwy(j)
11 continue
return
end
*******
subroutine rannum(idum, rn, ni)
dimension r(97),  rn(100)
parameter (m1=259200,  ia1=7141,  ic1=54773,
  rm1=1./m1)
parameter (m2=134456,  ia2=8121,  ic2=28411,
  rm2=1./m2)
parameter (m3=243000,  ia3=4561,  ic3=51349)
data iff /0/
do 10 i = 1,  ni
if (idum.lt.0.or.iff.eq.0) then
    iff = 1
    ix1 = mod(ic1-idum, m1)
    ix1 = mod(ia1*ix1+ic1, m1)
    ix2 = mod(ix1, m2)
    ix1 = mod(ia1*ix1+ic1, m1)
    ix3 = mod(ix1, m3)
    do 11 j = 1,  97
ix1 = mod(ia1*ix1+ic1, m1)
ix2 = mod(ia2*ix2+ic2, m2)
r(j) = (float(ix1)+float(ix2)*rm2)*rm1
11    continue
```

```
    idum = 1
endif
ix1 = mod(ia1*ix1+ic1, m1)
ix2 = mod(ia2*ix2+ic2, m2)
ix3 = mod(ia3*ix3+ic3, m3)
j = 1 + (97*ix3)/m3
if (j.gt.97.or.j.lt.1) pause 'stop'
rn(i) = r(j)
r(j) = (float(ix1)+float(ix2)*rm2)*rm1
10 continue
return
end
*******
subroutine normal (rn, vnor, nj, icx)
dimension rn(100),  vnor(100)
data iset/0/
i = 0
iov = 0

do 50 k = 1,  nj
vnor(k) = 0.0
50 continue

do 100 j = 1,  nj
if (iset.eq.0) then
    i = i + 2
10    v1 = 2.*rn(i-1) - 1.
    v2 = 2.*rn(i) - 1.
    r = v1**2 + v2**2
    if (r.ge.1) then
    i = i + 2
    if(i.gt.100) then
      i = 0
      iov = 1
      go to 100
    endif
    go to 10
    endif
    fac = sqrt(-2.*log(r)/r)
    gset = v1*fac
    gasdev = v2*fac
    jj = j
    if(iov.eq.1) jj = jj - 1
    vnor(jj) = gasdev
    iset = 1
else
    gasdev = gset
    jj = j
    if(iov.eq.1) jj = jj - 1
    vnor(jj) = gasdev
    iset = 0
endif
100 continue
return
end
```

References

Battle, R.M., et al. 1956. "Starting Delay and Timing Spacing of Vehicles Entering Signalized Intersection", Highway Reseasrch Board Bulletin 112, HRB, National Research Council, Washington, D.C., 33-41.

Branton, D. 1978. "A Comparison of Observed and Estimated Queue Lengths at Oversaturated Traffic Signals", Traffic Engineering and Control, Vol.19, 322-327.

Carstens, R.L. 1971. "Some Traffic Parameters at Signalized Intersections", Traffic Engineering, Vol.41, No.11, 33-36.

Chang, E.C.P., et al. 1988. Arterial Signal Timing Optimization Using PASSER II-87-Microcomputer User's Guide. Report TTI-2-18-86-467-1, TTI, Texas A&M University System, College Station,Texas.

Church, R. and ReVelle, C. 1978. "Modelling an Oversaturated Intersection", Transportation Research, Vol.12, 185-189.

Edie, L. C. 1961. "Car-following and steady-state theory for noncongested traffic" Operations Research, Vol.9, 66-76.

Efstathiadis, Stilianos 1992. Variability of Departure Headways at Signalized Intersections, Master's thesis, The University of Texas at Austin

FHWA. 1988. TRAF-NETSIM User's Manual. FHWA, U.S. Department of Transportation, Washington, D.C.

Gazis, D.C. 1964. "Optimum Control of a System of Oversaturated Intersections", Operations Research, Vol.12, 815-831.

Gazis, D.C. and Potts, R.B. 1965. "The Oversaturated Intersection", Proc. 2nd Int. Symp. on the Theory of Road Traffic Flow, 221-237.

Gerlough, D.L. and Huber, M.J. 1975. Traffic Flow Theory. Transportation Research Board, National Research Council, Washington, D.C.

Gerlough, D.L. and Wagner, F.A. 1967. Improved Criteria for Traffic Signals at Individual Intersections. NCHRP Report 32, HRB, National Research Council, Washington, D.C.

George, Jr., E.T. and Heroy, Jr., F.M. 1966. "Starting Response of Traffic at Signalized Intersections", Traffic Engineering, 39-43.

Gordon, R. L. 1969. "A Technique for Control of Traffic at Critical Intersections", Transportation Science, Vol.4, 279-287.

Greenshields, B.D., Schapiro, D., Ericksen, E.L. 1947. "Traffic Performance at Urban Street Intersections", Technical Report No. 1, Yale Bureau of Highway Traffic, Eno Foundation for Highway Traffic Control.

Kim, Y. 1990. Development of Optimization Models for Signalized Intersections During Oversaturated Conditions, Ph. D. Dissertation, Texas A&M University, College Station, Texas.

King, G.F. and Wilkinson, M. 1976. "Relationship of Signal Design to Dischage Headway, Approach Capacity, and Delay", Transportation Research Record 615, TRB, National Research Council, Washington, D.C., 37-44.

Lee, B., et al. 1975. "Better Use of Signals Under Oversaturated Flows", Transportation Research Board SR153, National Research Council, Washington, D.C., 60-72.

Lee, J. and Chen, R.L. 1986. "Entering Headway at Signalized Intersections in a Small Metropolitan Area", Transportation Research Board 1091, TRB, National Research Council, Washington, D.C., 117-126

Lieberman, E.B., et al. 1986. " Congestion-Based Control Scheme for Closely Spaced, High Traffic Density Networks", Transportation Research Board 1057, TRB, National Research Council, Washington, D.C., 49-57.

Lieberman, E.B. and Woo, J.L. 1982. SIGOP-III User's Manual. Report FHWA-IP-82-A, FHWA, U.S. Department of Transportation, Washington, D.C.

Longley, D. 1968. "A Control Strategy for a Congested Computer-Controlled Traffic Network", Transportation Research, Vol.2, 391-408.

Longley, D., 1971. "A Simulation Study of a Traffic Network Control Scheme", Transportation Research, Vol.5, 39-57.

Lu, Y.J. 1984. "A Study of Left-Turn Maneuver Time for Signalized Intersections", ITE Journal, Vol.41, No.10, 42-47.

May, A.D. 1990. Traffic Flow Fundamentals, Prentice Hall, New Jersey, 376-414.

May, A.D. and Montgomery, F.O. 1986. "Control of Congestion at Highly Congested Junctions", Transportation Research Record 1057, TRB, National Research Council, Washington, D.C., 42-48.

May, A.D. and Pratt, D. 1968. "A Simulation Study of Load Factor at Signalized Intersections", Traffic Engineering, Vol.38, No.5, 44-49.

Michalopoulos, P.G. and Stephanopoulos, G. 1977a. "Oversaturated Signal Systems with Queue Length Constraints-I", Transportation Research, Vol.11, 413-421.

Michalopoulos, P.G. and Stephanopoulos, G. 1977b. "Oversaturated Signal Systems with Queue Length Constraints-II", Transportation Research, Vol.11, 423-428.

Michalopoulos, P.G. 1978. "Optimal Control of Oversaturated Intersections: theoretical and practical considerations", Traffic Engineering and Control, Vol.19, 216-221.

Michalopoulos, P.G. 1979. "An Algorithm for Real-time Control of Critical Intersections", Traffic Engineering and Control, Vol.20, 9-15.

Michalopoulos, P.G., et al. 1981."An Application of Shock Wave Theory to Traffic Signal Control", Transportation Research, Vol.15B, 35-51.

Moussavi, Massoum and Tarawneh, Mohammed 1990. "Variability of departure headways at signalized intersections", ITE 1990 Compendium of Technical Papers, Institute of Transportation Engineers, 313-317.

Newell, G. 1989. Theory of Highway Traffic Signals. Course Notes UCB-ITS-CN-89-1, Institute of Transportation Studies, University of California at Berkeley.

OECD, 1981. Traffic Control in Saturated Conditions. Organization for Economic Co-operation and Development, Paris, France.

Pignataro, L.J., et al. 1978. Traffic Control in Oversaturated Street Networks. NCHRP Report 194, TRB, National Research Council, Washington, D.C.

Press, W.H., et al. 1986. Numerical Recipes, Cambridge University Press, Cambridge, 191-203.

Rathi, A.K. 1988. "A Control Scheme for High Traffic Density Sectors", Transportation Research,Vol.22B, 81-101.

Rouphail, N. 1991. "Cycle-by-Cycle Analysis of Congested Flows at Signalized Intersections", ITE Journal, Vol.61, No.3,33-36.

Shawaly, E.A.A., et al. 1988. "A Comparison of Observed, Estimated, and Simulated Queue Lengths and Delays at Oversaturated Signalized Junctions", Traffic Engineering and Control, Vol.29, 637-643.

Shibata, J. and Yamamoto, T. 1984. "Detection and Control of Congestion in Urban Road Networks", Traffic Engineering and Control, Vol.25, 438-444.

Wallace, C.E., et al. 1988. TRANSYT-7F User's Manual. Transportation Research Center, University of Florida, Gainesville, Florida.

Wong, S. 1990. "TRAF-NETSIM: How It Works, What It Does", ITE Journal, Vol.60, No.4, 22-27.