NOTE TO USERS

This reproduction is the best copy available.

UMI[®]

**Real-time Control and Optimization of Traffic Signal Timing
Transition for Emergency Vehicle Preemption**

By

Xiaolin Qin

B.Eng. (1985), M.Sc. (1991), Ph.D. (1994)

A Thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of the requirement
for the degree of Doctor of Philosophy

Department of Civil and Environmental Engineering

Carleton University

Ottawa, Ontario, Canada

December 2005

The Doctor of Philosophy Program in Civil Engineering
is a joint program with the University of Ottawa. administered by
the Ottawa-Carleton Institute for Civil Engineering

©Copyright 2005, Xiaolin Qin

Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-494-13397-X

Our file Notre référence

ISBN: 0-494-13397-X

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

Emergency vehicle (EV) operation has a very important role to play in saving lives and reducing property damage. In order to reduce the response time and the impact of EV operation on general traffic, research is underway around the world to find solutions. The EV signal preemption (EVSP) at intersections is one of the most attractive options.

Two control strategies are proposed for EVSP in this thesis. The first one, a real-time control strategy is used for signal transitioning from normal operation to EVSP (transition 1). Its main objective is to ensure that the approaching EV can cross the intersection at operating speed without any time delay, and it is accomplished by determining a suitable distance of EV detection away from the intersection on the basis of the EV operating speed, peak hour traffic conditions, and traffic signal timing plans. Another objective of the real-time control strategy is to decrease the impact of EVSP on general traffic. The impact on general traffic on the EV route is considered when the EV detection distance is determined. The impact on the cross road is considered by dynamically allotting green time that is not required for the EV to the traffic on the cross road according to current traffic conditions at the intersection and the state of signal indications. The real-time control strategy is also enhanced so that it can be used for single intersection control as well as coordinated route control

The second control strategy, implemented by an optimal control algorithm, is used

for the signal transitioning from EVSP back to normal operation (transition 2). Its objective is to decrease the impact of EVSP on general traffic after the EV has cleared the intersection by minimizing traffic queues at the intersection during a shortest possible time. The optimal control theory is first applied to the traffic signal transition problem, and a time-invarying dynamic system is established. A two-phase algorithm, consisting of a relaxation algorithm and a stepwise search strategy, is then adopted to overcome the difficulty in solving the optimal control model, which results from the interrelationship between successive signal sequences.

Computer software is developed in MATLAB environment for simulations of EVSP process under different signal timing transitioning control strategies. Simulation results indicate that the real-time control strategy and the optimal control strategy have better performance compared with commonly used fixed held time control strategy for transition 1 and well known smooth transitioning algorithm for transition 2, respectively.

Further simulations at different levels of saturation provide evidence that the proposed control strategies are applicable to various traffic conditions up to and slightly over-saturated level. Based on simulation results, recommendations are made on selection of weighting matrices in the optimal control algorithm. Finally, the real-time control strategy and the optimal control strategy are enhanced so that they can be used to deal with single EVSP occurrence as well as multiple EVSP occurrences.

Acknowledgements

Writing a thesis in English and finishing it is a challenging and rewarding process. I am excited that I can reach the final step of this work with help from many individuals.

The first and the most important one I would like to mention is Professor Ata Khan. As my supervisor, he has helped me at every stage of my research. Without his inspiration, I might not take interest in the study of traffic signal control for emergency vehicle operation. His constant encouragement and support enabled me to continue the study, and I am deeply impressed by his patience in editing and correcting my English. It has been my privilege to work under his supervision and I have learned so much from him.

I would also like to extend my thanks to members of the thesis proposal examination committee. They read my research proposal, and made valuable suggestions, which helped me to clarify my arguments and prepare the final thesis.

Many people have helped me in one way or another, and it is impossible to enumerate all of them here. I would, however, particularly like to express my gratitude to Mr. Kornel Mucsi. At the initial stage of the research, his information and suggestions helped me focus on the study of traffic signal timing transition for emergency vehicle preemption.

To my wife, Xiaohong Zhang, I owe my deep gratitude. Without her understanding and encouragement, I might not be able to overcome so many difficulties throughout the whole study process. I would like to dedicate this work to my wife for her endurance in support our family.

Finally, this thesis could not have been possible without the help from so many individuals, however, nobody except myself will take any responsibility for the argument put forward in this thesis.

Table of Contents

Chapter 1 Introduction	1
1.1 Overview	1
1.2 Research objectives and scope.....	3
1.3 Research approaches and methodologies.....	8
1.4 Organization of thesis.....	12
Chapter 2 Review of Literature.....	15
2.1 Overview of emergency vehicle signal preemption systems.....	15
2.2 Commonly used control strategies for EVSP operation.....	19
2.3 Potential possibilities for further research.....	23
2.4 Why is Dynamic Programming not used for this research.....	25
2.5 Why is Simulink not selected for this research.....	29
Chapter 3 Real-Time Control for Transition 1 - from Normal Operation to EVSP.....	33
3.1 Typical intersections and traffic signal timing phases.....	34
3.2 Assumptions.....	37
3.3 Determination of EV detection distance.....	40
3.4 Real-time control of traffic signal for EVSP.....	47

3.5 Real-time control for situations when minimum distance requirement of EV detection cannot be met.....	54
3.6 Real-time control strategy for coordinated route.....	56
3.6.1 General procedure to develop real-time control algorithm for coordinated route.....	56
3.6.2 Detailed formulation and algorithm for coordinated route control..	60
Chapter 4 Optimal Control for Transition 2 - from EVSP back to Normal Operation	65
4.1 Assumptions and control strategy for signal transitioning.....	65
4.2 General procedure to develop optimal control algorithm.....	66
4.3 Formulation of the optimal control model for signal transitioning.....	70
4.4 Theoretical analysis of the optimal control model.....	82
4.5 A Two-phase algorithm.....	85
4.6 Reduction of computation complexity.....	95
4.7 Algorithm summary.....	97
4.8 Computation consideration of data structure for M-step model.....	99
4.9 Can the M-step model solve original optimal control problem.....	103
4.10 Comparison between M-step model and Rolling Horizon strategy.....	104
Chapter 5 Simulation and Comparison of Control Strategies for	

Transition 1	106
5.1 Random generation of vehicle arrivals.....	106
5.2 Signal timing plans for normal operation.....	107
5.3 Simulation of normal operation.....	113
5.4 Simulation of control strategies for transition 1.....	114
5.4.1 Given conditions.....	114
5.4.2 Determination of EV detection distances.....	114
5.4.3 Situations for EV from main road.....	115
5.4.4 Situations for EV from side road.....	125
5.5 Analysis and comparison of simulation results for EV from main road...	133
5.6 Analysis and comparison of simulation results for EV from side road....	146
Chapter 6 Simulation and Comparison of Control Strategies for	
Transition 2	157
6.1 Situations for EV from main road.....	157
6.1.1 Simulation of smooth control strategy for transition 2.....	157
6.1.2 Simulation of optimal control strategy for transition 2.....	164
6.2 Situations for EV from side road..	168
6.2.1 Simulation of smooth control strategy for transition 2.....	168
6.2.2 Simulation of optimal control strategy for transition 2.....	170

6.3 Analysis and comparison of simulation results for EV from main road ..	171
6.4 Analysis and comparison of simulation results for EV from side road....	181
Chapter 7 Simulation and Comparison of Control Strategies for a	
Coordinated Route.....	191
7.1 Given Conditions.....	191
7.2 Simulations of normal operation.....	193
7.3 Simulations and comparison of control strategies for transition 1.....	195
7.3.1 Simulation of fixed held time control strategy.....	195
7.3.2 Simulation of real-time control strategy.....	196
7.3.3 Analysis and comparison of simulation results for transition 1.....	197
7.4 Simulations and comparison of control strategies for transition 2.....	205
7.4.1 Simulation of smooth transitioning algorithm.....	205
7.4.2 Simulation of optimal control strategy.....	206
7.4.3 Analysis and comparison of simulation results.....	208
Chapter 8 Control Efficiency at Different Saturation levels.....	216
8.1 Normal signal operation.....	216
8.2 Simulation of real-time control strategy for transition 1.....	222
8.3 Simulation of optimal control strategy for transition 2.....	225
8.4 Analysis of simulation results.....	228

Chapter 9 Impact of Weighting Matrices on Efficiency of Optimal	
Control Strategy	247
9.1 Simulation of optimal control strategy	247
9.2 Analysis of simulation results	265
9.3 Another simulation of optimal control strategy	267
Chapter 10 Control Efficiencies for Multiple EVSP Occurrences	286
10.1 Control strategy for multiple EVSP occurrences.....	286
10.2 Flow chart of control algorithm and simulation for multiple EVSP	
occurrences.....	293
10.3 Simulation of control strategy for multiple EVSP occurrences.....	297
10.4 Analysis of simulation results.....	300
Chapter 11 Summary, Conclusions and Recommendations	310
11.1 Summary of control strategy concept for EVSP.....	310
11.2 Summary of simulation results for proof of concept.....	311
11.3 Summary of further simulation results.....	316
11.4 Conclusions.....	323
11.5 Recommendations.....	326
References	329

List of Tables

Table 4.1 Possible signal indication sequences for a time step	91
Table 4.2 Data required in search for feasible signal sequences.....	95
Table 5.1 Queue lengths (veh) under normal operation.....	113
Table 5.2 EVSP scenarios for approach 1	116
Table 5.3 Simulation results for 50-second fixed held time on approach 1	118
Table 5.4 Simulation results for 30-second fixed held time on approach 1.....	119
Table 5.5 Simulation results for 59-second real-time control on approach 1.....	121
Table 5.6 Simulation results for 32-second real-time control on approach 1 (The requirement of minimum EV detection distance was not met)....	123
Table 5.7 EVSP scenarios for approach 2.....	125
Table 5.8 Simulation results for 30-second fixed held time on approach 2.....	127
Table 5.9 Simulation results for 20-second fixed held time on approach 2.....	128
Table 5.10 Simulation results for 23-second real-time control on approach 2....	130
Table 5.11 Simulation results for 12-second real-time control on approach 2 (The requirement of minimum EV detection distance was not met)....	132
Table 5.12 Failure number of control strategies for EV from approach 1	134
Table 5.13 Average spare time (sec) on approach 1 for transition 1	135

Table 5.14 Average green time (sec) for side road during transition 1	136
Table 5.15 Average queue lengths (veh) after transition 1 for EV from approach 1	138
Table 5.16 Average delay time (sec) over all scenarios during transition 1 for EV from approach 1	143
Table 5.17 Failure number of control strategies for EV from approach 2	146
Table 5.18 Average spare time (sec) on approach 2 for transition 1	146
Table 5.19 Average green time (sec) for main road during transition 1	148
Table 5.20 Average queue lengths (veh) after transition 1 for EV from approach 2	149
Table 5.21 Average delay time (sec) over all scenarios during transition 1 for EV from approach 2	154
Table 6.1 Queue lengths for EV from approach 1 under smooth transitioning	163
Table 6.2 Time parameters of optimal control for EV from approach 1	165
Table 6.3 Queue lengths for EV from approach 1 under optimal control strategy	166
Table 6.4 An example of control process for transition 2	167
Table 6.5 Queue lengths for EV from approach 2 under smooth transitioning	168
Table 6.6 Time parameters of optimal control for EV from approach 2	170

Table 6.7 Queue lengths for EV from approach 2 under optimal control strategy.....	170
Table 6.8 Average recovery time for EV from approach 1.....	172
Table 6.9 Average queue lengths (veh) after transition 2 for EV from approach 1.....	174
Table 6.10 Average delay time (sec) over all scenarios for EV from approach 1.....	178
Table 6.11 Average recovery time for EV from approach 2.....	181
Table 6.12 Average queue lengths (veh) after transition 2 for EV from approach 2.....	182
Table 6.13 Average delay time over all scenarios for EV from approach	186
Table 7.1 Traffic parameters for subject EV route.....	192
Table 7.2 Average queue under normal operation.....	194
Table 7.3 EVSP scenarios for subject route.....	195
Table 7.4 Simulation results of 50-second fixed held time control strategy.....	196
Table 7.5 Simulation results of real-time control strategy.....	197
Table 7.6 Average spare time (sec) under control strategies for transition 1.....	198
Table 7.7 Simulation results of smooth transitioning algorithm.....	206
Table 7.8 Simulation results of optimal control strategy.....	207
Table 7.9 Average recovery time (sec) under control strategies for transition	209

Table 8.1 Simulation scenarios and ratios of traffic volumes to saturation flows	218
Table 8.2 Volume/capacity ratios, x_i , for each approach.....	219
Table 8.3 Average queue lengths (veh) under normal operation.....	220
Table 8.4 Average queue lengths (veh) for each traffic ratio under normal operation.....	222
Table 8.5 EV detection distances (m) on approach 1 away from the intersection	223
Table 8.6 Simulation results of real-time control strategy.....	224
Table 8.7 Averages of simulation results for each traffic ratio under real-time control strategy.....	225
Table 8.8 Simulation results of optimal control strategy.....	226
Table 8.9 Averages of simulation results for each traffic ratios under optimal control strategy.....	228
Table 8.10 Relative measurements for each traffic ratio under real-time control strategy.....	230
Table 8.11 Relative queue lengths on EV route under real-time control strategy.	231
Table 8.12 Relative queue lengths on side road under real-time control strategy.	231
Table 8.13 Relative queue lengths on EV route under optimal control strategy...	232
Table 8.14 Relative queue lengths on side road under optimal control strategy...	233
Table 9.1 Combinations of weighting matrices.....	248

Table 9.2 Average queue lengths after transition 1	249
Table 9.3 Average queue lengths (veh) after transition 2 for traffic scenario 1 (simulation 1)	250
Table 9.4 Average queue lengths (veh) after transition 2 for traffic scenario 2 (simulation 1)	251
Table 9.5 Average queue lengths (veh) after transition 2 for traffic scenario 3 (simulation 1)	253
Table 9.6 Average queue lengths (veh) after transition 2 for traffic scenario 4 (simulation 1)	255
Table 9.7 Combinations of weighting matrices for the second simulation	268
Table 9.8 Average queue lengths (veh) after transition 2 for traffic scenario 1 (simulation 2)	269
Table 9.9 Average queue lengths (veh) after transition 2 for traffic scenario 2 (simulation 2)	270
Table 9.10 Average queue lengths (veh) after transition 2 for traffic scenario 3 (simulation 2)	272
Table 9.11 Average queue lengths (veh) after transition 2 for traffic scenario 4 (simulation 2)	274
Table 9.12 Changes in minimal average queues due to changes of P', R, and S ..	285

Table 10.1 Multiple EVSP scenarios.....	298
Table 10.2 Failure number of control strategy for multiple EVSPs.....	298
Table 10.3 Average green time for side road during multiple EVSPs.....	298
Table 10.4 Average spare time during multiple EVSPs.....	298
Table 10.5 Average queue lengths after real-time control for multiple EVSPs...	299
Table 10.6 Average queue lengths after optimal control for multiple EVSPs.....	299
Table 10.7 Relative queue lengths after real-time control.....	304
Table 10.8 Relative queue lengths after optimal control.....	307

List of Figures

Fig. 1.1 Research framework.....	9
Fig. 3.1 NEMA phase-numbering scheme.....	36
Fig. 3.2 Phase diagrams for traffic signals.....	37
Fig. 3.3 Classification of vehicles at an intersection.....	39
Fig. 4.1 Layout of intersections.....	69
Fig. 4.2 Calculation of RNC.....	70
Fig. 4.3 Tree structures for successive signal sequences.....	91
Fig. 4.4 Flow chart for determination of optimal signal sequences.....	94
Fig. 4.5 M-step models for stepwise search strategy.....	97
Fig. 4.6 Flow chart for determination of optimal control variables.....	98
Fig. 5.1 Flow chart for generation of vehicle arrivals.....	108
Fig. 5.2 An intersection layout and related traffic volumes.....	109
Fig. 5.3 Two-phase signal system.....	110
Fig. 5.4 Average queue for EV from approach 1 after transition 1.....	140
Fig. 5.5 Overall average queue for EV from approach 1 after transition 1.....	141
Fig. 5.6 Average delay time relative to base case for EV from approach 1 after transition 1.....	144

Fig. 5.7 Overall average delay time relative to base case for EV from approach 1 after transition 1	145
Fig. 5.8 Average queue for EV from approach 2 after transition 1	151
Fig. 5.9 Overall average queue for EV from approach 2 after transition 1	152
Fig 5.10 Average delay time relative to base case for EV from approach 2 after transition 1	155
Fig. 5.11 Overall average delay time relative to base case for EV from approach 2 after transition 1	156
Fig. 6.1 Situation 1: t_{fin} falls in green(1)	159
Fig. 6.2 Situation 2: t_{fin} falls in red(1)	159
Fig. 6.3 Flow chart for determination of smooth transitioning recovery process.	160
Fig. 6.4 Average queue for EV from approach 1 after transition 2	176
Fig. 6.5 Overall average queue for EV from approach 1 after transition 2	177
Fig. 6.6 Average delay time relative to base case for EV from approach 1 after transition 2	179
Fig. 6.7 Overall average delay time relative to base case for EV from approach 1 after transition 2	180
Fig. 6.8 Average queue for EV from approach 2 after transition 2	184
Fig. 6.9 Overall average queue for EV from approach 2 after transition 2	185

Fig. 6.10 Average delay time relative to base case for EV from approach 2 after transition 2.....	187
Fig. 6.11 Overall average delay time relative to base case for EV from approach 2 after transition 2.....	188
Fig. 7.1 Layout of subject coordinated route.....	192
Fig. 7.2 Average queues at intersection 1 after transition 1.....	200
Fig. 7.3 Average queues at intersection 2 after transition 1.....	201
Fig. 7.4 Average queues at intersection 3 after transition 1.....	202
Fig. 7.5 Average queues at intersection 4 after transition 1.....	203
Fig. 7.6 Average queues along the route after transition 1.....	204
Fig. 7.7 Average queues at intersection 1 after transition 2.....	210
Fig. 7.8 Average queues at intersection 2 after transition 2.....	211
Fig. 7.9 Average queues at intersection 3 after transition 2.....	212
Fig. 7.10 Average queues at intersection 4 after transition 2.....	213
Fig. 7.11 Average queues along the route after transition 2.....	214
Fig. 8.1 Critical approaches and relative traffic volumes.....	217
Fig. 8.2 Relative queues on approach 1 after transition 1.....	234
Fig. 8.3 Relative queues on approach 3 after transition 1.....	235
Fig. 8.4 Relative queues on approach 2 after transition 1.....	236

Fig. 8.5 Relative queues on approach 4 after transition 1	237
Fig. 8.6 Relative queues on approach 1 after transition 2	238
Fig. 8.7 Relative queues on approach 3 after transition 2	239
Fig. 8.8 Relative queues on approach 2 after transition 2	240
Fig. 8.9 Relative queues on approach 4 after transition 2	241
Fig. 9.1 Average queues on approaches 1 to 4 for traffic scenario 1 (simulation 1)	257
Fig. 9.2 Average queues over all approaches for traffic scenario 1 (simulation 1)	258
Fig. 9.3 Average queues on approaches 1 to 4 for traffic scenario 2 (simulation 1)	259
Fig. 9.4 Average queues over all approaches for traffic scenario 2 (simulation 1)	260
Fig. 9.5 Average queues on approaches 1 to 4 for traffic scenario 3 (simulation 1)	261
Fig. 9.6 Average queues over all approaches for traffic scenario 3 (simulation 1)	262
Fig. 9.7 Average queues on approaches 1 to 4 for traffic scenario 4 (simulation 1)	263

Fig. 9.8 Average queues over all approaches for traffic scenario 4	
(simulation 1).....	264
Fig. 9.9 Average queues on approaches 1 to 4 for traffic scenario 1	
(simulation 2).....	276
Fig. 9.10 Average queues over all approaches for traffic scenario 1	
(simulation 2).....	277
Fig. 9.11 Average queues on approaches 1 to 4 for traffic scenario 2	
(simulation 2).....	278
Fig. 9.12 Average queues over all approaches for traffic scenario 2	
(simulation 2).....	279
Fig. 9.13 Average queues on approaches 1 to 4 for traffic scenario 3	
(simulation 2).....	280
Fig. 9.14 Average queues over all approaches for traffic scenario 3	
(simulation 2).....	281
Fig. 9.15 Average queues on approaches 1 to 4 for traffic scenario 4	
(simulation 2).....	282
Fig. 9.16 Average queues over all approaches for traffic scenario 4	
(simulation 2).....	283
Fig. 10.1 EV(i+1) arrives before EV(i) crosses intersection.....	288

Fig. 10.2 Remaining transition time for EV(i+1).....	288
Fig. 10.3 EV(i+1) arrives during recovery period for EV(i).....	292
Fig 10.4. Flow chart of control algorithm for multiple EVSP occurrences.....	294
Fig. 10.5 Relative queue lengths after real-time control.....	305
Fig. 10.6 Relative queue lengths after optimal control.....	308

List of Acronyms and Symbols

EV	Emergency Vehicle
EVSP	Emergency Vehicle Signal Preemption
RNC	Remaining Normal Cycle time, which is needed to get traffic signal back to coordinated normal operation immediately after an EVSP
STI	Safety Time Interval between an EV and leading or following general traffic

AR1 = all red interval for main road

AR2 = all red interval for side road

$a_j(i)$ = number of vehicle arrivals on approach j during i^{th} time step

C = period of time for one traffic signal cycle under normal operation

$g_{1\min}$, $g_{1\max}$ = minimal and maximal continuous green times for main road, respectively

$g_{2\min}$, $g_{2\max}$ = minimal and maximal continuous green times for side road, respectively

g_j = green time for approach j during one cycle under normal operation

$g_j(i)$ = green time for approach j during i^{th} time step

L = distance of an EV detection away from an intersection

L_{QS} = average standing queue length at an intersection plus 2 times standard deviation under peak hour traffic conditions

L_{QM} = average equivalent queue length formed by vehicles moving on an approach to an intersection (including moving queue) plus 2 times standard deviation under peak hour traffic conditions

N = total number of time steps for the period of transition 2

q_j = average queue length on approach j under normal operation

$q_j(i)$ = queue length on approach j during i^{th} time step

$r_j(i)$ = red interval for approach j during i^{th} time step

S_{dis} = discharge speed of an approach to an intersection (which reflects the effect of queuing on the approach and is generally lower than the mean speed of traffic on the approach)

$seq(j)_i = j^{th}$ signal indication sequence for i^{th} time step

S_{EV} = EV operating speed

s_{gj} = saturation discharge rate for green time on approach j

S_{mean} = mean speed of general traffic

STI = Safety Time Interval between an EV and leading or following general traffic

s_{yj} = saturation discharge rate for yellow interval on approach j

T = period of time for each time step or for the whole recovery process

$TN(i)$ = total number of possible signal indication sequences for i^{th} time step

t_{clear} = period of time for all the vehicles ahead of an EV to clear an intersection

t_{EV} = period of time for an EV to traverse the distance L

t_{π} = period of time for real-time control strategy during transition 1

t_f = period of time for fixed held time control strategy during transition 1

t_{green} = period of time elapsed from the beginning of green of current cycle

t'_{green} = green time available for cross road during transition 1

t_{startup} = startup delay at an intersection

$Y1$ = yellow interval for main road

$Y2$ = yellow interval for side road

$y_j(i)$ = yellow interval for approach j during i^{th} time step

$\delta_1(i) = 1$ for signal switchover from main road to side road during i^{th} time step; 0 for no such switchover

$\delta_2(i) = 1$ for signal switchover from side road to main road during i^{th} time step; 0 for no such switchover

Δt = time headway between two successive EVs

List of Appendices

Appendix A	Computer Program for Generation of Vehicle Arrivals	343
Appendix B	Computer Program for Determination of EV detection distance..	344
Appendix C	Computer Program for Normal Operation.....	345
Appendix D	Computer Program for Fixed-held Time Transitioning.....	348
Appendix E	Computer Program for Smooth Transitioning.....	361
Appendix F	Computer Program for Real-time Control.....	387
Appendix G	Computer Program for Optimal Control.....	405
Appendix H	Computer Program for Coordinated Route Control.....	431
H.1	Normal operation.....	431
H.2	Fixed held time transitioning for transition 1.....	441
H.3	Real-time control for transition 1.....	451
H.4	Smooth transitioning for transition 2.....	466
H.5	Optimal control for transition 2.....	489
Appendix I	Computer Program for Multiple EVSP Occurrences Control.....	504

Chapter 1 Introduction

1.1 Overview

Traffic congestion is a worldwide severe problem, especially in urban areas, confronted by general transportation users and traffic professionals. The consequences of traffic congestion include environmental pollution, wasted fuel consumption, elongation of travel time, and even accidents. According to a recent research by Texas Transportation Institute (TTI, 2003), the total congestion bill for the 75 areas studied in 2001 came to US\$69 billion, which was the value of 3.6 billion hours of delay and 5.7 billion gallons of excess fuel consumed.

While a variety of ways to relieve traffic congestion are being introduced and improved, such as components of intelligent transportation systems (ITS), namely advanced traffic management systems (ATMS) and advanced traveller information systems (ATIS), traffic demand continues to increase at a rate higher than that of adding capacity. There are many contributors to traffic congestion, and one of them is the inevitable emergency vehicle (EV) operations in a modern society.

Generally speaking, an emergency can be any situation that is caused by an incident, an accident, an intentional act, or the forces of nature, and it constitutes a danger to life or

property (Watson, 2002). An emergency operation is associated with such a situation so as to prevent injury, loss of life and property damage. During each emergency operation, the task for the traffic/transportation professional is to create and maintain safe and smooth traffic flow in the existing road network (Shibuya et al., 2000). Various traffic management and control strategies and activities are adopted to ensure an efficient and effective traffic system. Most frequently used strategies include information dissemination, posting messages on VMSs (variable message signs), contraflow roadway segments, ramp metering, diversion of traffic, signal retiming, etc. (Ozbay and Kachroo, 1999)

While general transportation users should always be taken into consideration for emergency operation, the priority of transportation management and traffic control is to ensure the success and smoothness of emergency vehicle (EV) operations in a safe and quick manner. Generally, from daily small incidents to major disasters, police, fire fighters and ambulance are required to reach the emergency scene for initial lifesaving and scene stabilization. Therefore, emergency vehicles (EVs) usually include fire engines, ambulance vehicles, and police cars as they are dispatched for incidents, accidents, and other emergency cases. Because of the urgent characteristics of EV operations, they are required to reach the scene at the earliest possible time. Unfortunately, existing congested traffic conditions make it difficult for EVs to arrive at the scene in a quick and safe manner. On the other hand, it is not uncommon that EV operations also cause worse

congested traffic situations, even incidents or accidents, for which they were dispatched. According to a research study (USDOT, 2002), more than 643 EV crashes involving one or more fatalities occurred around the United States of America from 1994 to 2000. Another research result of the United States of America (TRB, 2002) indicated that in 1999, traffic crashes caused more than 50% of police officers killed on duty. Nearly 10,000 police cars, 2,000 fire trucks and more than 3,000 other service vehicles were struck while responding to traffic incidents.

With the trend of elderly society and the growing number of incidents and accidents, more and more EVs are dispatched, and EV operations are expected to grow unless some techniques are developed to deal with the problems. For instance, as statistics indicated (Steinbachs, 2004), paramedical emergency calls from residents had increased dramatically in Ottawa and the number was expected to continue to increase in 2004. While the goal for paramedical response time (the 90th percentile) announced years ago was to take about nine minutes to respond to calls in the city's urban area, the actual urban response times in May 2004 was eleven minutes and twenty seconds. Consequently, the control of EV operations emerges as an important area of traffic research and practice.

1.2 Research objectives and scope

From the perspective of traffic control, there are two levels in traffic research to

ensure that EV operations are accomplished smoothly. The first one is at the network level, and the second one is at individual intersections (intersection level). The task for the network level is to determine the quickest route from the origin of an EV to its destination, so that the EV can reach the scene at the earliest possible time along the selected route. The requirement at the intersection level is to ensure that the EV can pass through intersections along the route in a quick and safe manner.

Considering the importance of EV operations in saving lives and reducing property damage, preferential treatments are devised to ensure the success of EV operations. One of the preferential treatments is emergency vehicle signal preemption (EVSP) at the signalized intersection (Deeter et al., 2001). When an EV approaches a signalized intersection, it requests preferential signal indication for crossing the intersection. Once the EVSP system receives the request it interrupts normal signal operation to serve the call from the EV. After the EV has passed through the intersection, the EVSP system gets the signals back to normal operation. Because the preemption causes disruption of normal signal operation, especially in a coordinated signal system, effective strategies are needed to control the transition of traffic signals in a safe mode and reduce the impact on general traffic to a minimal possible extent.

If it is possible to find a way to let an EV clear each intersection at operating speed, and keep the disruption on general traffic at an acceptable level at the same time, the task for the network level in most situations will be simply to select the geometrically shortest

route between the origin of the EV and its destination. Therefore, the attention in this research is focussed on the traffic signal control at the intersection, i.e., real-time and optimal control strategies of signal timing transition for EV signal preemption (EVSP).

Actually, Intelligent Transportation System (ITS) technologies have rapidly developed to deal with the problem of EVSP. There already exist several commercial systems allowing EVs to request and receive green indications as they are en route to emergencies. Leading examples of traffic signal preemption system include OPTICOM system from 3M Corporation Inc. (Nelson, 2003), STROBECOM system from Tomar Electronics Inc. (2005), and MIRT system from Platinum One Net Inc. (2005).

A typical EVSP system consists of a vehicle-mounted signal emitter, EV sensors (detectors) mounted on or near traffic signals, a traffic signal controller, and traffic signal indicators. When an EV is en route to an emergency, its emitter sends signals for preemption. The signal can be strobe lights, siren sounds, radio waves, infrared beams, microwaves, etc. After the EV approaches within the operational range of EV sensors, the signals are received by EV sensors. Once EV sensors are triggered by the preemption emitter, it activates a control mechanism, which is stored in the traffic signal controller. According to the control mechanism, the signal controller instructs signal indicators to interrupt normal operation and provide green indication to the EV until it crosses the intersection. After it has passed through the intersection, the communication between the preemption emitter and EV sensors is lost, and traffic signals then return to normal

operation.

Although optical and sound based EVSP systems are widely adopted, they are limited in range. They must have a clear "line of sight" between signal emitter and EV sensors, and are highly vulnerable to noise and bad weather (Bachelder and Foster, 2004). Global Positioning Systems (GPS) are also used to obtain the position, speed, and heading of EVs equipped with GPS units, and the information is used to provide a green wave for EVs en route to emergencies (Thompson and Nicholls, 1997). However, GPS based EVSP systems are often ineffective in large cities because of occultation of GPS satellite signal by large structures (Bachelder and Foster, 2004).

In order to overcome the difficulties for optical-based, sound-based, and GPS-based systems, Bachelder and Foster (2004) proposed two inductive loop-based EVSP systems, named "Car-active" and "Car-passive" systems. In the Car-active system, each participating EV would be equipped with a computer and a radio transceiver. Whenever an EV passes over a traffic loop, the loop transceiver would transmit a signal identifying the location of the loop and the direction of travel. After the EV transceiver receives the position signal, the computer in the EV would use the signal along with the speed and heading of the EV to calculate its current position. At time intervals of one second, the EV transceiver would broadcast the updated estimation of position to loop receivers at neighbouring intersections. The signal controller would then determine when to provide green indications to the EV.

In the "Car-passive" system, a passive radio transponder would be installed on the underside of a participating EV, and it would be energized by a signal, at a frequency of 450 MHz, radiated via a traffic loop, whenever the EV passes over the loop. Once energized, the EV transponder would transmit a signal of vehicle identification number at a carrier frequency of 900 MHz. Upon detection of an participating EV at a loop at a given intersection, the traffic signal controller would change normal traffic signal timing plans to EVSP operation at the next or next few intersections until the EV passes through or until a specified time elapsed.

Although a variety of EV detection systems have been developed, the control mechanism, which alters the traffic signal timing plans in a safe and effective manner, is not well studied and documented. It is assumed in this research that all hardware used for EV operations are in place and communication among individual devices is assured. Therefore, only control strategy of signal timing and corresponding algorithm and software implementation will be studied for EV operations.

The main objectives of this research include:

- (1) establishment of a real-time signal control strategy to ensure the clearance of the EV at the intersection without stopping or reducing operating speed, and to decrease the impact on general traffic to a minimal extent;
- (2) development of an optimal control model for signal timing to minimize traffic queues at the intersection during a shortest possible period of recovery time after the EV

clears the intersection; and

- (3) programming of corresponding computer software for simulation of traffic situations during EV operations and verification of the effectiveness and efficiency of proposed signal control strategies.

1.3 Research approaches and methodologies

The research framework is presented in Figure 1.1.

The prerequisite for the success of a signal control strategy is that the controller can receive the information about the approaching EV in advance long enough before the EV arrives at the intersection. However, it is not always true that the longer the notification period the better the control effectiveness, because of the stochastic behaviour of individual vehicles ahead of the EV and unpredictable disruptions from road side. A suitable choice is to determine a minimum EV detection distance away from the intersection based on statistical values of queue length under peak hour traffic conditions, so that the controller has enough time to clear the way for the EV under most traffic conditions. During this notification period from the instant an EV is detected to the moment the EV clears the intersection, there are two tasks for the EV signal preemption system to complete. First, the traffic indication must be switched to the EV road from the cross road after a suitable interchange interval if green signal is facing the cross road

Fig. 1.1 Research framework

when the controller receives the call from the approaching EV. Second, the green time for the EV road should be long enough to clear the traffic queue ahead of the EV at the intersection. Therefore, the EV detection distance will be determined by peak hour traffic volume at the EV approach and normal signal sequences.

After the EV detection distance is determined, the notification period of the EV's approaching is also fixed. However, it is unnecessary to allocate all the notification period, minus any possible switchover time as green time for the approaching EV if there is no or a short traffic queue at the intersection when the EV is detected. Under this situation the cross road can share the notification period with the EV road to reduce traffic queues on the cross road, so long as the share-out ensures the clearance of queue ahead of the EV before it reaches the intersection. The switchover of signal indications should therefore be determined dynamically according to current traffic conditions in real-time mode.

Once the EV has cleared the intersection the recovery time begins to get traffic signals back to the background coordinated operation pattern. The requirements for this recovery time are:

- (1) as short as possible;
- (2) suitable signal sequences for safe vehicle operations; and
- (3) minimization of impact on general traffic at the intersection.

The optimal control theory was applied to this problem and a time-invarying

dynamic system was then established to formulate the traffic signal control strategy. However, a closed-form solution could not be deduced from the optimal control model because of the interrelationship between sequences of greens, yellows and all red intervals in successive time intervals. A two-phase algorithm was developed to solve the optimal control model. First, a low bound of the optimal control system was solved by excluding the interrelationship between successive signal sequences. During the second phase, practical solution for the optimal control system was obtained by introducing the interrelationship neglected in the first phase. A stepwise search strategy in an M-step model is also adopted to reduce the search space so that the optimal control signal sequences can be produced in a real-time manner.

After the real-time control strategy and the optimal control strategy were established, computer software was developed in MATLAB environment for traffic control simulation. First, simulations were conducted as proof of concept to compare the real-time control strategy and the optimal control strategy with the fixed held time transitioning and the smooth transitioning for transition 1 and transition 2, respectively. Second, simulations were performed at different saturation levels to verify the effectiveness and efficiency of the control strategies for different traffic conditions. Then, the optimal control strategy was simulated using different weighting matrices to investigate their impact on control efficiency. At last, the real-time control strategy and the optimal control strategy were enhanced to control multiple EVSP occurrences and

simulations were conducted.

1.4 Organization of thesis

In Chapter 2 a review of literature is presented in terms of previous research related to this study. Chapter 3 first introduces how to determine the minimum distance of the EV detection away from the intersection based on statistical values of queue lengths under peak hour traffic conditions. Then a real-time control algorithm is developed to dynamically allocate green time to the EV road and the cross road according to current traffic conditions at the intersection during the period of time, termed transition 1, from normal operation to EV operation. If the minimum distance requirement of EV detection cannot be met, it is possible that the real-time control algorithm cannot discharge the queue at an intersection before an EV reaches the intersection. This situation is discussed in Chapter 3. Considering the difference between the single intersection control and the coordinated route control, the enhancement of the real-time control algorithm for coordinated route is also described in Chapter 3. The optimal control model for transition 2, from EV operation back to normal operation, is formulated in Chapter 4. A relaxation algorithm and a stepwise search strategy are then described to obtain practical signal sequences, which minimize the queues at the intersection after the transitioning is completed.

Chapter 5 presents simulation results of the real-time control strategy for transition 1 and the "fixed held time" transitioning as well. Comparison and analysis of the simulation results are then presented. Simulation results for transition 2 under the optimal control strategy and the well-known smooth transitioning algorithm are described in Chapter 6, and then the results are analyzed and compared with each other. Based on the success of our control strategies on a single intersection described in Chapters 5 and 6, simulations of a coordinated route were conducted to confirm the effectiveness and efficiency of our control strategies for route control. Chapter 7 describes the simulation results for a coordinated route control.

Traffic volumes used for simulations in chapters 5, 6, and 7 had specific combinations. Any change in traffic volumes will likely affect the effectiveness of the transition algorithms. Simulations were conducted to investigate the way and the extent of the effect. Traffic conditions for simulation were set at different levels of saturation, including under saturation and over saturation. Chapter 8 presents the simulations and analysis of simulation results.

In the application of the optimal control strategy, we need to select weighting matrices to weight terminal state, intermediate state, and control variables differently for the transitioning from EVSP to normal operation. In chapters 5 through 8, identity matrices were used for simulation as we had a limited knowledge of the extent the weighting matrices would affect the control strategy. In order to identify the impact of

these matrices on the effectiveness of the optimal control strategy and provide suggestions on the selection of the weighting matrices for different traffic conditions. Chapter 9 selects different weighting matrices for simulations, and then simulations results are discussed.

In previous chapters, we only consider a single preemption occurrence. If multiple emergency vehicles preempt at closely timed intervals, the control process of traffic signal timing transition should be adjusted. In Chapter 10, enhancements are made to the real-time control strategy and the optimal control strategy for multiple EVSP occurrences, and then simulations and result analysis are presented.

Chapter 11 summarizes the real-time control strategy for transition 1, the optimal control strategy for transition 2, and the simulation results under the control strategies. Based on the analysis and comparison of simulation results, conclusions are extracted. At last, recommendations for further studies are also included in Chapter 11.

Computer programs are included as Appendices A to I.

Chapter 2 Review of Literature

2.1 Overview of emergency vehicle signal preemption systems

Emergency vehicle signal preemption (EVSP) technique has been used for a long time (Fox et al., 1998; Deeter et al., 2001). Its objective is to give preferential treatment to emergency vehicles (EVs), such as fire engine, ambulance, and police car, at signalized intersections, so that EVs can pass through intersections quickly and safely, and can reach the scene at the earliest possible time.

During the emergency operation, other vehicles are supposed to give the right-of-way (ROW) to EVs. EVs drive at high speed and many traffic regulations are ignored (Shibuya et al., 2000). Many older EVSP systems were designed to provide continuous green signals along fixed EV corridors, usually fire routes. The green wave routes are implemented by pushing a button in the fire station (or by remote request from the emergency service control centre), and the fire route stays in effect for a specified period of time or until the switch is tuned off (Federal Highway Administration, 1996).

More modern EVSP systems are implemented through real-time control strategies. Special detectors are installed at the signalized intersection to receive coded signals from EVs, and then the EVSP systems preempt regular signals to allow a green signal in the direction needed by the approaching EVs (Federal Highway Administration, 1996).

According to Ken Fox et al.(1998), a GPS-based EVSP system, RESPONSE, has been developed and installed in Ottawa, Canada (Thompson and Nicholls, 1997). The location, speed and heading of EVs are determined using a GPS unit on each EV, and the information is used to estimate the time of arrival at each signalized intersection. The traffic control system is able to use this information to automatically preempt local controller to provide the ROW.

In many traffic signal control systems, EVSP is incorporated with transit priority and the control strategy is the same for both EV and transit except that the EV has a higher level of priority (Fox et al., 1998; Nelson et al., 2000).

As mentioned previously, older EVSP systems adopted pre-timed control strategy, and current traffic condition was not taken into consideration for changing traffic signal. The development of traffic signal control strategies is a function of computation and communication technologies. With the continuously increasing capacity to price ratio of computer, detector, and communication instruments, more and more attention is paid to traffic-responsive traffic signal control model (Dion and Yagar, 1996; Urbanik, 2003). A variety of systems were developed to control traffic signal in response to actual traffic conditions at signalized intersections. Representative systems include SCOOT (Hunt et al, 1981; Hunt et al., 1982), SCATS (Lowrie, 1982), OPAC (Gartner, 1983; Gartner et al., 2002), PRODNY (Henry and Farges, 1989), MOVA (Vincent and Young, 1986), SPPORT (Yagar and Han, 1994), and UTOPIA (Donati et al., 1984; Mauro and Di Taranto, 1989; Mauro, 1991), etc.

Although these systems do not have identical formulations, their objectives usually can be represented as an objective function, subject to some constraints. The

objective function is a weighted combination of delay, stop, etc., and the constraints consist of maximum/minimum phase durations, etc. The signal control problem is then formulated as an optimal control problem, and a variety of optimization technologies can be used to obtain the optimal control strategy. These optimization techniques can be broadly classified into the following two groups.

- 1) Mathematical programming methods, such as dynamic programming; linear programming; integer programming; linear quadratic method; etc. (Gartner et al., 2002; Lin, 2001)
- 2) Rule-base heuristic search methods, such as decision-making model (Yagar and Han, 1994); control principle model (Janos, Melanie and Furth, 2002); etc.

Mathematical programming models usually require more computational effort, and some time it is difficult to implement for real time control, especially when there are many control parameters, such as in cases of complex intersection geometric layouts and corresponding phase combinations. On the other hand, heuristic models can provide rapid approximate solutions to the optimal problem when applied properly, although it is sometime difficult to assess if a heuristic procedure really achieves global optimization. Usually, a heuristic procedure can be judged on the basis of practical experience and established theory as well (Lin, 2001).

In some optimization models, such as OPAC, the network control problem is broken into much smaller local control problems so that the local controllers can optimize signal timings in parallel. This parallel processing enables the control of the network independent of the size of the network and therefore enables the control system

to adjust signals in true real time manner. While each intersection is treated as an independent optimization problem, the progression in the network is maintained by coordinating adjacent signals. The coordination with adjacent intersections is achieved by peer-to-peer communication between adjacent intersection controllers. Each local controller provides its neighbours with its current status, such as signal display, queue vehicles, etc., and the forecasted timing strategy.

Theoretically, the above optimization models are applicable to transit priority and EVSP as well. Practically, few systems have fully succeeded in applying these models to EVSP due partly to the difference in nature of transit operation and emergency vehicle operation. Transit priority and EVSP are in fact different processes. "While signal priority modifies the normal signal operation process to better accommodate transit vehicles, preemption interrupts the normal signal process for special events such as an approaching train or responding fire engine." (Advanced Traffic Management Systems Committee et al., 2002). It is believed that EVSP causes significant disruptions in traffic and its perceived negative impact results in the general reluctance to deploy preemption technology by governmental agencies and traffic signal maintenance professionals. While current research focuses on the impact of transit priority (Rakha and Zhang, 2004; Daniel, Lieberman and Srinivasan, 2004; Dion, Rakha and Zhang, 2004, Baker et al., 2004), the impact of EV preemption has not been well documented. At the same time, with the increasing number of dispatches of EVs, the impacts of EVSP on the general traffic are also expected to increase.

However, EVSP is an important and indispensable part of transportation system and modern society. The problem for us is how to minimize the impacts of EVSP on general traffic instead of whether to implement EVSP.

2.2 Commonly used control strategies for EVSP operation

While many EVSP systems are used worldwide, how to minimize the impacts of EVSP on general traffic is not well documented. Specifically, few research results are available on the control mechanism of signal timing transition out of and then back to normal operation from EVSP operation.

When an EV approaches an intersection, it requests signal preemption. For the commonly used fixed held time control strategy, after the EVSP system receives the request, it interrupts normal operation and provides a specified period of green time for the EV route. If such a green time is not long enough to clear the EV, another period of green time will be provided. This process will continue until the EV crosses the intersection.

After the EV has crossed the intersection, the signal phase needs to be brought back to normal operation in a safe and quick manner. The commonly used transition algorithms for this recovery period include (Nelson et al., 2000):

- (1) Smooth. The local cycle length is adjusted by adding up to a maximum of 20% or subtracting up to a maximum of 17% of the cycle length per cycle. The option, which is quickest to recover the signal operation to coordination pattern and satisfies safety requirements for every phase, is chosen

- (2) Add Only. The local cycle length is adjusted by adding up to a maximum of 20% of the cycle length per cycle. There is no need to check safety requirements for this transitioning because no phase will be cut short.
- (3) Dwell. The signals at each intersection stay in certain phases for up to a maximum dwell time until coordination is reached. This usually quickest recovery procedure probably also results in significant congestion on side roads, especially during saturated conditions.

To quantify the effects that EV preemption has on coordinated signal networks, a preemption study was performed in 1998 along a corridor with three coordinated intersections over a 1.5 mile section of road (Bullock et al., 1999). The study indicated that the preemption operation had minor impact on general traffic with the largest increase in average travel time on the order of only 1 to 2% during the morning peak. This may be caused by the relatively long spacing between intersections and the long cycle length.

In order to evaluate different transition procedures used to resynchronize preempted signals, and to estimate the impacts of different distances between intersections and different cycle lengths as well, another study was conducted on a principal arterial with four coordinated intersections (Nelson et al., 2000). The study results showed that among the three transition algorithms (i.e. smooth, add only, and dwell), the smooth transitioning algorithm performed the best. In this study the parameters for 20% addition, 17% subtraction, and 50% cycle length of dwell time were fixed.

In SCOOT (Version 4.2, 1998), an improved recovery logic for EV or Light Rapid Transit (LRT) operation is provided. The optimization for general traffic control is suspended during fire priority, and fast recovery to normal conditions is implemented at the end of priority period (Siemens, 2000; Bretherton et al., 1998). But the implementation detail is not described.

Mussa and Selekwa (2004) proposed a traffic signal timing transition procedure based on the quadratic optimization method. The optimization problem of transition is defined as to determine the optimal path for moving the coordination parameters (cycle length, phase split, and offset) from one known optimal timing plan to another, and the objective is to minimize the sum of squares of deviations of coordination parameters from the optimal values during the transition. The transition is modelled as a linear dynamic process and optimal control techniques are used to determine the time step size (i.e., time step interval) and the optimal number of steps required to complete the transition. The result of the optimization procedure shows that the number of steps, N , required to achieve synchronization is directly proportional to the number of intersections in the coordinated corridor, and the optimal transition step length is obtained by dividing the range of signal coordination parameters between the target timing plan and current timing plan into N equal parts.

This method is primarily used for signal transition between time-of-day (TOD) timing plans, in which the starting timing plan and the ending timing plan are known. For the problem of EVSP, the EV entry time and the period of EV operation cannot be pre-determined, and therefore the signal timing transition for EVSP cannot be modelled as an optimization process from one known signal timing plan to another.

To develop a route-based control strategy for EV operation, a study was conducted recently by E.L. Kwon et al. (Kwon and Kim, 2003). First, an optimal route from a given origin to a desired destination is selected by using Dijkstra's algorithm so that the travel cost on the route has a minimal value under current traffic conditions. Then the control algorithm preempts signals at each intersection on the selected route sequentially by choosing a specific phase combination for each intersection. Once the EV passes through an intersection, a recovery process begins to recover the original offset and cycle length at that intersection by adjusting the "walk" time interval for pedestrians in each phase.

Because the main purpose of the research was to provide safety to pedestrians crossing the intersection, no consideration was given to current traffic conditions at individual intersections, and only location of the EV and the state of signal phase for each intersection were taken into account when granting the preemption at the intersection. To overcome the difficulty in determining the transition period from the beginning of EVSP operation to the instant the EV crosses the intersection, a fixed time was pre-selected for each intersection.

It should be pointed out that signal timing transition processes are not solely determined by traffic demand at the intersection under some unusual situations. Hanbali and Fornal (2004) presented some constraints that standard emergency vehicle preemption strategies could not deal with. Particular types of preemption strategies were introduced at signalized intersections near a lift bridge. Bridge preemption has priority over fire preemption if the lift bridge and a fire engine request preemption at the same time, i.e., the lift bridge will be granted the preemption and the fire engine preemption

is delayed until the bridge preemption is completed. In another unusual situation, if two fire engines simultaneously approach an intersection on conflicting approaches, a particular confirmation light is used to indicate an active preemption with granted right of way (steady confirmation light) or to indicate that a conflicting approach is receiving priority control with right of way (flashing confirmation light). Except aforementioned strategies to solve conflicts among different preemption strategies at an intersection, all the other control policies were conventional and predetermined. After receiving a fire preemption request at an intersection, traffic signal was turned to and held in green for a fixed period of time to serve the fire engine. However, recovery policy was not mentioned in this article.

2.3 Potential possibilities for further research

Actually, the impact of EVSP comes from two periods. First, the controller must clear the path for the EV before the EV enters the intersection. This operation usually interrupts the normal coordinated signal control. Besides, there must be a safety time interval (STI) between the instant of the path being cleared and the instant of the EV entering the intersection. On the other hand the STI could be viewed as a waste of traffic capacity resource since no traffic is utilizing the intersection during the period. The longer the STI, the more delay will be resulted for general traffic.

Second, after the EV passes through the intersection, the traffic signals must be resynchronized and recovered to coordination pattern. The impact of EVSP depends on traffic condition and recovery method. While the traffic condition is beyond the control

of the traffic engineer, the recovery method can be carefully designed and chosen to minimize the impact of EVSP.

Consequently there are two transition periods instead of only one transitioning back to coordinated traffic signal control which attracts current research focus. For the first transitioning, the objective will be to ensure that an EV can cross the intersection without delay and a specified STI (say 0-2 seconds) is provided through (1) a suitable advance notice of the EV's approaching; and (2) a real-time traffic queue control strategy.

For the second transition period, i.e. the commonly named recovery transition, the objective is not just to recover quickly to coordination pattern. The impact of transition procedure itself should be analyzed carefully as well. It can be rather safe to estimate that a transition method using a fixed parameter of addition, subtraction, or dwell time, as in Nelson et al. (2000), cannot accommodate the fluctuation in traffic demand at different intersections during different periods of time in a coordinated signal network.

In order to meet this challenge, a dynamic control mechanism was developed in this research project to solve the problem. The control strategy was achieved by dynamically optimizing the signal timing according to current traffic condition, subject to the constraints of maximal allowable queue length and maximal/minimal green time, and by maintaining background cycle length and offset to coordinate the downstream intersection with its upstream intersection subsequently throughout the signal network.

2.4 Why is Dynamic Programming not used for this research?

Dynamic Programming (DP) is a global optimization strategy/method for multistage decision process. After a decision problem is divided into a sequence of stages, sequential decisions are made one after another for each corresponding stage.

The principle of optimality can be enunciated as follows. “An optimal policy has the property that whatever the stage the system is in at a particular stage, and whatever the decision taken in that stage, then the resulting decision are optimal for the subsequent stage.” (Bellman 1957; Smith, 1991). It implies that “Given the current state, an optimal policy for the remaining stages is independent of the policy adopted in previous stages.” (Hillier et al., 1967).

DP applications to transportation problems have a long history. Actually, the most commonly used problem to illustrate the features of DP is the “stagecoach problem”, which concerns the optimal decision to travel from one place to another one (Wagner, 1963). In terms of optimal control of traffic signal, DP techniques were first adopted by Grafton and Newell (1967), and Robertson and Brotherhood (1974). Motivated by their work, the first version of OPAC, one of the most famous methods for optimal signal control, was developed by Gartner (1983) based on DP. In OPAC, the optimal control of traffic signal at a signalized intersection is formulated as follows.

- (1) The optimization process is decomposed into N stages, and each stage represents a discrete time interval.
- (2) At stage i , transformations, T_i , R_i , from input to output are established.

$$O_i = T_i(I_i, A_i, X_i) \quad \text{and}$$

$$r_i = R_i(I_i, A_i, X_i)$$

where O_i = output state vector;

r_i = economic return (cost) output;

I_i = input state vector;

A_i = vehicle arrivals; and

X_i = input decision variable.

The state vector of the intersection consists of signal state (green or red) and queue length on each of the approaches. The input decision variable, X_i , indicates whether the signal is to be switched at this stage or remain in its present stage ($X_i = 1$ or 0 , respectively). The return cost output r_i is the intersection index of performance (the total delay time), which has to be minimized.

- (3) DP optimization is carried out backwards, i.e., starting from the last time interval and backtracking to the first, and the recursive optimization function is given by

$$f_i^*(I_i) = \min_{X_i} [R_i(I_i, A_i, X_i) + f_{i-1}^*(I_i, A_i, X_i)]$$

- (4) When the optimization is completed at stage $i = 1$ with given stage I_1 , the optimal policy can be retraced and it consists of the optimal sequence of switching decisions (X_i^* , $i=1, 2, \dots, N$) at all stages of the optimization process.

Although this method can produce the theoretically optimal control strategy for each input state, it cannot be used for real-time implementation because of two problems. First, it involves extensive amount of processing to generate optimized policies for all possible combinations of initial conditions at each stage of the control period, and only

optimum policy would be implemented in practice. Second, this method assumes that accurate information on arrivals for the entire control period is available, and it is difficult to satisfy such an assumption in practice.

As a result, in OPAC's other versions DP is replaced with other optimization techniques.

Another famous DP-based optimization algorithm, COP, was proposed by Sen and Head for real-time control of a single intersection (Sen and Head, 1997). The formulation of COP can be represented as follows.

- (1) Each phase in a cycle is used as a stage in the DP calculation, and the optimization process is divided into T discrete time steps.
- (2) State variable, s_j , is defined as the total number of time steps that have been allocated after stage j has been completed. Next, a discrete set, $X_j(s_j)$, is used as the values of feasible control decisions for stage j .

$$X_j(s_j) = \{0\} \text{ if } s_j - r < \gamma \quad \text{or}$$

$$X_j(s_j) = \{0, \gamma, \gamma+1, \dots, s_j - r\} \text{ if } s_j - r = \gamma$$

where r = effective clearance interval (integer number of time steps);

γ = minimum green time (integer number of time steps).

- (3) Define a performance measure of state j (queue length, delay, stop, etc.), $f_j(s_j, x_j)$, given state s_j , and control x_j .
- (4) Optimization is carried out in a forward recursive manner, i.e., the recursion is performed in a forward fashion, starting from the initial decision and then a value function for any stage represents an accumulated measure of effectiveness for the current and all previous stages. Calculation is conducted for value function, $V_j(s_j)$,

$$V_j(s_j) = \min_{x_j} [f_j(s_j, x_j) + V_{j-1}(s_{j-1}) \mid x_j \in X_j(s_j)]$$

and an optimal solution, $X_j^*(s_j)$, to the above problem is recorded.

COP makes the same assumption with OPAC, i.e., it assumes that accurate predictions of vehicle arrivals for the entire control period can be provided. This assumption prevented the DP-based OPAC from real-time implementation 20 years ago, and now it seems that it remains as a difficulty for COP to overcome. While arrivals probably can be predicted accurately for a short period, it is not reasonable to expect accurate estimation for a long period. Maybe this is one reason COP chose a 30-40 second time horizon for simulation. Its requirement for data of vehicle arrivals for the entire control period before global optimal policies can be made also precludes the opportunity for correction of future control policies as more recent real-time data become available.

Perhaps this is a common characteristic that is unavoidable for DP-based system. For example, in the algorithm by Fang and Elefteriadou (2005) for diamond interchange control, the control process is divided into optimization horizons and each horizon consists of only 10 seconds. Optimization is performed over each horizon subsequently. As a result, the policies obtained are optimal only for each 10-second period, respectively and they are not optimal for the whole control process. In addition, the coordination of traffic signal between intersections is not considered. Clearly, we cannot always treat any control problem as a short process, such as in the case of EVSP (emergency vehicle signal preemption), especially in a coordinated network.

As Sen and Head (1997) themselves agreed that COP may not be an appropriate control method if it is difficult to provide accurate predictions of vehicle arrivals. On the other hand, the M-step model used in this thesis research is more attractive in dealing with this problem. For each M-step model, only those control parameters for the first time step are implemented. During the calculation for the next M-step model, control parameters for the next time step can be adjusted according to updated information on vehicle arrivals.

Finally, it should be noted that more than 10 years have passed since the DP-based control algorithm COP was proposed (Sen, 1993). It seems COP will not be put in practical operation in the near future. Maybe it is time to try other methods for real-time control of traffic signals.

2.5 Why is Simulink not selected for this research?

The characteristic of dynamic systems is that their states and outputs change over time. Simulink is a software development environment used for modelling, simulating, and analyzing dynamic systems (The Mathworks Inc., 2004b). For modelling, Simulink provides a graphical user interface for building a model as block diagram. After a model is built, simulation can be performed from the Simulink menus or by entering commands in the MATLAB command window. Model analysis tools include linearization and trimming tools.

There are two steps to simulate a dynamic system with Simulink. First, a user creates a block diagram, using the Simulink model editor, that graphically depicts time-

dependent mathematical relationships among the system's input, states, and outputs. Then the user commands Simulink to simulate the system represented by the model from a specified start time to a specified stop time.

Both continuous state and discrete state can occur in a Simulink model. While a continuous state changes continuously, a discrete state is often an approximation of a continuous state where the state is updated using finite intervals. A constraint on the simulation time step size is set for both continuous and discrete states. Specifically, a time step must be chosen to ensure that all the sample times of the model's states are hit. This task is accomplished by a component of the Simulink system called a discrete solver. There are two discrete solvers in Simulink: a fixed-step discrete solver, and a variable-step discrete solver.

The fixed-step discrete solver determines a fixed step size that hits all the sample times of all the model's discrete states, regardless of whether a state actually changes value at the sample time hits. By contrast, the variable-step discrete solver varies the step size to ensure the sample time occurs at times when the state changes values.

When simulating discrete systems, Simulink provides two ways to determine the sample time of a discrete block: Sample Time block parameter, and Sample Time inheritance. Discrete blocks allow the user to specify their sample times via a Sample Time parameter. If a block has no Sample Time parameter, an implicit discrete sample time is determined automatically based on input sample times. The implicit discrete sample time is equal to the shortest input sample time of all the input sample times, or integer multiples of the shortest sample time. Otherwise, the implicit sample time is equal to the fundamental sample time of the input sample times, where the fundamental

sample time of a set of sample times is defined as the greatest integer divisor of the set of sample times

A product relevant to Simulink is Real-Time Workshop. Real-Time Workshop is used to automatically generate C code from Simulink models and create real-time software applications on a variety of hardware systems.

From the above introduction, we can see that Simulink is a software development environment for microscopic models. If such a model for emergency vehicle signal preemption (EVSP) is developed, it is able to track space-time trajectories for every vehicle from its origin to its destination (Florian et al., 2001; Festa et al., 2001).

A microscopic model needs to describe every single driver's behaviour during EVSP, i.e., the process when a driver pulls over and stops at roadside after he/she notices the approach of an emergency vehicle, and then accelerates and resumes normal operation after the emergency vehicle has overtaken his/her stopped vehicle. Unfortunately, the mathematical relationship for the process does not exist at present, and it will be another research topic for emergency vehicle operation. Therefore, a simplification for the manoeuvre process of general traffic during EVSP is introduced in this research (see Section 3.2, Chapter 3). Instead of describing the detailed behaviour of every single driver, vehicles at an intersection are divided into three groups, i.e., standing queue, moving queue, and moving vehicles, and then their corresponding manoeuvres are determined as passing through the intersection or pulling over and stopping at road side in order to clear the way for the emergency vehicle.

While this simplification avoids the necessity to calibrate a high number of parameters for a microscopic model, it doesn't affect the comparative analysis results

for identification of potential and limits of different signal control strategies because the simplification has the same impact for different strategies.

As the result, Simulink as a microscopic simulation tool is not suitable for the proposed research and MATLAB is selected as the software development environment because it provides the maximum flexibility to develop simulation software for virtually any situation (The Mathworks Inc., 2003).

Chapter 3 Real-Time Control for Transition 1 - from Normal Operation to EVSP

It is generally known that motorists do not always have the opportunity to give the right-of-way (ROW) to EVs or they are not always aware of the approaching of an EV in some circumstances, such as in a traffic queue at a signalized intersection or when they are going to cross an intersection with green light facing them. Therefore, the traffic signal control system is required to clear the queue at the intersection before the EV crosses the intersection, in order to reduce the risk of collision of EVs with other vehicles, and to reduce the stress experienced by EV crews. Obviously, the benefit for EVs implies a penalty for general traffic because normal control is interrupted and the delay for general traffic is expected to increase to some extent especially in a coordinated traffic signal control system.

Before discussing EVSP control strategies in detail, the following section briefly describes typical intersections and traffic signal timing phases (Banks, 2002; Garber and Hoel, 2002)

3.1 Typical intersections and traffic signal timing phases

One objective of traffic design and traffic control is to resolve traffic conflicts, including merging, diverging, weaving, and crossing. Crossing conflicts at an intersection can be solved through three ways: time-sharing, space-sharing, and grade separation (Banks, 2002). Time-sharing solutions assign the right of way (ROW) to particular movement for particular time, such as in a signalized intersection at grade. Space-sharing solutions convert crossing conflicts into weaving conflicts, such as in a traffic circle or rotary. Grade separation solutions provide for the cross flow of traffic at different levels without interruption, such as in a freeway interchange. In this research, only time-sharing solutions at the signalized intersection at grade will be discussed for EVSP operation.

Typical intersections at grade include 3-leg intersection, 4-leg intersection, and multileg intersection. The main objectives of signal timing at an intersection are to reduce the average delay of all vehicles and the possibilities of accidents by assigning the ROW to different traffic streams at different times.

At an intersection, a driver has to select one of at most three alternative choices to proceed: left turn, through movement, or right turn. Accordingly, traffic signal control should provide each of the movements with an opportunity to clear the intersection. Usually there is no separate indication for right turns, and right turns have the ROW

during the green time for through movements on the same approach. Therefore, there are at most two separate green indications for each approach to an intersection: green for left turn, and green for through movement.

Based on the geometric layout of an intersection and associated traffic conditions, a left turn can be a protected movement or a permitted movement. A permitted left turn will conflict with its opposing through movement, while a protected left turn is not considered to conflict with its opposing through movement because it receives the green at the same time.

In the design of traffic signal timing, the following terms are commonly used (Garber and Hoel, 2002).

Cycle (cycle length) is the time required for one complete sequence of signal indications.

Interval is any part of the cycle length during which signal indications do not change.

Phase (signal phase) is any part of the cycle length during which a stream of traffic or a combination of two or more streams of traffic has the ROW simultaneously.

Clearance interval (yellow interval) is the length of time of yellow signal indication. This time is provided for vehicles to clear the intersection after the green interval.

All red interval is the length of time of red signal indication for all approaches to an intersection. It is sometimes used as a phase exclusively for pedestrians crossing or to allow vehicles and pedestrians to clear very wide intersections before conflicting movements are given the green indication.

Offset is the time lapse between the beginning of a green phase at an intersection and the beginning of the corresponding green phase at the next intersection.

Figure 3.1 shows the standard phase numbering scheme devised by the National Electrical Manufacturers Association (NEMA) of USA (Banks, 2002). In this figure, right turns are normally not numbered separately. Figure 3.2 shows phase diagrams for simple two-, three-, and four-phase signals (Banks, 2002).

It should be noted that the yellow interval and the all red interval are necessary between a green interval for a phase and the green interval for the next phase. They are determined by considering the geometric layout of the intersection, vehicular speed and pedestrian speed at the intersection. During an EVSP occurrence, the durations of the yellow interval and the all red interval usually must be kept for safety, and the durations of green intervals can be adjusted to serve the call from the EV.

Fig. 3.1 NEMA phase-numbering scheme

Fig. 3.2 Phase diagrams for traffic signals

3.2 Assumptions

The transition 1 period begins from the moment an EV is detected and ends at the time the EV clears the intersection with a safety time interval to keep it safe from the leading traffic.

When general traffic is moving at operating speed, the traffic density is low, and therefore there exists ample space for vehicles to pull over and stop at roadside. On the other hand, vehicles in a queue at an intersection usually do not have such an opportunity to pull over, since the space available there is very limited.

Based on the above observation, we divide vehicles into three groups according to

their states at an intersection when an approaching EV is detected (see Fig. 3.3):

- (1) vehicles in a queue at an intersection (standing queue) must pass through the intersection to clear the way for EV;
- (2) vehicles moving at operating speed (moving vehicle) must pull over and stop at roadside; and
- (3) vehicles moving at a speed less than operating speed (moving queue) should have both the opportunities to pull over and pass through the intersection.

The problem is that we do not know how far away from the end of standing queue vehicles begin to reduce speed and join the moving queue. While the relationship between the position of separating point (i.e., the location that separates moving vehicles and vehicles in a moving queue) and the operating speed can be established by experiment, we simplify the calculation in this research by the following process.

- (1) The queue comprises all the vehicles which have passed the EV-detection point and have not crossed the intersection. This simplification overestimates the queue by including the vehicles moving at operating speed in addition to vehicles in standing and moving queues.
- (2) The length of the road covered by the queue is estimated using the total number of vehicles and an average spacing between successive vehicles.

Fig. 3.3 Classification of vehicles at an intersection

In addition, we also assume that

- There is no mid-block friction (e.g., vehicles exiting from a parking lot/garage) since vehicles intending to enter the main road will keep off the main road after the audible message of EV's approaching is given. If the mid-block friction must be taken into consideration, vehicle detectors should be placed at the entrance to the EV road in order to provide information on the mid-friction.
- The queue length is less than the maximum value permitted under the normal signal control. If this assumption is not met, it means that a spillback or gridlock traffic condition has happened. In such a situation, a control strategy at the network level should be put in place along with the real-time control strategy at the intersection level.

3.3 Determination of EV detection distance

The distance of EV detection at an intersection should be determined so that after the EV is detected the control system will have enough time to discharge the queue at the intersection before the EV reaches the intersection. In order to provide such a notification time period, a minimum distance of EV detection can be determined using the following procedure.

- 1 Obtain traffic operating parameters at the intersection, such as EV speed, general traffic mean speed, discharge speed, startup delay, safety time interval, normal signal timing plans.
- 2 Establish the equation to calculate the notification time period, i.e., the time for EV to traverse the distance of EV detection.
- 3 Obtain the average queue length on the approach from which the EV will enter the EVSP system. In practice, we cannot locate the EV detection point according to the real-time value of queue length since it is a random variable. In order to ensure that in almost all traffic conditions the EV can pass through the intersection, we will use (average queue length + 2 times standard deviation) under peak hour traffic conditions to determine the required minimum distance of EV detection.
- 4 Calculate the discharge time, i.e., the time required to discharge the queue of (average

queue length + 2 times standard deviation) under peak hour traffic conditions.

- 5 Find out the switchover time, i.e., the time required to switch green indication from other approaches to the EV approach. Since an EV may approach an intersection at any instant in the signal cycle, the switchover time may be different for different EVSP occurrences. It is needed to consider all the possibilities of EVSP occurrences in the entire signal cycle, so that the longest switchover time will be used to determine the minimum distance of EV detection.
- 6 The queue of (average queue length + 2 times standard deviation) under peak hour traffic conditions should be discharged before the EV reaches the intersection, and a safety time interval (STI) must be kept between the last vehicle in the queue and the EV, i.e.,

$$\text{notification time period} \geq \text{switchover time} + \text{discharge time} + \text{STI}$$

From the above equation, the minimum distance of EV detection can be deduced.

In the following sections, the procedure to determine the minimum distance of EV detection is demonstrated using a four-leg intersection with two-phase signal system.

The phase sequence of the two-phase signal system is as follows.

<u>Main road:</u>	green	yellow	all red-1	red		all red-2
<u>Side road:</u>	red		all red-1	green	yellow	all red-2

Let L be the distance from the EV detection point on the approach to an intersection to the point the EV clears the intersection; S_{EV} the EV operating speed; and t_{EV} the time for EV to traverse the distance L . We have

$$t_{EV} = L / S_{EV} \quad (3.1)$$

Let STI be the safety time interval; S_{mean} the mean speed of traffic (i.e., general vehicles) on the approach before joining the queue; S_{dis} the discharge speed of the approach to the intersection, i.e., the speed that reflects the effect of queuing if any (generally S_{mean} is greater than S_{dis}); L_{QS} the average standing queue length plus 2 times standard deviation under peak hour traffic conditions; and L_{QM} the average equivalent queue length formed by vehicles moving on the approach (including the moving queue) plus 2 times standard deviation under peak hour traffic conditions. L_{QS} and L_{QM} can be calculated by counting the numbers of incoming vehicles and outgoing vehicles, or using other methods, such as described by Chang (2000).

Based on the instant in the signal cycle when the signal controller receives the message of the EV's approaching, the following scenarios exist.

- (1) Signal indication facing the EV is in green when the controller is informed that the EV is approaching.

Let t_{green} be the time elapsed from the beginning of green of current cycle, and t_{clear} ,

the duration for all the vehicles ahead of EV to clear the intersection. While t_{clear} should be calculated based on the current position of the last vehicle and the distribution of vehicles on the approach, we simplify the calculation by estimating two extreme situations: (a) the last vehicle traverses $(L_{QS} + L_{QM})$ and passes through the intersection without reducing speed; and (b) the last vehicle traverses $(L_{QS} + L_{QM})$ and passes through the intersection after joining the standing queue.

For the first situation we have

$$t = (L_{QS} + L_{QM}) / S_{\text{mean}} \quad (3.2)$$

and for the second situation we have

$$t' = (L_{QS} + L_{QM}) / S_{\text{dis}} \quad (3.3)$$

Usually $t < t'$ and therefore we use t' to determine the distance of EV detection based on the statistical value of $(L_{QS} + L_{QM})$ under peak hour traffic conditions. However, in real-time mode, it is possible that the actual queue length is less than the statistical value used for the determination of EV detection distance. This point will be discussed in detail in following sections.

In this section, we use t' to be equal to t_{clear} as follows

$$t_{\text{clear}} = (L_{QS} + L_{QM}) / S_{\text{dis}} \quad (3.4)$$

Consequently we have the following equation to determine the nearest distance L_{min} for EV to notify the controller that it is approaching.

$$t_{\text{EV}} - t_{\text{clear}} \geq \text{STI} \quad (3.5)$$

From equations (3.1), (3.4) and (3.5) we have

$$L_{\min} \geq (STI + (L_{QS} + L_{QM}) / S_{dis}) * S_{EV} \quad (3.6)$$

(2) Signal indication facing the EV is in yellow when the controller is informed that the EV is approaching.

In this scenario the indication for cross approaches must be in red, and no vehicle on the cross road will be in the intersection. Therefore, while signal for the cross road continues red, signal for the EV road can turn green immediately.

Thus we obtain similar equations to those for scenario (1) to determine the nearest distance $L_{\min}$ for EV to notify the controller of its approaching.

$$t_{clear} = t_{startup} + (L_{QS} + L_{QM}) / S_{dis} \quad (3.7)$$

where $t_{startup}$ = startup delay.

$$t_{EV} - t_{clear} \geq STI \quad (3.8)$$

$$L_{\min} \geq (STI + t_{startup} + (L_{QS} + L_{QM}) / S_{dis}) * S_{EV} \quad (3.9)$$

(3) Signal indication facing the EV is in red when the controller is informed that the EV is approaching.

Let t_{red} be the time elapsed from the beginning of red of current cycle; and T_{red-1} ,

T_{red-2} , the first and the second all red periods of time, respectively. According to the state of signal indicators facing the cross road we have the following sub-scenarios.

(3.1) Signal indication for cross road is in red

If $t_{red} \leq T_{red-1}$ then the signal for the cross road is at the end of its red, and no vehicle on the cross road will be in the intersection. Therefore, while signal for the cross road continues red, signal for the EV road can turn green immediately. The equation for scenario (2) is applicable to this scenario.

If $t_{red} > (T_{red-1} + red)$, i.e., t_{red} is in the period of T_{red-2} , then the signal for the cross road is at the beginning of its red; and probably some vehicles on the cross road are still in the intersection.

Let t'_{red} be the time elapsed from the beginning of red of current cycle for the cross road. The signal indications must go through the following procedure to clear the queue ahead of the EV

- Signal for the cross road continues red for $(T_{red-2} - t'_{red})$
- Signal for the EV road turns green to clear the queue:

$$t_{clear} = t_{startup} + (L_{QS} + L_{QM}) / S_{dis} \quad (3.7)$$

Thus we obtain the equation to determine the nearest distance L_{min} for EV to notify the controller of its approaching.

$$t_{EV} - (T_{red-2} - t'_{red} + t_{clear}) \geq STI \quad (3.10)$$

Using equations (3.1), (3.7) and (3.10) we get

$$L_{\min} \geq (STI + t_{\text{startup}} + (L_{QS} + L_{QM}) / S_{\text{dis}} + T_{\text{red-2}} - t'_{\text{red}}) * S_{EV} \quad (3.11)$$

(3.2) Signal indication for cross road is in green.

The signal indications must go through the following procedure to clear the queue ahead of the EV

- Signal for the cross road turns yellow for T'_{yellow}
- Signals turn red for all approaches for $T_{\text{red-2}}$
- Signal for the EV road turns green to clear the queue:

$$t_{\text{clear}} = t_{\text{startup}} + (L_{QS} + L_{QM}) / S_{\text{dis}} \quad (3.7)$$

Thus we obtain the equation to determine the nearest distance $L_{\min}$ for EV to notify the controller of its approaching.

$$t_{EV} - (T'_{\text{yellow}} + T_{\text{red-2}} + t_{\text{clear}}) \geq STI \quad (3.12)$$

From equations (3.1), (3.7) and (3.12) we have

$$L_{\min} \geq (STI + t_{\text{startup}} + (L_{QS} + L_{QM}) / S_{\text{dis}} + T'_{\text{yellow}} + T_{\text{red-2}}) * S_{EV} \quad (3.13)$$

(3.3) Signal indication for cross road is in yellow.

There is the possibility that some vehicles on the cross road have entered the intersection. Let t'_{yellow} be the time elapsed from the beginning of yellow of current cycle for the cross road. The signal indications must go through the following procedure to

clear the queue ahead of the EV

- Signal for the cross road continues yellow for $(T_{yellow} - t_{yellow}')$
- Signals turn red for all approaches for T_{red-2}
- Signal for the EV road turns green to clear the queue:

$$t_{clear} = t_{startup} + (L_{QS} + L_{QM}) / S_{dis} \quad (3.7)$$

Thus we obtain the equation to determine the nearest distance L_{min} for EV to notify the controller of its approaching.

$$t_{EV} - (T_{yellow} - t_{yellow}' + T_{red-2} + t_{clear}) \geq STI \quad (3.14)$$

From equations (3.1), (3.7) and (3.14) we have

$$L_{min} \geq (STI + t_{startup} + (L_{QS} + L_{QM}) / S_{dis} + T_{yellow} - t_{yellow}' + T_{red-2}) * S_{EV} \quad (3.15)$$

3.4 Real-time control of traffic signals for EVSP

After the distance of EV detection is determined, however, real-time control of signal system must be based on current queue length when an EVSP occurs, in order to minimize the impact of the EVSP on general traffic. This can be achieved by allotting just enough green time for the EV to pass through the intersection, and allocating possible spare green time to general traffic on the cross road.

The real-time control algorithm for EVSP at an intersection can be accomplished using the following procedure.

- 1 The notification time period is fixed and is determined using the method described in the previous section.
- 2 Obtain current queue length at the intersection when an approaching EV is detected.
- 3 Calculate discharge time, i.e., the time required to discharge current queue at the intersection.
- 4 Find out switchover times, i.e., the time required to switch green indication from the EV approach to other approaches (if other approaches are not facing green when the EV is detected) and the time required to switch green indication back to the EV approach.
- 5 A safety time interval (STI) must be kept between the last vehicle in the queue and the EV.
- 6 Calculate the possible green time for the cross road, using the following equation.

Possible green time for the cross road =

$$\text{notification time period} - (\text{switchover times} + \text{discharge time} + \text{STI})$$

This general procedure for the real-time control of EVSP is demonstrated using a four-leg intersection with two-phase signal system in the following sections.

For each scenario discussed in Section 3.3, after the signal controller has been informed about the approaching EV, we need to find out whether there is the possibility to

change or extend green for the cross road traffic and then to meet the EV requirement to clear the intersection without delay.

(1) Signal indication facing the EV is in green (at t_{green}) when the controller is informed that the EV is approaching.

To serve the cross traffic and then to meet the EV requirement, the signal indications must go through the following procedure.

- Signal for the EV road turns yellow for T_{yellow} , and then turns red (for $t_{\text{red}} \geq T_{\text{red-1}}$)
- Signal for the cross road turns green at $t_{\text{red}} = T_{\text{red-1}}$ for t'_{green}
- Signal for the cross road turns yellow for T'_{yellow} , and then turns red (for $t'_{\text{red}} \geq T_{\text{red-2}}$)
- Signal for the EV road turns green at $t'_{\text{red}} = T_{\text{red-2}}$ to clear the queue, and then serves the EV.

Thus we obtain the equation to determine the available green time for the cross road as follows.

$$t'_{\text{green}} = t_{\text{EV}} - (T_{\text{yellow}} + T_{\text{red-1}} + T'_{\text{yellow}} + T_{\text{red-2}} + t_{\text{clear}} + \text{STI}) \quad (3.16)$$

It should be noted that t_{EV} in above equation is predetermined according to statistical queue length value under peak hour traffic conditions, while t_{clear} is based on current value. Denoting the current value of queue length as $L_{Q \text{ real}}$, we have

$$t = L_{Q_real} / S_{mean} \quad (3.17)$$

$$t' = L_{Q_real} / S_{dis} \quad (3.18)$$

and

$$t_{clear} = \max(t, t') \quad (3.19)$$

If there exists the possibility that the discharge at an intersection causes queue length greater than the maximum queue length permitted on the approach to downstream intersection, $(L_{QS} + L_{QM})_{max}$, the t'_{green} should be cut shorter to avoid such a situation.

(2) Signal indication facing the EV is in yellow (at t_{yellow}) when the controller is informed that the EV is approaching.

To serve the crossing traffic and then to meet the EV requirement, the signal indications must go through the following procedure.

- Signal for the EV road continues yellow for $(T_{yellow} - t_{yellow})$, and then turns red (for $t_{red} \geq T_{red-1}$)
- Signal for the cross road turns green at $t_{red} = T_{red-1}$ for t'_{green}
- Signal for the cross road turns yellow for T'_{yellow} , and then turns red (for $t'_{red} \geq T_{red-2}$)
- Signal for the EV road turns green at $t'_{red} = T_{red-2}$ to clear the queue, and then serves the EV.

$$t_{clear} = t_{startup} + \max(t, t') \quad (3.19a)$$

Thus we obtain the equation to determine the available green time for the cross road as follows.

$$t'_{green} = t_{EV} - (T_{yellow} - t_{yellow} + T_{red-1} + T'_{yellow} + T_{red-2} + t_{clear} + STI) \quad (3.20)$$

(3) Signal indication facing the EV is in red (at t_{red}) when the controller is informed that the EV is approaching.

(3.1) Signal indication for cross road is in red.

If $t_{red} \leq T_{red-1}$ then the signal for the EV road is at the beginning of its red, and probably some vehicles on the EV road are still in the intersection. To serve the cross traffic and then to meet the EV requirement, the signal indications must go through the following procedure.

- Signal for the EV road continues red for $(T_{red-1} - t_{red})$
- Signal for the cross road turns green for t'_{green}
- Signal for the cross road turns yellow for T'_{yellow} , and then turns red (for $t'_{red} \geq T_{red-2}$)
- Signal for the EV road turns green at $t'_{red} = T_{red-2}$ to clear the queue, and then serves the EV.

$$t_{clear} = t_{startup} + \max(t, t') \quad (3.19a)$$

Thus we obtain the equation to determine the available green time for the cross road

as follows.

$$t'_{\text{green}} = t_{\text{EV}} - (T_{\text{red-1}} - t_{\text{red}} + T'_{\text{yellow}} + T_{\text{red-2}} + t_{\text{clear}} + \text{STI}) \quad (3.21)$$

If $t_{\text{red}} > T_{\text{red-1}}$ then the signal for the EV road is at the end of its red; and no vehicle on the EV road will be in the intersection. Therefore the signal for the cross road can turn green immediately while signal for EV road remains red.

To serve the cross road traffic and then to meet the EV requirement, the signal indications must go through the following procedure.

- Signal for the cross road turns green for t'_{green}
- Signal for the cross road turns yellow for T'_{yellow} , and then turns red (for $t'_{\text{red}} \geq T_{\text{red-2}}$)
- Signal for the EV road turns green at $t'_{\text{red}} = T_{\text{red-2}}$ to clear the queue, and then serves the EV.

$$t_{\text{clear}} = t_{\text{startup}} + \max(t, t') \quad (3.19a)$$

Thus we obtain the equation to determine the available green time for the cross road as follows.

$$t'_{\text{green}} = t_{\text{EV}} - (T'_{\text{yellow}} + T_{\text{red-2}} + t_{\text{clear}} + \text{STI}) \quad (3.22)$$

(3.2) Signal indication for cross road is in green.

To serve the cross road traffic and then to meet the EV requirement, the signal indications must go through the following procedure.

- Signal for the cross road continues green for t'_{green}
- Signal for the cross road turns yellow for T'_{yellow} , and then turns red (for $t'_{\text{red}} \geq T_{\text{red-2}}$)
- Signal for the EV road turns green at $t'_{\text{red}} = T_{\text{red-2}}$ to clear the queue, and then serves the EV.

$$t_{\text{clear}} = t_{\text{startup}} + \max(t, t') \quad (3.19a)$$

Thus we obtain the equation to determine the available green time for the cross road as follows.

$$t'_{\text{green}} = t_{\text{EV}} - (T'_{\text{yellow}} + T_{\text{red-2}} + t_{\text{clear}} + \text{STI}) \quad (3.23)$$

(3.3) Signal indication for cross road is in yellow.

In this scenario, the signal for the EV road is near the end of its red; and no vehicle on main road will be in the intersection. Therefore, the signal for the cross road can be turned green immediately while signal for the EV road remains red.

To serve the cross road traffic and then to meet the EV requirement, the signal indications must go through the following procedure.

- Signal for the cross road turns green for t'_{green}
- Signal for the cross road turns yellow for T'_{yellow} , and then turns red (for $t'_{\text{red}} \geq T_{\text{red-2}}$)
- Signal for the EV road turns green at $t'_{\text{red}} = T_{\text{red-2}}$ to clear the queue, and then serves the EV.

$$t_{\text{clear}} = t_{\text{startup}} + \max(t, t') \quad (3.19a)$$

Thus we obtain the equation to determine the available green time for the cross road as follows.

$$t_{\text{green}} = t_{\text{EV}} - (T_{\text{yellow}} + T_{\text{red-2}} + t_{\text{clear}} + \text{STI}) \quad (3.24)$$

3.5 Real-time control for situations when minimum distance requirement of EV detection cannot be met

The real-time control algorithm presented in previous sections assumes that an EV can be detected at the minimum distance of EV detection from an intersection. However, in practice, the EVSP does not always occur in such a perfect setting. A fire station or a hospital is probably located near an intersection and its distance to the intersection is shorter than the required minimum distance of EV detection. Another scenario is that an EV, such as a police car, may start suddenly from a mid-block for an emergency and the distance to the nearby intersection is shorter than the required minimum distance of EV detection.

For the first scenario, i.e., the fire station or hospital location problem, one solution is using the conventional EVSP system. The EVSP system is activated by pushing a button in the fire station or hospital and the green indication at the nearby intersection is switched to the EV approach for a specified period of time or until the switch is turned off (Federal Highway Administration, 1996). This conventional solution will not be

discussed further in this research.

Another solution is to control the EVSP system in a real-time manner, and it is the same for the second scenario, in which an EV starts suddenly from a mid-block. An optical-based, sound-based, or GPS-based EVSP system should be used to detect the EV and activate the EVSP system. Once the EVSP system receives the call from the EV, it starts the process to change the traffic signal timing plan from normal operation to EVSP operation. The notification period of time for preemption (t_{EV}) is equal to the quotient of the distance between the EV and the intersection divided by the EV operating speed plus a startup delay.

The objective of the EVSP system is to discharge the queue between the EV and the intersection and clear the way for the EV. Another task for the EVSP system is to find out if it is possible to allocate the remaining portion of the notification period of time as green time for other approaches in order to minimize the impact of the EVSP operation on general traffic.

The real-time control algorithm presented in previous sections is still applicable to the situations discussed in this section, i.e., the minimum distance requirement of EV detection cannot be met, except that the notification period of time for preemption probably is not long enough to discharge all the queue between the EV and the intersection and consequently the EV has to stop or reduce speed before it crosses the intersection in many individual EVSP occurrences. In such a situation, other methods at

network level have to be developed in order to provide an adequate notification time for the real-time control strategy at intersection level to clear the way for the EV.

3.6 Real-time control strategy for coordinated route

A difference between the coordinated route control and the single intersection control is that the vehicle arrivals at a coordinated intersection are determined by the control strategy at its upstream intersection. Another difference is that a link length between two successive intersections in a coordinated route is probably shorter than required distance for EV detection. If such a case happens, the EV detection point for an intersection will be beyond its upstream intersection.

The method to determine the minimum distance of EV detection is the same as for single intersection, which is described in Sections 3.3, and only the real-time control process is different.

3.6.1 General procedure to develop real-time control algorithm for coordinated route

It is assumed that the minimum distance requirement of EV detection is met at the first intersection and a link length between two successive intersections is probably

shorter than the required minimum distance. Therefore, they are discussed separately in the following sections.

(1) EV at the first intersection of the route

- (a) Calculate the notification period of time, t_{EV} , which is the quotient of EV detection distance divided by the EV operating speed.
- (b) Obtain queue lengths on individual approaches to the first intersection of the route at the moment the EV is detected.
- (c) Calculate the minimum green time, t_{min} , which is required to discharge the queue ahead of the EV and clear the way for the EV to cross the first intersection.
- (d) Determine the switchover time, $t_{switchover}$, which is needed to switch green indication from the EV approach to other approaches (if other approaches are not facing green when the EV is detected) and then back to the EV approach.
- (e) Calculate the maximum green time for other approaches to the first intersection, t'_{max}

$$= t_{EV} - t_{min} - t_{switchover}$$
- (f) Calculate vehicular arrivals into the next link at the moment the EV just crosses the first intersection.
- (g) Calculate the queue length, Δq , which is discharged at the next intersection during the period from the instant the EV is detected to the moment the EV has just crossed the first intersection.
- (h) The duration for the EV to traverse the link length to the next intersection should be

long enough to clear the queue at the next intersection before the EV reaches the intersection. Otherwise, cut short the maximum green time for other approaches, t'_{max} , to prevent vehicles on other approaches from entering the EV route. If the queue at the next intersection is still too long to be cleared, increase green time up to t_{EV} (minus any possible switchover time) for the EV approach to the next intersection.

(2) EV at another intersection of the route

If the link length to the subject intersection is longer than the required minimum distance of EV detection, the real-time control process is the same as for the first intersection. If the link length to the subject intersection is shorter than the required minimum distance of EV detection, the EV detection point will be beyond the upstream intersection, and the real-time control process is as follows.

- (a) Calculate the notification period of time, t_{EV} , which is the quotient of EV detection distance divided by the EV operating speed.
- (b) Calculate the traverse time, t_{LINK} , which is the quotient of the link length to the subject intersection divided by the EV operating speed.
- (c) Obtain queue lengths on individual approaches to the subject intersection of the route at the moment the EV is detected.
- (d) Based on the signal timing plans for the upstream intersection, calculate queue length on individual approaches to the subject intersection at the moment the EV has just

crossed the upstream intersection.

- (e) Calculate the minimum green time, $t_{\min}$, which is required to discharge the queue at the subject intersection and clear the way for the EV.
- (f) Determine the switchover time, $t_{\text{switchover}}$, which is needed to switch green indication from the EV approach to other approaches (if other approaches are not facing green when the EV is detected) and then back to the EV approach.

- (g) Calculate the maximum green time for other approaches to the subject intersection,

$$\dot{t}_{\max} = t_{\text{LINK}} - t_{\min} - t_{\text{switchover}}$$

- (h) Calculate vehicular arrivals into the next link at the moment the EV just crosses the subject intersection.
- (i) Calculate the queue length, Δq , which is discharged at the next intersection during the period from the instant the EV is detected to the moment the EV has just crossed the subject intersection.
- (j) The duration for the EV to traverse the link length to the next intersection should be long enough to clear the queue at the next intersection before the EV reaches the intersection. Otherwise, cut short the maximum green time for other approaches, $\dot{t}_{\max}$, to prevent vehicles on other approaches from entering the EV route. If the queue at the next intersection is still too long to be cleared, increase green time up to t_{EV} (minus any possible switchover time) for the EV approach to the next intersection.

3.6.2 Detailed formulation and algorithm for coordinated route control

The above general procedures for coordinated route control are demonstrated using a route consisting of four-leg intersections with two-phase signal system in the following sections. The EV is assumed to reach an intersection from its eastbound (EB) approach.

(1) EV at the first intersection of the route

Denote queue lengths on approaches to the first intersection as $q_{1E}(t_1)$, $q_{1N}(t_1)$, and $q_{1S}(t_1)$ for eastbound, northbound, and southbound, respectively. t_1 stands for the starting instant of EVSP, at which an EV enters the system and it is detected at distance D away from the intersection. We need a minimum green time, $t_{\min}(1)$, to clear the queue ahead of the EV.

$$t_{\min}(1) = q_{1E}(t_1)/S_{g1E} \quad (3.25)$$

where S_{g1E} = discharge rate for eastbound approach to intersection 1.

Thus, the maximum green time available for the cross road, $t'_{\max}(1)$, can be calculated as:

$$t'_{\max}(1) = D/S_{EV} - t_{\min}(1) - t_{\text{switchover}} \quad (3.26)$$

where D = EV detection distance away from the intersection;

S_{EV} = EV operating speed; and

$t_{\text{switchover}}$ = period of time needed to switchover traffic signals.

Expected arrivals into the next link, L_{12} , after D/S_{EV} will be:

$$a_{2E}(t_1:t_1 + D/S_{EV}) = q_{1E}(t_1) \rho_{1TH} + \min[t'_{\max}(1) s_{g1N}, q_{1N}(t_1) + a_{1N}(t_1:t_1 + D/S_{EV})] \rho_{1RT} + \min[t'_{\max}(1) s_{g1S}, q_{1S}(t_1) + a_{1S}(t_1:t_1 + t'_{\max}(1))] \rho_{1LT} \quad (3.27)$$

where $a_{2E}(t_1:t_1 + D/S_{EV})$ = vehicle arrivals into eastbound approach to intersection 2 during the period of time t_1 to $t_1 + D/S_{EV}$;

$a_{1N}(t_1:t_1 + D/S_{EV})$, $a_{1S}(t_1:t_1 + t'_{\max}(1))$ = vehicle arrivals into northbound and southbound approaches to intersection 1 during the period of time t_1 to $t_1 + D/S_{EV}$ and t_1 to $t_1 + t'_{\max}(1)$, respectively;

ρ_{1TH} , ρ_{1RT} , ρ_{1LT} = through, right-turn, left-turn percentages at intersection 1, respectively.

S_{g1N} , S_{g1S} = discharge rates for northbound and southbound approaches to intersection 1, respectively.

Denote queue lengths on eastbound approach to intersection 2 (i.e., L_{12}) at t_1 as $q_{2E}(t_1)$, and green time for eastbound approach to intersection 2 during t_1 to $t_1 + D/S_{EV}$ under normal operation as $t(2)$. $t(2)$ will discharge the maximum queue:

$$\Delta q(2) = \min[t(2) S_{g2E}, q_{2E}(t_1) + a_{2E}(t_1:t_1 + D/S_{EV})] \quad (3.28)$$

The duration for the EV to traverse the link length, L_{12} , should be long enough to clear the queue on eastbound approach to intersection 2, $q_{2E}(t_1)$, considering the opportunity for vehicle arrivals $a_{2E}(t_1:t_1 + D/S_{EV})$ to pull over and stop at road side after

they just pass through an intersection. Another conservative strategy is to clear both $q_{2E}(t_1)$ and $a_{2E}(t_1:t_1 + D/S_{EV})$. It means if

$$q_{2E}(t_1) + a_{2E}(t_1:t_1 + D/S_{EV}) - \Delta q(2) > (L_{12} / S_{EV}) S_{g2E} \quad (3.29)$$

or

$$q_{2E}(t_1) + a_{2E}(t_1:t_1 + D/S_{EV}) - \Delta q(2) > q_{2E,max} \quad (3.30)$$

where $q_{2E,max}$ = maximum queue length permitted on eastbound approach to intersection 2;

the maximum green time available for the cross road, $t_{max}(1)$, must be cut short to prevent vehicles on the cross road from entering EV route, or green time for eastbound approach to intersection 2 must be increased.

(2) EV at intersection i other than the first intersection

If the link length, $L_{i-1,i}$, to intersection i is equal to or greater than required EV detection distance D, the control algorithm remains the same as for the first intersection.

If $L_{i-1,i} < D$, we need a minimum green time, $t_{min}(i)$, to clear the queue, $q_{iE}(t_1)$, ahead of the EV at the instant, $t(i)$, at which EV just crossed its upstream intersection.

$$t_{min}(i) = q_{iE}(t_1)/S_{gii} \quad (3.31)$$

where S_{gii} = discharge rate for eastbound approach to intersection i.

And then the maximum green time available for the cross road, $t_{max}(i)$, can be calculated as:

$$t'_{\max}(i) = L_{i-1,i} / S_{EV} - t_{\min}(i) - t_{\text{switchover}} \quad (3.32)$$

Expected arrivals into the next link, $L_{i,i+1}$, after $L_{i-1,i}/S_{EV}$ will be:

$$\begin{aligned} a_{i+1,E}(t_i:t_i + L_{i-1,i}/S_{EV}) &= q_{iE}(t_i) \rho_{iTH} + \\ &\min[t'_{\max}(i) S_{gN}, q_{iN}(t_i) + a_{iN}(t_i:t_i + L_{i-1,i}/S_{EV})] \rho_{iRT} + \\ &\min[t'_{\max}(i) S_{gS}, q_{iS}(t_i) + a_{iS}(t_i:t_i + t'_{\max}(i))] \rho_{iLT} \end{aligned} \quad (3.33)$$

where $a_{i+1,E}(t_i:t_i + L_{i-1,i}/S_{EV})$ = vehicle arrivals into eastbound approach to intersection $i+1$

during the period of time t_i to $t_i + L_{i-1,i}/S_{EV}$;

$a_{iN}(t_i:t_i + L_{i-1,i}/S_{EV})$, $a_{iS}(t_i:t_i + t'_{\max}(i))$ = vehicle arrivals into northbound and

southbound approaches to intersection i during the period of

time t_i to $t_i + L_{i-1,i}/S_{EV}$ and t_i to $t_i + t'_{\max}(i)$, respectively;

ρ_{iTH} , ρ_{iRT} , ρ_{iLT} = through, right-turn, left-turn percentages at intersection i ,

respectively.

S_{gN} , S_{gS} = discharge rates for northbound and southbound approaches to

intersection i , respectively.

Denote queue lengths on eastbound approach to intersection $i+1$ (i.e., $L_{i,i+1}$) at t_i as

$q_{(i+1)E}(t_i)$, and green time as $t(i+1)$ for eastbound approach to intersection $i+1$ during t_i to

$t_i + L_{i-1,i}/S_{EV}$ under normal operation. The $t(i+1)$ will discharge the maximal queue:

$$\Delta q(i+1) = \min[t(i+1) S_{g(i+1)E}, q_{(i+1)E}(t_i) + a_{(i+1)E}(t_i:t_i + L_{i-1,i}/S_{EV})] \quad (3.34)$$

The duration for EV to traverse the link length, $L_{i,i+1}$, should be long enough to clear

the queue on eastbound approach to intersection $i+1$, $q_{(i+1)E}(t_i)$, or to conservatively clear both $q_{(i+1)E}(t_i)$ and $a_{(i+1)E}(t_i; t_i + L_{i+1,i} / S_{EV})$. It means if

$$q_{(i+1)E}(t_i) + a_{(i+1)E}(t_i; t_i + L_{i+1,i} / S_{EV}) - \Delta q(i+1) > (L_{i+1,i} / S_{EV}) S_{g(i+1)E} \quad (3.35)$$

or

$$q_{(i+1)E}(t_i) + a_{(i+1)E}(t_i; t_i + L_{i+1,i} / S_{EV}) - \Delta q(i+1) > q_{(i+1)E,max} \quad (3.36)$$

where $q_{(i+1)E,max}$ = maximum queue length permitted on eastbound approach to intersection $i+1$;

the maximum green time available for the cross road, $t_{max}(i)$, must be cut short to prevent vehicles on the cross road from entering EV route, or green time for eastbound approach to intersection $i+1$ must be increased.

Chapter 4 Optimal Control for Transition 2 - from EVSP back to Normal Operation

4.1 Assumptions and control strategy for signal transitioning

Transition 2 begins from the instant the detector decides that the EV has cleared the intersection, and ends at the moment the controller begins the normal operation. Our assumptions for this transition 2 are as follows.

- After the EV passes through the intersection, there must be a safety space or safety time interval (STI) between the EV and the vehicle following it.
- There is no gridlock so that vehicles discharged at the intersection can enter the link to the next intersection. If a gridlock does happen, it cannot be solved by the optimal control strategy at the intersection level alone, and the optimal control strategy has to be combined with other strategies at the network level to deal this situation.

The duration of transition should be as short as possible, and at the end of the transition the queue length on each approach to the intersection should be equal to or less than the average queue length under normal operating condition. Therefore, normal signal control strategy can be put in place as soon as possible while the impact of EV on

general traffic is minimized.

4.2 General procedure to develop optimal control algorithm

The general procedure to develop the optimal control algorithm for the transitioning is described as follows.

- (1) Find out the time duration (remaining normal cycle time, RNC) from the instant an EV just crosses an intersection to the beginning of immediate next normal green signal phase.
- (2) Determine the transition period (recovery period, RP) for the optimal control process. RP is initially selected as (RNC-STI), and it is probably extended by additional signal cycle lengths for normal operation if the initial RP is too short or at the end of RP queue lengths on individual approaches are longer than corresponding average values under normal operation.
- (3) Discretize the recovery period RP into N time steps (N time step equations), and each time step consists of green times, yellow intervals, and all red intervals for individual approaches. The time duration for each time step is fixed, and green times for individual approaches are variables and are to be determined by the optimal control algorithm. A switchover during a time step will introduce an entire yellow interval and corresponding all red interval into this time step because of safety consideration.

If there is no switchover during a time step, there will be no yellow interval and all red interval. The optimal control algorithm will determine whether there is any switchover or not during a time step.

- (4) For each time step, establish the dynamic relationship (functions of time) among queue lengths, vehicular arrivals, green times, and yellow intervals.
- (5) Formulate restrictions on each green time, minimum green time, maximum green time, and queue length.
- (6) Define a performance index (PI) in terms of green times, queue lengths for each of all the time steps, and required final queue lengths at the end of the optimal control process. In the PI, different weighting matrices can be used to weight terminal state (final queue lengths), intermediate states (queue lengths at the end of individual intermediate time steps), and control variables (green times for individual time steps) differently.
- (7) Construct an augmented Hamiltonian function using the PI, functions of time, time step equations, and all the constraints on green times and queue lengths.
- (8) A dynamic system is established by deducing the first order necessary conditions for the augmented Hamiltonian function, and the solution to the dynamic system will be the optimal control strategy for the recovery process from EVSP to normal signal operation.
- (9) The dynamic system cannot be solved directly because of the interrelationship

between signal sequences of greens, yellows and all reds in successive time intervals. A two-phase algorithm is proposed to solve the dynamic system. The first phase is a relaxation process in which inequality constraints and the relationship between successive signal sequences are neglected, and a low bound of the solution to the original optimal control problem is deduced. Inequality constraints and the relationship between successive signal sequences are considered in the second phase of the two-phase algorithm, and a practical solution is obtained.

- (10) In order to reduce computation complexity and control the signal timing transition in a real-time mode, only M time steps out of the total N time steps are considered at a time. After one M -step model is solved, only those optimal control parameters of the first time step within the M steps are implemented. The next M -step model will begin with the second time step in the last M -step model. This mathematical problem solving technique is especially useful in situations where a large number of total time steps is needed for the recovery process from EVSP to normal operation.

To demonstrate the procedure for the optimal control algorithm, detailed formulations are deduced for the typical four-leg intersection with two-phase signal system in the following sections.

The intersection layout is shown in Fig. 4.1. In Fig. 4.1, the number of lanes on each approach can be one, two, or more than two. However, no matter how many lanes on an

approach, all the lanes on the approach are in one lane group, and therefore only one signal phase is needed for the approach.

The traffic signal phase sequence of two-phase signal system is illustrated as follows.

<u>Main road:</u>	green	yellow	all red-1	red		all red-2
<u>Side road:</u>	red		all red-1	green	yellow	all red-2

Fig. 4.1 Layout of intersections

4.3 Formulation of the optimal control model for signal transitioning

Let C denote the normal cycle length; t_1 the time period from the beginning of immediately past normal green signal to the instant EV is detected; t_2 the time period from the instant EV is detected to the instant EV just clears the intersection (see Fig. 4.2). Then the remaining normal cycle time (RNC), which is needed to get the signal control back to coordinated normal cycle, will be $RNC = C - (t_1 + t_2) \bmod C$, where mod is a function whose result is the remainder of $(t_1 + t_2)$ divided by C .

Fig. 4.2 Calculation of RNC

The time duration of transition (excluding STI), RP , can be determined as

$$RP = [RNC + nC - STI] \quad n = 0, 1, 2, \dots \quad (4.1)$$

such that the normal operation can begin immediately at the end of the transition. n stands for the number of additional normal cycles required for the transition so that at the end of the transition queue lengths on individual approaches are shorter than or equal to average

queue length under normal operation. The value of n begins from zero and it will increase by one subsequently until the queue length requirement at the end of transition is met. In order to achieve the EVSP control strategy, we evenly divide the transition period into N time intervals (or time steps)

$$T = [RNC + n C - STI] / N \quad (4.2)$$

Therefore, we have the following relationship

$$\begin{aligned} NT &= [RNC + n C - STI] \\ &= \sum_{i=1}^N [g_1(i) + Y1 + AR1 + r_1(i) + AR2] \\ &= \sum_{i=1}^N [r_2(i) + AR1 + g_2(i) + Y2 + AR2] \end{aligned} \quad (4.3)$$

where

$g_1(i)$, $g_2(i)$ = green time for main road and side road during i^{th} time step, respectively;

$r_1(i)$, $r_2(i)$ = red time for main road and side road during i^{th} time step, respectively;

$AR1$, $AR2$ = all red time-1 and all red time-2 during i^{th} time step, respectively; and

$Y1$, $Y2$ = yellow time intervals for main road and side road during i^{th} time step, respectively.

From the signal phase sequence, we have

$$r_2(i) = g_1(i) + Y1 \quad (4.4)$$

and

$$r_1(i) = g_2(i) + Y2 \quad (4.5)$$

Then, we can write the transition duration as

$$\begin{aligned} N T &= [RNC + n C - STI] \\ &= \sum_{i=1}^N [g_1(i) + g_2(i) + Y1 + AR1 + Y2 + AR2] \end{aligned} \quad (4.6)$$

From the signal phase sequences, each time step T is a combination of green, yellow, and red intervals, depending on whether there is any switchover or not. If there is any switch over, we can select a T such that at most only one switchover can be made.

$$T \leq 2 \max(g_{1min}, g_{2min}) + Y1 + AR1 + Y2 + AR2 \quad (4.7)$$

and therefore the total number of time steps will be

$$N = [RNC + n C - STI] / T \quad (4.8)$$

But there also exists the possibility that no switchover happens during such a time step. Then the time step can be written as

$$T = g_1(i) + g_2(i) + \delta_1(i) [Y1 + AR1] + \delta_2(i) [Y2 + AR2] \quad (4.9)$$

where $\delta_j(i) = 0$ or 1 , depending upon signal sequences during last and current time steps $\delta(i)$ can be determined dynamically according to traffic condition at the intersection during an optimal control process, which will be presented in following sections.

During each time step, the queues are calculated using the following functions of time.

$$q_1(i+1) = q_1(i) + a_1(i) - S_{g1} g_1(i) - S_{y1} y_1(i) \delta_1(i)$$

$$\begin{aligned}
q_3(i+1) &= q_3(i) + a_3(i) - S_{g3} g_3(i) - S_{y3} y_3(i) \delta_1(i) \\
q_2(i+1) &= q_2(i) + a_2(i) - S_{g2} g_2(i) - S_{y2} y_2(i) \delta_2(i) \\
q_4(i+1) &= q_4(i) + a_4(i) - S_{g4} g_4(i) - S_{y4} y_4(i) \delta_2(i)
\end{aligned} \tag{4.10}$$

where

$q_j(i)$ = standing queue (veh) at the beginning of i^{th} time step on approach j ($j=1, 2, 3, 4$);

$a_j(i)$ = the number of vehicle arrivals (veh) during i^{th} time step on approach j ($j=1, 2, 3, 4$);

$g_j(i)$ = green time (sec) during i^{th} time step on approach j ($j=1, 2, 3, 4$);

$y_j(i)$ = yellow time (sec) during i^{th} time step on approach j ($j=1, 2, 3, 4$);

S_{gj} = saturation discharge rate (veh/sec) for green time on approach j ($j=1, 2, 3, 4$);

S_{yj} = saturation discharge rate (veh/sec) for yellow time on approach j ($j=1, 2, 3, 4$);

In matrix notation, we have

$$Q(i+1) = Q(i) + A(i) - S_g^T G(i) - S_y^T \Delta(i) Y \tag{4.11}$$

where

$$Q(i) = (q_1(i), q_2(i), q_3(i), q_4(i))^T$$

$$A(i) = (a_1(i), a_2(i), a_3(i), a_4(i))^T$$

$$G(i) = (g_1(i), g_2(i), g_3(i), g_4(i))^T \text{ and assume here green intervals } g_1(i) = g_3(i) \text{ and}$$

$$g_2(i) = g_4(i) \text{ for all time steps}$$

$$Y(i) = (y_1(i), y_2(i), y_3(i), y_4(i))^T \text{ and assume here yellow intervals } y_j(i) = Y; \quad j =$$

$$1, 2, 3, 4$$

$$S_g = \text{diag} (s_{g1}(i) , s_{g2}(i) , s_{g3}(i) , s_{g4}(i))_{4 \times 4}$$

$$S_y = \text{diag} (s_{y1}(i) , s_{y2}(i) , s_{y3}(i) , s_{y4}(i))_{4 \times 4}$$

$$\Delta(i) = (\delta_1(i), \delta_2(i), \delta_1(i), \delta_2(i))^\top$$

Physically, all green time intervals must be nonnegative at all time:

$$g_j(i) \geq 0 \quad j=1,2,3,4 \quad (4.12)$$

or in matrix notation

$$G(i) \geq 0 \quad (4.13)$$

In practice, there are minimum and maximum green time restrictions.

$$\sum_{j=1}^k g_1(i-j+1) \geq g_{1\min} \quad (4.14)$$

$$\sum_{j=1}^k g_1(i-j+1) \leq g_{1\max} \quad (4.15)$$

if $\prod_{j=1}^k g_1(i-j+1) \neq 0$; $g_1(i-1) = g_1(i-k) = 0$; and

$$g_2(i-j+1) = 0 \quad j=1,2, \dots, k$$

$$\sum_{j=1}^k [g_2(i-j+1)] \geq g_{2\min} \quad (4.16)$$

$$\sum_{j=1}^k [g_2(i-j+1)] \leq g_{2\max} \quad (4.17)$$

if $\prod_{j=1}^k g_2(i-j+1) \neq 0$; $g_2(i-1) = g_2(i-k) = 0$; and

$$g_1(i-j+1) = 0 \quad j=1,2, \dots, k$$

For calculation convenience, we transfer above continuous green restrictions to their

following equivalences.

Denote $\xi_1, \xi_2, \xi_3, \xi_4, \xi_5, \xi_6$ as parameters indicating continuous green duration history, and define $g_{c1}(i)$ and $g_{c2}(i)$ as the latest continuous green durations until time step i for main road and side road, respectively. Then above minimum and maximum green time restrictions can be represented as

$$[g_{c1}(i-1) + g_1(i)] \geq g_{1min} (1 - \xi_1) \quad (4.18)$$

$$[g_{c1}(i-1) + g_1(i)] (1 - \xi_2) \leq g_{1max} \quad (4.19)$$

$$[g_{c2}(i-1) + g_2(i)] \geq g_{2min} (1 - \xi_3) \quad (4.20)$$

$$[g_{c2}(i-1) + g_2(i)] (1 - \xi_4) \leq g_{2max} \quad (4.21)$$

where

$$g_{c1}(i) = [g_{c1}(i-1) + g_1(i)] (1 - \xi_5)$$

$$g_{c2}(i) = [g_{c2}(i-1) + g_2(i)] (1 - \xi_6)$$

$$\xi_1 = 1 \text{ if } g_1(i) \neq 0; \quad (\text{no need to check minimum green})$$

$$\xi_1 = 0 \text{ if } g_1(i)=0 \text{ and } g_{c1}(i-1) \neq 0 \quad (\text{need to check minimum green})$$

$$\xi_2 = 1 \text{ if } g_1(i)=0; \quad (\text{no need to check maximum green})$$

$$\xi_2 = 0 \text{ if } g_1(i) \neq 0 \quad (\text{need to check maximum green})$$

$$\xi_5 = 1 \text{ if } g_1(i) = 0 \quad (\text{reset: } g_{c1}(i) = 0)$$

$$\xi_5 = 0 \text{ if } g_1(i) \neq 0 \quad (\text{accumulate green time } g_{c1}(i))$$

$$\xi_3 = 1 \text{ if } g_2(i) \neq 0; \quad (\text{no need to check minimum green})$$

$$\xi_3 = 0 \text{ if } g_2(i)=0 \text{ and } g_{c2}(i-1) \neq 0 \quad (\text{need to check minimum green})$$

$$\xi_4 = 1 \text{ if } g_2(i)=0; \quad (\text{no need to check maximum green})$$

$$\xi_4 = 0 \text{ if } g_2(i) \neq 0 \quad (\text{need to check maximum green})$$

$$\xi_6 = 1 \text{ if } g_2(i)=0 \quad (\text{reset: } g_{c2}(i)=0)$$

$$\xi_6 = 0 \text{ if } g_2(i) \neq 0 \quad (\text{accumulate green time } g_{c2}(i))$$

In matrix notation above restrictions have the following forms

$$G_c(i-1) + G(i) \geq \Xi_{\min} G_{\min} \quad (4.22)$$

$$\Xi_{\max} [G_c(i-1) + G(i)] \leq G_{\max} \quad (4.23)$$

$$\text{where } G_c(i) = (g_{c1}(i), g_{c2}(i))^T = \Xi_c [G_c(i-1) + G(i)]$$

$$G_{\min} = (g_{1\min}, g_{2\min})^T$$

$$G_{\max} = (g_{1\max}, g_{2\max})^T$$

$$\Xi_{\min} = \text{diag}(1 - \xi_1, 1 - \xi_3)_{2 \times 2}$$

$$\Xi_{\max} = \text{diag}(1 - \xi_2, 1 - \xi_4)_{2 \times 2}$$

$$\Xi_c = \text{diag}(1 - \xi_5, 1 - \xi_6)_{2 \times 2}$$

There are also restrictions on queue length.

$$Q(i) \geq 0 \quad (4.24)$$

$$Q(i) \leq Q_{\max} \quad \text{and } Q(i) \text{ are integers} \quad (4.25)$$

For assessment of the control efficiency, define the performance index (PI) as

$$J = \Psi(N+1, X_{N+1}) + 0.5 \sum_{i=1}^N [X^T(i) R X(i) + U^T(i) S U(i)] \quad (4.26)$$

In the above equation, the superscript T represents transpose. $\Psi(N+1, X_{N+1})$ is a function of the final time (N+1) and corresponding state at the final time $X(N+1)$. For fixed final state and free final state situations $\Psi[N+1, X(N+1)]$ can be defined as follows, respectively.

$$\Psi[N+1, X(N+1)] = 0.5 X_{N+1}^T P X_{N+1} \quad (4.27)$$

$$\Psi[N+1, X(N+1)] = 0.5 [X(N+1) - X_{N+1}]^T P [X(N+1) - X_{N+1}] \quad (4.28)$$

and X_{N+1} is the desired final state.

$$X(i) = (q_1(i) / q_1, q_2(i) / q_2, q_3(i) / q_3, q_4(i) / q_4)^T \quad i = 1, 2, \dots, n$$

and q_1, q_2, q_3, q_4 are average queue lengths under normal control on approaches 1, 2, 3, and 4, respectively. In matrix notation, $Q = (q_1, q_2, q_3, q_4)^T$.

$$U(i) = (g_1(i) / g_1, g_2(i) / g_2)^T \quad i = 1, 2, \dots, n$$

and g_1, g_2 are average green times under normal control for main and side roads, respectively. In matrix notation, $G = (g_1, g_2)^T$.

$P_{4 \times 4}, P'_{4 \times 4}, R_{4 \times 4}, S_{2 \times 2}$ are weighting matrices. They are selected by user to weight terminal state, intermediate state, and control variable differently. Here, they are assumed symmetric and positive definite.

Rewrite (4.9), (4.11), (4.13), (4.22), (4.23), (4.24), (4.25) in terms of state variables

$X(i)$ and control variables $U(i)$.

$$T = W_u U(i) + \delta_1(i) [Y1 + AR1] + \delta_2(i) [Y2 + AR2] \quad (4.29)$$

where $W_u = (g_1, g_2)_{1 \times 2}$.

$$X(i+1) = X(i) + W_x^T [A(i) - S^T g W_u U(i) - S_y^T \Delta(i) Y] \quad (4.30)$$

where $W_x = \text{diag} (1/q_1, 1/q_2, 1/q_3, 1/q_4)_{4 \times 4}$ and

$$W_u = [(g_1, 0), (0, g_2), (g_1, 0), (0, g_2)]_{4 \times 2}^T$$

$$U(i) \geq 0 \quad (4.31)$$

$$V_u G_c(i-1) + U(i) \geq \Xi_{\min} U_{\min} \quad (4.32)$$

$$\Xi_{\max} [V_u G_c(i-1) + U(i)] \leq U_{\max} \quad (4.33)$$

where $V_u = \text{diag} (1/g_1, 1/g_2)_{2 \times 2}$;

$$U_{\min} = (g_{1\min} / g_1, g_{2\min} / g_2)^T \text{ and}$$

$$U_{\max} = (g_{1\max} / g_1, g_{2\max} / g_2)^T$$

$$X(i) \geq 0 \quad (4.34)$$

$$X(i) \leq X_{\max} \quad (4.35)$$

where $X_{\max} = (q_{1\max} / q_1, q_{2\max} / q_2, q_{3\max} / q_3, q_{4\max} / q_4)^T$

Denote the Lagrangian multipliers associated with (4.29) and (4.30) as $\sigma(i)$ and $\Lambda(i) = (\lambda_1(i), \lambda_2(i), \lambda_3(i), \lambda_4(i))$, respectively; and denote the Lagrangian multipliers associated with (4.31)- (4.35) as $\Phi_1(i) = (\varphi_1(i), \varphi_2(i))$, $\Phi_2(i) = (\varphi_3(i), \varphi_4(i))$, $\Phi_3(i) = (\varphi_5(i), \varphi_6(i))$, $\Phi_4(i) = (\varphi_7(i), \varphi_8(i), \varphi_9(i), \varphi_{10}(i))$, $\Phi_5(i) = (\varphi_{11}(i), \varphi_{12}(i), \varphi_{13}(i), \varphi_{14}(i))$. Then the

augmented Hamiltonian function H for our optimal control problem is constructed as

$$\begin{aligned}
 H = & 0.5 [X^T(i) R X(i) + U^T(i) S U(i)] \\
 & + \Lambda(i) W_x^T [A(i) - S_g^T W_u U(i) - S_y^T Y \Delta(i)] \\
 & + \sigma(i) \{ W_u U(i) + \delta_1(i) [Y1 + AR1] + \delta_2(i) [Y2 + AR2] - T \} \\
 & - \Phi_1(i) U(i) - \Phi_4 X(i) + \Phi_5 [X(i) - X_{max}] \\
 & - \Phi_2(i) [V_u G_c(i-1) + U(i) - \Xi_{min} U_{min}] \\
 & + \Phi_3(i) \{ \Xi_{max} [V_u G_c(i-1) + U(i)] - U_{max} \}
 \end{aligned} \tag{4.36}$$

The first order necessary conditions for the optimal control problem are as follows

(Ran, 1996; and Lewis, 1995):

- State equation

$$\hat{c}H / \hat{c}\Lambda(i) = X(i+1) - X(i) = W_x^T [A(i) - S_g^T W_u U(i) - S_y^T \Delta(i) Y] \tag{4.37}$$

- Costate equation

$$\hat{c}H / \hat{c}X(i) = -\Lambda^T(i+1) + \Lambda^T(i) = R X(i) - \Phi_4^T + \Phi_5^T \tag{4.38}$$

- Stationarity condition

$$\begin{aligned}
 \hat{c}H / \hat{c}U(i) &= 0 \\
 &= S U(i) - [\Lambda(i) W_x^T S_g^T W_u]^T + [\sigma(i) W_u]^T - \Phi_1^T - \Phi_2^T + \Xi_{max}^T \Phi_3^T \\
 &= S U(i) - W_u^T S_g W_x \Lambda^T(i) + W_u^T \sigma(i) - \Phi_1^T - \Phi_2^T + \Xi_{max}^T \Phi_3^T
 \end{aligned} \tag{4.39}$$

- Equality constraint

$$\hat{c}H / \hat{c}\sigma(i) = 0 = U^T(i) W_u^T + \delta_1(i) [Y1 + AR1] + \delta_2(i) [Y2 + AR2] - T$$

$$(4.40)$$

- Inequality constraints

$$\partial H / \partial \Phi_j(i) \leq 0 \quad j=1 \text{ to } 5 \quad (4.41)-(4.45)$$

$$\text{and } \varphi_i(i) \partial H / \partial \varphi_i(i) = 0 \quad j=1 \text{ to } 14 \quad (4.46)$$

- Boundary conditions

$$\text{Given initial state: } X(1) = X_1 \quad (4.47)$$

$$\text{Given final state: } X(N+1) = X_{N+1} \text{ or} \quad (4.48)$$

$$\text{Free final state: } \Lambda^T(N+1) =$$

$$\begin{aligned} & \partial \{0.5 [X(N+1) - X_{N+1}]^T P^* [X(N+1) - X_{N+1}]\} / \partial X(N+1) \\ & = P^* [X(N+1) - X_{N+1}] \end{aligned} \quad (4.49)$$

Equation (4.47) represents that transition 2 starts at a known initial state X_1 , which is the final queue state on all approaches to an intersection after the EV passes through the intersection. Equation (4.48) represents that transition 2 ends at a fixed final state X_{N+1} , which is the desired queue state after EVSP. If a certain deviation from the desired queue state is acceptable, and the deviation will benefit the transition procedure, the free-final-state condition (4.49) can be used to allow the variation of X_{N+1} in determining the optimal solution.

In summary, the solution to the following time-invarying dynamic system is the

optimal control strategy for our transition 2 operation.

$$X(i+1) - X(i) = W_x^T [A(i) - S_g^T W_u^T U(i) - S_y^T \Delta(i) Y] \quad (4.37)$$

$$-\Lambda^T(i+1) + \Lambda^T(i) - R X(i) - \Phi_4^T + \Phi_5^T \quad (4.38)$$

$$S U(i) - W_u^T S_g W_x \Lambda^T(i) + W_u^T \sigma(i) - \Phi_1^T - \Phi_2^T + \Xi_{\max}^T \Phi_3^T = 0 \quad (4.39)$$

$$U^T(i) W_u^T + \delta_1(i) [Y1 + AR1] + \delta_1(i) [Y2 + AR2] - T = 0 \quad (4.40)$$

$$U(i) \geq 0 \quad (4.41)$$

$$Vu Gc(i-1) + U(i) \geq \Xi_{\min} U_{\min} \quad (4.42)$$

$$\Xi_{\max} [Vu Gc(i-1) + U(i)] \leq U_{\max} \quad (4.43)$$

$$X(i) \geq 0 \quad (4.44)$$

$$X(i) \leq X_{\max} \quad (4.45)$$

$$\varphi_j(i) \partial H / \partial \varphi_j(i) = 0 \quad j=1 \text{ to } 14 \quad (4.46)$$

$$X(1) = X_1 \quad (4.47)$$

$$X(N+1) = X_{N+1} \text{ or } \quad (4.48)$$

$$\Lambda^T(N+1) = P^* [X(N+1) - X_{N+1}] \quad (4.49)$$

4.4 Theoretical analysis of the optimal control model

Equation (4.46) implies that $\varphi_j(i) = 0$ if $\partial H / \partial \varphi_j(i) < 0$ and $\varphi_j(i) > 0$ if $\partial H / \partial \varphi_j(i) = 0$.

Since (4.42) and (4.43) cannot be both active (i.e., left side = right side), we will have

$$\varphi_3(i)=0 \text{ or } \varphi_5(i)=0 \text{ or both}$$

$$\varphi_4(i)=0 \text{ or } \varphi_6(i)=0 \text{ or both}$$

Similarly, (4.44) and (4.45) cannot be both active, and we will have

$$\varphi_7(i)=0 \text{ or } \varphi_{11}(i)=0 \text{ or both}$$

$$\varphi_8(i)=0 \text{ or } \varphi_{12}(i)=0 \text{ or both}$$

$$\varphi_9(i)=0 \text{ or } \varphi_{13}(i)=0 \text{ or both}$$

$$\varphi_{10}(i)=0 \text{ or } \varphi_{14}(i)=0 \text{ or both}$$

From equation (4.38) and after a few operations, we have

$$\varphi_7(i) - \varphi_{11}(i) = \lambda_1(i+1) - \lambda_1(i) + \sum_{j=1}^4 r_{1j} x_j(i),$$

$$\varphi_8(i) - \varphi_{12}(i) = \lambda_2(i+1) - \lambda_2(i) + \sum_{j=1}^4 r_{2j} x_j(i),$$

$$\varphi_9(i) - \varphi_{13}(i) = \lambda_3(i+1) - \lambda_3(i) + \sum_{j=1}^4 r_{3j} x_j(i),$$

$$\varphi_{10}(i) - \varphi_{14}(i) = \lambda_4(i+1) - \lambda_4(i) + \sum_{j=1}^4 r_{4j} x_j(i),$$

If the right side of each equation is greater than zero, respectively, we must have

$$\varphi_7(i) > 0 \text{ and } \varphi_{11}(i) = 0$$

$$\varphi_8(i) > 0 \text{ and } \varphi_{12}(i) = 0$$

$$\varphi_9(i) > 0 \text{ and } \varphi_{13}(i) = 0$$

$$\varphi_{10}(i) > 0 \text{ and } \varphi_{14}(i) = 0$$

They mean that $x_1(i)=0$, $x_2(i)=0$, $x_3(i)=0$, $x_4(i)=0$, respectively.

If the right side of each equation is less than zero, respectively, we must have

$$\varphi_7(i) = 0 \text{ and } \varphi_{11}(i) > 0$$

$$\varphi_8(i) = 0 \text{ and } \varphi_{12}(i) > 0$$

$$\varphi_9(i) = 0 \text{ and } \varphi_{13}(i) > 0$$

$$\varphi_{10}(i) = 0 \text{ and } \varphi_{14}(i) > 0$$

They mean that $x_1(i) = x_{1\max}$, $x_2(i) = x_{2\max}$, $x_3(i) = x_{3\max}$, $x_4(i) = x_{4\max}$, respectively.

Similarly, from equation (4.39) and after a few operations, we have

$$\varphi_3(i) - \varphi_5(i) (1 - \xi_2) = \sum_j i^2 s_{1j} u_j(i) - \lambda_1(i) g_1 s_{g1} / q_1 + g_1 \sigma(i) - \varphi_1(i)$$

$$\begin{aligned} \varphi_4(i) - \varphi_6(i) (1 - \xi_4) = \sum_j i^2 s_{2j} u_j(i) - [\lambda_2(i) g_2 s_{g2} / q_2 + \lambda_3(i) g_2 s_{g3} / q_3] \\ + g_2 \sigma(i) - \varphi_2(i) \end{aligned}$$

If the right side of each equation is greater than zero, respectively, we must have

$$\varphi_3(i) > 0 \text{ and } \varphi_5(i) = 0$$

$$\varphi_4(i) > 0 \text{ and } \varphi_6(i) = 0$$

They mean that the following equations hold, respectively.

$$[g_{c1}(i-1) + g_1(i)] = g_{1\min} (1 - \xi_1)$$

$$[g_{c2}(i-1) + g_2(i)] = g_{2\min} (1 - \xi_3)$$

Therefore, we have results:

if $\xi_1 = 1$ then $g_1(i) = g_{c1}(i-1) = 0$; and if $\xi_1 = 0$ then $g_1(i) = 0$, and $g_{c1}(i-1) = g_{1min}$

if $\xi_2 = 1$ then $g_2(i) = g_{c2}(i-1) = 0$; and if $\xi_2 = 0$ then $g_2(i) = 0$, and $g_{c2}(i-1) = g_{2min}$

If the right side of each equation is less than zero, respectively, we must have

$$\varphi_3(i) = 0 \text{ and } \varphi_5(i) > 0 \text{ and } \xi_2 = 0$$

$$\varphi_4(i) = 0 \text{ and } \varphi_6(i) > 0 \text{ and } \xi_3 = 0$$

They mean that the following equations hold, respectively.

$$[g_{c1}(i-1) + g_1(i)] = g_{1max}$$

$$[g_{c2}(i-1) + g_2(i)] = g_{2max}$$

Therefore, we have results:

$$g_1(i) = g_{1max} - g_{c1}(i-1)$$

$$g_2(i) = g_{2max} - g_{c2}(i-1)$$

Unfortunately, we cannot obtain a closed-form analytic solution to our established dynamic system, although we can deduce some helpful results. We have to resort to another numerical method for the solution to the optimal control problem. In the following section, a two-phase algorithm will be introduced for this purpose.

4.5 A Two-phase Algorithm

The difficulty in solving our optimal control problem results from the interrelationship between sequences of greens, yellows and all reds in successive time intervals. The interrelationship means that the sequence of control variables $u_1(i-1)$, and $u_2(i-1)$ (or equivalently $g_1(i-1)$, and $g_2(i-1)$) in time step $i-1$ influences the sequence of control variables $u_1(i)$, and $u_2(i)$ in time step i .

During the first phase of our two-phase algorithm, we will consider only: State equation (4.37), Costate equation (4.38), Stationarity condition (4.39), Equality constraint (4.40), and Boundary conditions (4.47) and (4.48) or (4.49). This is a relaxation process in which inequality constraints and the interrelationship between successive sequences are neglected. The solution to the relaxed problem must be a low bound of the solution to our original problem. During the second phase, inequality constraints (4.41)-(4.46) and the interrelationship between successive sequences are introduced to determine the practical solution, which satisfies all the restrictions imposed.

4.5.1 Phase 1

Because inequality constraints are excluded from the consideration for this phase, let $\Phi_1 = \Phi_2 = \Phi_3 = \Phi_4 = \Phi_5 = 0$ in (4.38) and (4.39). Rewriting (4.37) to (4.40) as (4.50) to

(4.53) we have

$$\partial H / \partial \Lambda(i) = X(i+1) - X(i) = W_x^T [A(i) - S_g^T W_u U(i) - S_y^T \Delta(i) Y] \quad (4.50)$$

$$\partial H / \partial X(i) = -\Lambda^T(i+1) + \Lambda^T(i) = R X(i) \quad (4.51)$$

$$\partial H / \partial U(i) = 0 = S U(i) - W_u^T S_g W_x \Lambda^T(i) + W_u^T \sigma(i) \quad (4.52)$$

$$\partial H / \partial \sigma(i) = 0 = U^T(i) W_u^T + \delta_1(i) [Y1 + AR1] + \delta_1(i) [Y2 + AR2] - T \quad (4.53)$$

Equation (4.51) holds for every time step from $i=1$ to $N+1$.

$$-\Lambda^T(2) + \Lambda^T(1) = R X(1) \quad (4.54-1)$$

$$-\Lambda^T(3) + \Lambda^T(2) = R X(2) \quad (4.54-2)$$

...

$$-\Lambda^T(N+1) + \Lambda^T(N) = R X(N) \quad (4.54-N)$$

Adding the left and the right sides of all the equations, respectively results in:

$$-\Lambda^T(N+1) + \Lambda^T(1) = \sum_{i=1}^N R X(i) \quad (4.55)$$

and then:

$$\Lambda^T(1) = \Lambda^T(N+1) + \sum_{i=1}^N R X(i) \quad (4.56-1)$$

All the other $\Lambda^T(i)$ can be obtained recursively from (4.54-i):

$$\begin{aligned} \Lambda^T(2) &= \Lambda^T(1) - R X(1) \\ &= \Lambda^T(N+1) + \sum_{i=2}^N R X(i) \end{aligned} \quad (4.56-2)$$

$$\Lambda^T(3) = \Lambda^T(N+1) + \sum_{i=3}^N R X(i) \quad (4.56-3)$$

...

$$\Lambda^T(N) = \Lambda^T(N+1) + R X(N) \quad (4.56-N)$$

Solving (4.52) for $U(i)$ in terms of $\Lambda(i)$ and $\sigma(i)$ yields

$$U(i) = S^{-1} [W_u^T S_g W_x \Lambda^T(i) - W_u^T \sigma(i)] \quad (4.57)$$

Eliminating $U(i)$ in (4.50) using (4.57) gives:

$$X(i+1) - X(i) = W_x^T \{ A(i) - S_g^T W_u S^{-1} [W_u^T S_g W_x \Lambda^T(i) - W_u^T \sigma(i)] - S^T y \Delta(i) Y \} \quad (4.58)$$

Using (4.56-i) in (4.58) gets:

$$\begin{aligned} X(i+1) - X(i) = & W_x^T \{ A(i) - S_g^T W_u S^{-1} [W_u^T S_g W_x \Lambda^T(i) - W_u^T \sigma(i)] - S^T y \Delta(i) Y \} = \\ & W_x^T \{ A(i) - S_g^T W_u S^{-1} [W_u^T S_g W_x \Lambda^T(N+1) + W_u^T S_g W_x \sum_{j=i}^N R X(j) - \\ & W_u^T \sigma(i)] - S^T y \Delta(i) Y \} \end{aligned} \quad (4.59)$$

Adding the left and the right sides of (4.59) for $i=1$ to N , respectively results in:

$$\begin{aligned} X(N+1) - X(1) = & W_x^T \{ \sum_{i=1}^N A(i) - S_g^T W_u S^{-1} [W_u^T S_g W_x \Lambda^T(N+1)N + \\ & W_u^T S_g W_x \sum_{i=1}^N R X(i) - W_u^T \sum_{i=1}^N \sigma(i)] - S^T y Y \sum_{i=1}^N \Delta(i) \} \end{aligned} \quad (4.60)$$

Given initial state and given final state

$$\text{Given initial state: } X(1) = X_1 \quad (4.61)$$

$$\text{Given final state: } X(N+1) = X_{N+1} \quad (4.62)$$

Solving (4.60) for $\Lambda^T(N+1)$ in terms of the given $X(1)$, the known desired $X(N+1)$ and other $X(i)$ yields:

$$\begin{aligned} \Lambda^T(N+1) = & \\ & W_x^{-1} S_g^{-1} W_u^{-T} \{ -S W_u^{-1} S_g^T [W_x^{-T} X(N+1) - W_x^{-T} X(1) - \sum_{i=1}^N A(i) + \\ & S_y^T Y \sum_{i=1}^N \Delta(i)] - W_u^{-T} S_g W_x \sum_{i=1}^N R X(i) + W_u^T \sum_{i=1}^N \sigma(i) \} / N \} \end{aligned} \quad (4.63)$$

All the other $\Lambda^T(i)$ can be obtained in terms of $X(i)$, $\Delta(i)$ and $\sigma(i)$ using (4.56-i). Substituting all the $\Lambda^T(i)$ in (4.58) gives N simultaneous equations with N unknown $X(i)$, and consequently we obtain $X(i)$ in terms of $\Delta(i)$ and $\sigma(i)$. Equations (4.63) and then (4.56-i) give all the $\Lambda^T(i)$ with known $X(i)$. Then from (4.57) we get $U(i)$ in terms of $\Delta(i)$ and $\sigma(i)$.

Equations (4.52) and (4.53) constitute $3N$ simultaneous equations, and we can solve them for $3N$ unknown $u_1(i)$, $u_2(i)$, and $\sigma(i)$ ($i=1$ to N) in terms of $\delta_1(i)$ and $\delta_2(i)$.

$$\partial H / \partial u_1(i) = 0 = S_{11} u_1(i) + S_{12} u_2(i) - \lambda_1(i) S_{g1} g_1 / q_1 - \lambda_3(i) S_{g3} g_1 / q_3 + g_1 \sigma(i) \quad (4.52-1)$$

$$\partial H / \partial u_2(i) = 0 = S_{21} u_1(i) + S_{22} u_2(i) - \lambda_2(i) S_{g2} g_2 / q_2 - \lambda_4(i) S_{g4} g_2 / q_4 + g_2 \sigma(i) \quad (4.52-2)$$

$$\partial H / \partial \sigma(i) = 0 = u_1(i) g_1 + u_2(i) g_2 + \delta_1(i) [Y1 + AR1] + \delta_2(i) [Y2 + AR2] - T \quad (4.53)$$

Given initial state and free final state

$$\text{Given initial state: } X(1) = X_1 \quad (4.64)$$

$$\text{Free final state: } \Lambda^T(N+1) = P^* [X(N+1) - X_{N+1}] \quad (4.65)$$

Eliminating $\Lambda^T(N+1)$ in (4.60) using (4.65) and then solving (4.60) for $X(N+1)$ in terms of the given $X(1)$, the known desired X_{N+1} and other $X(i)$ yields:

$$\begin{aligned} X(N+1) = & [W_N^{-T} + S_g^T W_u^* S^{-1} W_u^{*T} S_g W_N P^* N]^{-1} \{ W_N^{-T} X(1) + \sum_{i=1}^N A(i) - \\ & S_y^T Y \sum_{i=1}^N \Delta(i) - S_g^T W_u^* S^{-1} [-W_u^{*T} S_g W_N P^* X_{N+1} N + \\ & W_u^{*T} S_g W_N \sum_{i=1}^N R X(i) (i) - W_u^{*T} \sum_{i=1}^N \sigma(i)] \} \end{aligned} \quad (4.66)$$

Then $\Lambda^T(N+1)$ can be obtained in terms of the given $X(1)$, the known desired X_{N+1} and other $X(i)$ from (4.65). All the other $\Lambda^T(i)$ can be obtained in terms of $X(i)$, $\Delta(i)$ and $\sigma(i)$ using (4.56-i). Substituting all the $\Lambda^T(i)$ in (4.58) gives N simultaneous equations with N unknown $X(i)$, and consequently we obtain $X(i)$ in terms of $\Delta(i)$ and $\sigma(i)$. (4.66), (4.65) and then (4.56-i) give all the $\Lambda^T(i)$ with known $X(i)$. Then from (4.57) we get $U(i)$ in terms of $\Delta(i)$ and $\sigma(i)$.

Similar to the situation of fixed initial state and fixed final state, (4.52) and (4.53) constitute $3N$ simultaneous equations, and we can solve them for $3N$ unknown $u_1(i)$, $u_2(i)$, and $\sigma(i)$ ($i=1$ to N) in terms of $\delta_1(i)$ and $\delta_2(i)$.

$$\hat{C}H / \hat{C}u_1(i) = 0 = S_{11} u_1(i) + S_{12} u_2(i) - \lambda_1(i) S_{g1} g_1 / q_1 - \lambda_3(i) S_{g3} g_1 / q_3 + g_1 \sigma(i) \quad (4.52-1)$$

$$\hat{C}H / \hat{C}u_2(i) = 0 = S_{21} u_1(i) + S_{22} u_2(i) - \lambda_2(i) S_{g2} g_2 / q_2 - \lambda_4(i) S_{g4} g_2 / q_4 + g_2 \sigma(i) \quad (4.52-2)$$

$$\hat{C}H / \hat{C}\sigma(i) = 0 = u_1(i) g_1 + u_2(i) g_2 + \delta_1(i) [Y1 + AR1] + \delta_2(i) [Y2 + AR2] - T \quad (4.53)$$

4.5.2 Phase 2

After control/state variables are solved in terms of $\delta_1(i)$ and $\delta_2(i)$, the next task is to determine the values of $\delta_1(i)$ and $\delta_2(i)$, and then the practical optimal control parameters, which satisfy all the constraints.

For each time step i , the sequence of control variables can be one of the fourteen situations listed in Table 4.1.

Once the sequence in a time step is determined, the next sequence is not free and it must follow certain rules. These rules can be illustrated in the tree structures illustrated in Fig.4.3.

Table 4.1 Possible signal indication sequences for a time step

Sequence #	Indication Sequence	$g_1(i)$	$\delta_1(i)$	$g_2(i)$	$\delta_2(i)$	Remark
Seq(1) _i	g_1	T	0	0	0	
Seq(2) _i	$g_1 \delta_1$	>0	1	0	0	
Seq(3) _i	$g_1 \delta_1 g_2$	>0	1	>0	0	
Seq(4) _i	$g_1 \delta_1 g_2 \delta_2$	>0	1	>0	1	
Seq(5) _i	g_2	0	0	T	0	
Seq(6) _i	$g_2 \delta_2$	0	0	>0	1	
Seq(7) _i	$g_2 \delta_2 g_1$	>0	0	>0	1	
Seq(8) _i	$g_2 \delta_2 g_1 \delta_1$	>0	1	>0	1	
Seq(9) _i	$\delta_1 g_2$	0	1	>0	0	
Seq(10) _i	$\delta_1 g_2 \delta_2$	0	1	>0	1	
Seq(11) _i	$\delta_1 g_2 \delta_2 g_1$	>0	1	>0	1	
Seq(12) _i	$\delta_2 g_1$	>0	0	0	1	
Seq(13) _i	$\delta_2 g_1 \delta_1$	>0	1	0	1	
Seq(14) _i	$\delta_2 g_1 \delta_1 g_2$	>0	1	>0	1	

Notes: g_1 =green for main road;

g_2 =green for side road;

δ_1 =yellow and all red-1 for main road; and

δ_2 =yellow and all red-2 for side road.

Fig. 4.3 Tree structures for successive signal sequences (continued on next page)

Fig. 4.3 Tree structures for successive signal sequences

In a simple way, the above rules can be summarized as follows.

Rule 1: If the last signal indication for time step i is $g_1(i)$, then the sequence for time step

$i+1$ must be $\text{Seq}(1)_{i+1}$, $\text{Seq}(2)_{i+1}$, $\text{Seq}(3)_{i+1}$, $\text{Seq}(4)_{i+1}$, $\text{Seq}(9)_{i+1}$, $\text{Seq}(10)_{i+1}$, or $\text{Seq}(11)_{i+1}$

Rule 2: If the last signal indication for time step i is $g_2(i)$, then the sequence for time step

$i+1$ must be $\text{Seq}(5)_{i+1}$, $\text{Seq}(6)_{i+1}$, $\text{Seq}(7)_{i+1}$, $\text{Seq}(8)_{i+1}$, $\text{Seq}(12)_{i+1}$, $\text{Seq}(13)_{i+1}$, or $\text{Seq}(14)_{i+1}$

Rule 3: If the last signal indication for time step i is $\delta_1(i)$, then the sequence for time step

$i+1$ must be $\text{Seq}(5)_{i+1}$, $\text{Seq}(6)_{i+1}$, $\text{Seq}(7)_{i+1}$, or $\text{Seq}(8)_{i+1}$

Rule 4: If the last signal indication for time step i is $\delta_2(i)$, then the sequence for time step

$i+1$ must be $\text{Seq}(1)_{i+1}$, $\text{Seq}(2)_{i+1}$, $\text{Seq}(3)_{i+1}$, or $\text{Seq}(4)_{i+1}$

Consequently, we can adopt the control strategy of broad-first search in Artificial Intelligence (AI) (Schalkoff, 1990; Russell et al., 1995) to generate and then check all the sequences for feasibility. From all the feasible options, the optimal one is selected to control traffic indications. The algorithm is shown in Fig. 4.4 and the corresponding data for each time step are summarized in Table 4.2.

Fig. 4.4 Flow chart for determination of optimal signal sequences

Table 4.2 Data required in search for feasible signal sequences

Total # of options for time step i: TN(i)			
Sequence # of option 1	Control variables for option 1: $u_1(i)$, $u_2(i)$, $\delta_1(i)$ and $\delta_2(i)$	Performance index for option 1	Total # of sub-options for option 1
Sequence # of option 2	Control variables for option 2: $u_1(i)$, $u_2(i)$, $\delta_1(i)$ and $\delta_2(i)$	Performance index for option 2	Total # of sub-options for option 2
...	...	...	...
Sequence # of option TN(i)	Control variables for option TN (i): $u_1(i)$, $u_2(i)$, $\delta_1(i)$ and $\delta_2(i)$	Performance index for option TN(i)	Total # of sub-options for option TN(i)

Note: Time step $i=1$ to N .

4.6 Reduction of computation complexity

Theoretically, if the number of time steps (N) is large enough and therefore the time interval (T) is small enough, we can get the optimal control strategy as accurately as required. However, a large N results in the implementation problem for real-time control because of the computation complexity. As we can see from the sequence rules, there are four or seven possibilities of sequences for each preceding sequence. Consequently, the search space grows in an exponential time at about 6^N for N time steps. It will prevent our algorithm from practical application in a real-time mode in the situations with a large N .

To reduce the computation complexity, a stepwise search strategy is introduced in this research. In this stepwise search strategy, only M time steps out of the total N steps

are considered at a time, and the corresponding search space will be about 6^M possibilities. After one M-step model is solved for optimal control variables, only those parameters of the first time step within the M steps are implemented. Then the next M time steps beginning with the second time step in last M-step model consist of our next M-step model. In such a stepwise manner, we can solve all the control parameters in a reduced search space of about $N \cdot 6^M$ possibilities.

In total, $(N - M + 1)$ M-step models will be needed to solve a control problem with N time steps. The stepwise search strategy is illustrated in Fig. 4.5.

Accompanying the stepwise search strategy is that we also need to know all the other sequences of control variables following current M-step sequences. A natural thought is to set $\delta_1 = \delta_2 = 1$, and allocate proportionally green time to each phase according to their green durations under normal operation, i.e.,

$$g_1(i) = g_1 * (T - Y1 - Y2 - AR1 - AR2) / (g_1 + g_2)$$

$$g_2(i) = g_2 * (T - Y1 - Y2 - AR1 - AR2) / (g_1 + g_2)$$

since these combinations of control variables can follow any control sequence and it is not a bad preliminary prediction of green time distribution based on normal operation

Fig. 4.5 M-step models for stepwise search strategy

4.7 Algorithm summary

In summary, the algorithm for determining the control variables is described in Fig.

4.6.

Fig. 4.6 Flow chart for determination of optimal control variables

4.8 Computation consideration of data structure for M-step model

4.8.1 Data structure for control sequences

The control sequences in the M-step model will be produced dynamically, and then the M-step model is solved for the control parameters of the first time step. To achieve the solution strategy, a proper data structure has a critical role to play. Corresponding to the dynamic process of control sequence generation and model solution, we construct the data structure for the M-step model at time step i as follows.

Level 1:

Position	content	Remark
1	option #	
	$g_1(i)$	To be filled after all sub-options are solved and used as optimal strategy for this option.
	$g_2(i)$	
	$q_1(i)$	
	$q_2(i)$	
6	$q_3(i)$	
	$q_4(i)$	
	$\delta_1(i)$	
	$\delta_2(i)$	
	PI(i)	
11	Sub_TN_#	
	gc_1(i)	Excluding $g_1(i)$
	gc_2(i)	Excluding $g_2(i)$
14~20		Reserved for future use
21	Pointer to the first sub_option	

Notes: $q_1(i) \sim q_4(i)$ are queue lengths on approaches 1 to 4, respectively, after this signal control strategy (corresponding to $g_1(i)$, $g_2(i)$, $\delta_1(i)$, and $\delta_2(i)$) is executed.

Level 2:

Position	content	Remark
1	option #	
	$g_1(i+1)$	To be filled after all sub-options are solved and used as optimal strategy for this option.
	$g_2(i+1)$	
	$q_1(i+1)$	
	$q_2(i+1)$	
6	$q_3(i+1)$	
	$q_4(i+1)$	
	$\delta_1(i+1)$	
	$\delta_2(i+1)$	
	$PI(i+1)$	
11	Sub_TN #	
	$gc_1(i+1)$	Excluding $g_1(i+1)$
	$gc_2(i+1)$	Excluding $g_2(i+1)$
14~20		Reserved for future use
21	Pointer to the first sub_option	

.....

Level M:

Position	Content	Remark
1	option #	
	$g_1(i+M-1)$	To be filled after all sub-options are solved and used as optimal strategy for this option.
	$g_2(i+M-1)$	
	$q_1(i+M-1)$	
	$q_2(i+M-1)$	
6	$q_3(i+M-1)$	
	$q_4(i+M-1)$	
	$\delta_1(i+M-1)$	
	$\delta_2(i+M-1)$	
	$PI(i+M-1)$	
11	Sub_TN #	
	$gc_1(i+M-1)$	Exclude $g_1(i+M-1)$
	$gc_2(i+M-1)$	Exclude $g_2(i+M-1)$
14~20		Reserved for future use
21	Pointer to the first sub_option	

Level M+1 and other levels after it:

Position	Content	Remark
1	Total time steps needed	Predetermined.
	$g_1(i+M)$	
	$g_2(i+M)$	
8	$\delta_1(i+M)$	
	$\delta_2(i+M)$	

Notes: 1. $g_1(i+M)$ and $g_2(i+M)$ are fixed and predetermined.

2. $\delta_1(i+M) = \delta_2(i+M) = 1$.

4.8.2 Data structure for unknowns

For the level corresponding to initial condition we need to solve for the following unknowns:

Position	Content	Remark
1	$g_1(i)$	i is the first time step for each M-step model.
	$g_2(i)$	
3	$q_1(i)$	$q_1(i) \sim q_4(i)$ are queues before $g_1(i)$ and $g_2(i)$ are executed.
	$q_2(i)$	
	$q_3(i)$	
	$q_4(i)$	
7	$q_1(i+1)$	$q_1(i+1) \sim q_4(i+1)$ are queues after $g_1(i)$ and $g_2(i)$ are executed.
	$q_2(i+1)$	
	$q_3(i+1)$	
	$q_4(i+1)$	
11	$\lambda_1(i)$	
	$\lambda_2(i)$	
	$\lambda_3(i)$	
	$\lambda_4(i)$	
15	$\lambda_1(i+1)$	
	$\lambda_2(i+1)$	
	$\lambda_3(i+1)$	
	$\lambda_4(i+1)$	
19	$\sigma(i)$	

Exactly speaking, $q_1(i)$, $q_2(i)$, $q_3(i)$, and $q_4(i)$ are known because they are given as initial conditions for each M-step model. However, they are represented as unknowns here in order to keep a uniform format for solving the equation.

For each other level i we need to solve for the following unknowns:

Position	Content	Remark
1	$g_1(i)$	
	$g_2(i)$	
3	$q_1(i+1)$	$q_1(i) \sim q_4(i)$ have been solved in previous level.
	$q_2(i+1)$	
	$q_3(i+1)$	
	$q_4(i+1)$	
7	$\lambda_{1,i+1}$	$\lambda_{1,i} \sim \lambda_{4,i}$ have been solved in previous level.
	$\lambda_{2,i+1}$	
	$\lambda_{3,i+1}$	
	$\lambda_{4,i+1}$	
11	$\sigma(i)$	

4.9 Can the M-step model solve original optimal control problem?

A question we must answer is that “is the solution to the M-step model the same as, or approaching to, the solution to our original optimal control problem?”

For the situations with fixed initial and final states, our original optimal control model is an open-loop control. It means that the optimal control variables are dependent only on the fixed initial state and the fixed final state as long as the values of intermediate states do not deviate from the optimal state trajectory (Lewis, 1995: 47-50). It is almost certain that such a deviation will occur because of the replacement of M-step model for the original N-step model. As a result, we conclude that the solution to our original control problem is different from that to the M-step model.

Fortunately, for the situations with fixed initial state and free final state, our original optimal control problem is a closed-loop control, and any deviations from the optimal state trajectory can be automatically accounted for (Lewis, 1995: 52-59). This characteristic of feedback control gives our M-step model an opportunity to dynamically adjust control variables and approach to the optimal solution to the original optimal control problem.

Because of the above consideration, in all the following discussions we will focus only on the free final state situations.

4.10 Comparison between M-step model and Rolling Horizon strategy

We can also find the similarities between our M-step model and the Rolling Horizon strategy for adaptive control systems (Miller, 1963; Newell, 1998; Conrad et al., 1998; Gartner et al., 2002). Both of them calculate the optimal traffic control parameters over a period of time (M time steps for the M-step model, and a projection horizon for the Rolling Horizon strategy, respectively), and only those parameters for a portion of the entire calculation period of time (the first time step for M-step model, and the rolling period for the Rolling Horizon strategy, respectively) are implemented. The difference between them lie in that the Rolling Horizon strategy aims at dynamic revisions of switching decisions in the process of continuous recent data becoming available, while

the M-step model is used for reducing the search space and increasing computation efficiency. This mathematical problem solving technique can also be used in other real-time control systems as well as in rolling horizon-based system.

Chapter 5 Simulation and Comparison of Control Strategies for Transition 1

Software development environment, MATLAB (The Math Work Inc., 2003), is used for numerical simulation in this research. MATLAB is an interactive software system for numerical computations and graphics. It provides a high-level programming language for software development. In addition to many other numerical computing capabilities, its built-in state-of-the-art subroutine libraries for solving linear equations and eigenvalue problems make MATLAB especially suitable for users in the field of engineering applications and research.

5.1 Random generation of vehicle arrivals

It is assumed in this research that the arrivals of vehicles at the first intersection of a coordinated system follow a discrete random distribution. It means that each arrival is completely independent of any other arrivals, and equal intervals of time are equally like to contain equal numbers of arrivals.

With a given vehicle arrival volume Q (veh/h), the arrival rate will be $q=Q/3600$ (veh/sec), or the average interval between two arrivals is $1/q$ (sec/veh). We adopt the

random range between zero and one, and then the average value of the random numbers will be 0.5. If we produce a random number and then generate a vehicle arrival only when the random number is equal to or greater than 0.5, we have to double the arrival rate, or equivalently the interval between two arrivals should be $1/2q$ (sec/veh) so that the total vehicle arrivals generated during one hour will be

$$0.5 \times 3600 / (1/2q) = 0.5 \times 3600 \times 2q = 3600q = Q \text{ (veh/h)}$$

The generation algorithm of vehicle arrivals is presented in Fig. 5.1.

5.2 Signal timing plans for normal operation

Generally, signal timing plans for normal operation can be pre-timed, traffic-actuated, or adaptive. For our research their differences will affect only the control performances before the EVSP is carried out. To focus on the control of EVSP and without losing its applicability to other control strategies, we adopt the pre-timed plan for normal operation in this research.

The pre-timed plan controls the signals according to a set of predetermined schedules (fixed cycles, splits, and offsets), based on historic traffic data or patterns. An off-line optimization process usually is conducted to produce the predetermined schedules for advanced control systems. For example, the TOD (Time-Of-Day) timing plans belong to this category. During each time-of-day period, traffic flow is averaged and this constant

Fig. 5.1 Flow chart for generation of vehicle arrivals.

averaged flow is used to produce corresponding signal plan (Jagannathan and Khan, 2001).

Webster's method is used to determine the signal timing (Khisty et al., 1998; Garber et al., 2002). The layout of an intersection and related traffic data are shown in Figure 5.2 (revised from Example 8.5, Garber et al., 2002). In Figure 5.2, the number of lanes on each approach can be one, two, or more than two. However, all the lanes on each approach are in one lane group.

Fig. 5.2 An intersection layout and related traffic volumes

1. Given parameters:

- Saturation flow = 2000 passenger cars / hour of green time / approach;

- Left-turn rate = 4%, left-turn factor = 1.4;
- Right-turn rate = 6%, right-turn factor = 1.2;
- Traffic volumes on individual approaches as shown in Figure 5.2;
- Phase system: two-phase signal system, as shown in Fig. 5.3;

Fig. 5.3 Two-phase signal system

- Discharge rate during yellow interval = saturation flow / 2;
- Lost time per phase = 3 sec;
- Yellow (amber) time = 3 sec; ($\tau_1 = \tau_2 = 3$ sec)
- All red time = 2 sec; ($AR1 = AR2 = 1$ sec);

2. Determine fixed time signal control parameters using Webster's method.

- Convert the mixed volumes to equivalent straight-through passenger cars.

Approach 1: $V_1 = 1.0 \times 1000 \times (1 - 0.04 - 0.06) + 1.4 \times 1000 \times 0.04 + 1.2 \times 1000 \times 0.06$

$$= (1 + 0.4 \times 0.04 + 0.2 \times 0.06) \times 1000 = 1.028 \times 1000$$

$$= 1028 \text{ (veh/hr)}$$

Approach 2: $V_2 = 1.028 \times 400 = 411.2 \approx 411 \text{ (veh/hr)}$

Approach 3: $V_3 = 1.028 \times 800 = 822.4 \approx 822 \text{ (veh/hr)}$

Approach 4: $V_4 = 1.028 \times 650 = 668.2 \approx 668 \text{ (veh/hr)}$

- Total lost time:

$$L = \sum_{j=1}^{\Phi} l_j + R$$

$$= 2 \times 3 + 2 = 8 \text{ (sec)}$$

where l_j = lost time for phase j ; R = total all red during a cycle.

- Determine the maximum value of the ratio of approach volumes to saturation flows for phase j , Y_j (observed volume/saturation flow, for the critical approach in each phase).

Phase A:

	Approach 1	Approach 3
Volume/hr: V_i	1028	822
Saturation flow: S_i	2000	2000
V_i / S_i	0.514	0.411
Y_1	$\max(V_i / S_i) = 0.514$	

Phase B:

	Approach 2	Approach 4
Volume/hr: V_i	411	668
Saturation flow: S_i	2000	2000
V_i / S_i	0.206	0.334
Y_2	$\max(V_i / S_i) = 0.334$	

$$\sum_{j=1}^{\Phi} Y_j = 0.514 + 0.344 = 0.848$$

- Determine optimal cycle length C_0 (sec).

$$C_0 = (1.5L + 5) / (1 - \sum_{j=1}^{\Phi} Y_j) = (1.5 \times 8 + 5) / (1 - 0.848) = 17 / 0.152$$

$$= 111.8 \text{ (sec)}$$

- Find the total effective green time g_T .

$$g_T = C_0 - L = 111.8 - 8 = 103.8 \text{ (sec)}$$

- Total effective green times for phase A (g_1) and B (g_2):

$$g_1 = g_T Y_1 / \sum_{j=1}^{\Phi} Y_j = 103.8 \times 0.514 / 0.848 = 62.9 \approx 60 \text{ (sec)}$$

$$g_2 = g_T Y_2 / \sum_{j=1}^{\Phi} Y_j = 103.8 \times 0.334 / 0.848 = 40.9 \approx 40 \text{ (sec)}$$

- Total green time for each phase G_{1w} and G_{2w} :

$$G_{1w} = g_1 + l_1 - \tau_1 = 60 + 3 - 3 = 60 \text{ (sec)}$$

$$G_{2w} = g_2 + l_2 - \tau_2 = 40 + 3 - 3 = 40 \text{ (sec)}$$

- Cycle length:

$$C = G_{1w} + \text{yellow}_1 + \text{all_red}_1 + G_{2w} + \text{yellow}_2 + \text{all_red}_2$$

$$= 60 + 3 + 1 + 40 + 3 + 1$$

$$= 108 \text{ (sec)}$$

5.3 Simulation of normal operation

According to the traffic data and signal control parameters determined previously, 20 cycles were simulated. The queue lengths on each approach are listed in Table 5.1.

Table 5.1 Queue lengths (veh) under normal operation

Cycle #	Approach 1	Approach 2	Approach 3	Approach 4
1	10	9	9	12
2	14	7	14	18
3	16	8	10	11
4	13	8	13	13
5	17	6	8	7
6	23	6	15	13
7	23	9	12	14
8	28	7	5	11
9	26	9	13	15
10	23	9	11	13
11	14	5	10	15
12	15	4	12	11
13	18	7	15	14
14	16	11	12	15
15	18	6	15	12
16	19	8	11	12
17	8	6	11	18
18	19	7	16	16
19	28	8	13	15
20	28	5	11	16
mean	18.8	7.25	11.8	13.55
std	5.9259	1.713	2.6675	2.6253
mean+2std	30.65	10.68	17.14	18.80

- Notes: 1. queue lengths are measured at the end of each red interval.
 2. mean = mean of queue length.
 3. std = standard deviation of queue length.

5.4 Simulation of control strategies for transition 1

5.4.1 Given conditions

- EV operating speed $S_{EV} = 15$ (m/s or 54 Km/h)
- General vehicle mean speed $S_{mean} = 12$ (m/s or 43.2 Km/h)
- Discharge speed $S_{dis} = 4.44$ (m/s or 16 Km/h)
- Minimum greens for main road $g_{1min} = 7$ sec, and for side road $g_{2min} = 10$ sec
- Maximum greens for main road $g_{1max} = 100$ sec, and for side road $g_{2max} = 100$ sec
- Startup delay $t_{startup} = 2$ sec
- Safety time interval $STI = 3$ sec
- Average spacing between successive vehicles in a queue, $S_{spacing} = 8$ metre

5.4.2 Determination of EV detection distance

When determining the minimum distance of EV detection away from the intersection, we use (average queue length + 2 times standard deviation) as the queue length to ensure in almost all traffic situations EV can pass through the intersection. The queue lengths for each approach have been obtained in previous section. For each scenario discussed in Chapter 3, we have deduced the minimum distance L_{min} from the following information.

$$L_0 \geq (STI + (L_{QS} + L_{QM}) / S_{dis}) * S_{EV} \quad (3.6)$$

$$L_1 \geq (STI + t_{startup} + (L_{QS} + L_{QM}) / S_{dis}) * S_{EV} \quad (3.9)$$

$$L_2 \geq (STI + t_{startup} + (L_{QS} + L_{QM}) / S_{dis} + T_{red-2} - t'_{red}) * S_{EV} \quad (3.11)$$

$$L_3 \geq (STI + t_{startup} + (L_{QS} + L_{QM}) / S_{dis} + T'_{yellow} + T_{red-2}) * S_{EV} \quad (3.13)$$

$$L_4 \geq (STI + t_{startup} + (L_{QS} + L_{QM}) / S_{dis} + T'_{yellow} - t'_{yellow} + T_{red-2}) * S_{EV} \quad (3.15)$$

Finally, the minimum distance should be maximum(L_0, L_1, L_2, L_3, L_4). Based on the simulation results of normal operation, we find the EV detection distances should be at least 962.6, 423.4, 597.8, and 642.7 metres away from the intersection on approach 1, 2, 3, and 4, respectively.

We will use the distances of 960, 420, 600, and 640 metres on corresponding approaches in the following simulations and analysis. In order to assess the effectiveness and efficiency of the real-time control strategy for situations when an EV requests preemption from a mid-block and the minimum distance requirement of EV detection cannot be met, another set of simulations will also be conducted with EV distances of 480, 210, 300, and 320 metres on approaches 1, 2, 3, and 4 from the intersection, respectively.

5.4.3 Situations for EV from main road

Usually an EV travels on a main road; however, occasionally it also approaches an

intersection from side road. We treat them in different simulations because of their differences in traffic densities. First, we deal with situations in which EV enters the system from approach 1. Then in section 5.4.4, we will simulate situations in which EV enters the system from side road (approach 2).

5.4.3.1 Simulation scenarios

EVSP scenarios are listed in Table 5.2. The EV enters the system from approach 1, and EV entry time (sec) is measured from the beginning of green(1) interval (i.e., green interval for approach 1 with the EV) during 5th to 10th cycle period for normal operation, respectively.

Table 5.2 EVSP scenarios for approach 1

scenario #	EV entry time (sec)	Remarks
0-base case	N/A	no EVSP
1	$t = 0$	beginning of G_{1w}
2	$t = \text{Green}(1)/2$	half G_{1w}
3	$t = \text{Green}(1)$	beginning of yellow
4	$t = \text{Green}(1) + \text{Yellow}(1)$	beginning of red
5	$t = \text{Green}(1) + \text{Yellow}(1) + \text{All_red}(1)$	beginning of G_{2w}
6	$t = \text{Green}(1) + \text{Yellow}(1) + \text{All_red}(1) + \text{Green}(2)/2$	half G_{2w}

5.4.3.2 Simulation of fixed held time control

In order to compare the effectiveness of our real-time control strategy with those of existing systems, the fixed held time algorithm for transition 1 is selected as reference in this research. The held time for the fixed time transitioning on approach 1 is fixed at $t_{11} = 50$ and $t_{12} = 30$ seconds, separately. Tables 5.3 and 5.4 summarize simulation results for $t_{11} = 50$ and $t_{12} = 30$ seconds, respectively.

In the tables, "more time needed" is additional time over the held time needed to clear the queue ahead of EV; "spare time" is the remaining time of the held time after the queue ahead of EV is cleared; and "queue" is undischarged queue on approach 1 using the fixed held time when failure occurs; otherwise "queue" is the queue formed at the end of the fixed held time.

More time needed > 0 means that the fixed held time is not long enough to clear the queue ahead of the EV, and the EV has to stop or reduce operating speed. Therefore the signal control strategy fails to serve the EVSP operation. Spare time is the green time duration between the instant the queue ahead of the EV clears the intersection and the moment the EV clears the same intersection. The longer the spare time the less efficient the signal control system.

Table 5.3 Simulation results for 50-second fixed held time on approach 1

Scenario #	Cycle #	More time needed (sec)	Spare time (sec)	Queue (veh) on Approach			
				1	2	3	4
1	5		26.6	15	5	0	7
	6		18.32	15	5	0	12
	7		8.6	16	6	1	12
	8		7.52	17	6	1	7
	9	2.2	-	2	7	0	12
	10		1.4	11	7	0	9
2	5		39.2	14	8	0	12
	6		31.28	16	7	1	18
	7		25.88	17	11	1	17
	8		20.12	18	11	0	15
	9		22.28	18	12	0	19
	10		22.28	14	13	0	16
3	5		50	18	12	0	19
	6		46.4	22	12	0	24
	7		35.24	17	14	0	21
	8		29.48	18	17	0	22
	9		30.92	17	17	0	25
	10		37.4	17	14	0	22
4	5		48.2	18	12	1	20
	6		44.6	21	12	1	25
	7		33.44	16	14	0	21
	8		29.48	18	17	0	22
	9		29.12	17	17	1	25
	10		35.6	16	14	0	22
5	5		48.2	19	14	1	21
	6		42.8	20	13	1	21
	7		33.44	17	12	0	26
	8		29.48	19	15	0	22
	9		27.32	17	17	1	22
	10		33.8	16	18	0	26
6	5		36.32	19	6	0	12
	6		28.4	17	5	0	13
	7		21.92	17	6	1	20
	8		16.72	12	6	0	13
	9		16.52	13	8	1	18
	10		26.6	16	9	1	17

Table 5.4 Simulation results for 30-second fixed held time on approach 1

Scenario #	Cycle #	More time needed (sec)	Spare time (sec)	Queue (veh) on Approach			
				1	2	3	4
1	5		6.6	10	4	2	4
	6	1.68		1	4	1	7
	7	11.4		7	4	6	10
	8	12.48		7	5	4	3
	9	22.2		12	4	1	9
	10	18.6		10	5	1	5
2	5		19.2	8	6	0	7
	6		11.28	8	6	0	13
	7		5.88	11	9	0	14
	8		0.12	11	7	0	10
	9		2.28	12	9	0	15
	10		2.28	9	9	0	12
3	5		30	10	10	1	14
	6		26.4	13	9	1	20
	7		15.24	9	12	1	18
	8		9.48	11	12	0	16
	9		10.92	11	14	1	21
	10		17.4	9	13	0	18
4	5		28.2	10	11	0	15
	6		24.6	13	10	1	21
	7		13.44	9	12	1	18
	8		9.48	12	13	0	17
	9		9.14	11	14	1	22
	10		15.6	9	13	1	18
5	5		28.2	11	11	0	17
	6		22.8	13	11	1	15
	7		13.44	10	10	1	21
	8		9.48	13	12	0	18
	9		7.32	10	13	1	17
	10		13.8	9	14	1	22
6	5		16.32	13	4	1	10
	6		8.4	12	4	1	10
	7		1.92	11	4	0	17
	8	5.28		3	4	0	11
	9	3.48		2	6	1	11
	10		6.6	12	7	0	15

5.4.3.3 Simulation of real-time control

Fifty nine seconds were used for our real-time control transitioning and the simulation results are tabulated in Table 5.5.

In the table, "green time for side road" is the portion of the transition period allocated as green for side road; "more time needed" is additional time over the transition period needed to clear the queue ahead of the EV; "spare time" is the portion of green time for approach 1 between the moment the queue ahead of the EV clears the intersection and the instant EV clears the intersection; and "queue" is undischarged queue on approach 1 using the transition period when failure occurs; otherwise "queue" is the queue formed at the end of the transition period.

The first objective of the real-time control strategy is to ensure that the EV can cross the intersection without stopping or reducing speed. "More time needed" is used to measure whether this objective is met or not. More time needed > 0 means that the control strategy fails to serve the EV successfully. Another objective of the real-time control strategy is to decrease the impact of the EVSP on general traffic. A longer "green time for side road" means a less impact of the EVSP on traffic on the side road. The longer the "spare time" and the "queue" the less efficient the real-time control strategy.

Table 5.5 Simulation results for 59-second real-time control on approach 1

Scenario #	Cycle #	Green time for side road (sec)	More Time needed (sec)	Spare time (sec)	Queue (veh) on approach			
					1	2	3	4
1	5	31.6		0	18	2	7	3
	6	23.32		0	18	2	0	7
	7	13.6		0	17	6	0	7
	8	12.52		0	20	4	0	9
	9	0		6.8	15	8	0	15
	10	0		10.4	14	8	0	12
2	5	40.2		0	17	3	6	3
	6	32.28		0	20	3	2	6
	7	26.88		0	19	4	0	4
	8	21.12		0	22	5	0	7
	9	23.28		0	22	6	0	8
	10	23.28		0	17	5	0	7
3	5	51		0	22	1	11	1
	6	44		3.4	24	2	12	2
	7	36.24		0	19	3	3	2
	8	30.48		0	19	5	0	6
	9	31.92		0	19	4	0	8
	10	38.4		0	19	0	4	4
4	5	47		5.2	22	1	7	3
	6	47		1.6	23	1	14	3
	7	37.44		0	19	3	3	2
	8	33.48		0	20	5	0	6
	9	33.12		0	18	4	0	9
	10	39.6		0	18	0	3	5

(continued on next page)

5	5	48		5.2	22	2	7	1
	6	47.8		0	22	2	13	3
	7	38.44		0	19	2	4	7
	8	34.48		0	20	3	0	4
	9	32.32		0	17	6	0	7
	10	38.8		0	18	3	2	6
6	5	41.32		0	22	1	5	3
	6	33.4		0	21	2	6	5
	7	26.92		0	21	4	0	7
	8	19.72		0	16	5	0	5
	9	21.52		0	15	5	0	9
	10	31.6		0	19	4	0	5

If an EV starts from a mid-block and its distance from a nearby intersection is shorter than the required minimum distance of EV detection, it is possible that the EVSP system has not enough time to discharge the queue at the intersection before the EV reaches the intersection. In order to test the behaviour of the real-time control strategy in such a situation, another simulation was conducted. The EV requested preemption at 420 metres from the intersection and thirty two seconds were used for the real-time control transitioning. The simulation results are listed in Table 5.6.

Table 5.6 Simulation results for 32-second real-time control on approach 1 (The requirement of minimum EV detection distance was not met)

Scenario #	Cycle #	Green time for side road (sec)	More Time needed (sec)	Spare time (sec)	Queue (veh) on approach			
					1	2	3	4
1	5	0		3.6	9	4	4	0
	6	0	4.68		3	3	7	0
	7	0	14.4		8	3	9	0
	8	0	15.48		9	5	3	0
	9	0	25.2		14	4	9	0
	10	0	21.6		12	4	5	0
2	5	0		12.2	8	6	7	0
	6	0		4.3	8	5	13	0
	7	0		1.1	10	8	14	0
	8	0	5.38		3	7	10	0
	9	0	3.22		2	8	14	0
	10	0	3.22		2	8	11	0
3	5	19.0		0	8	2	4	4
	6	12.0		3.4	11	3	13	0
	7	0		8.2	8	12	17	0
	8	0		2.5	10	12	16	0
	9	0		3.9	9	14	21	0
	10	0		10.4	8	13	18	0
4	5	15.0		5.2	9	3	6	1
	6	15.0		1.6	12	2	12	1
	7	0		9.4	8	12	18	0
	8	0		5.5	11	12	16	0
	9	0		5.1	10	14	21	0
	10	0		11.6	8	13	18	0

(continued on next page)

5	5	16.0		5.2	9	2	8	1
	6	15.8		0	11	2	5	1
	7	0		10.4	8	9	20	0
	8	0		6.5	11	12	18	0
	9	0		4.3	9	12	16	0
	10	0		10.8	7	14	21	0
6	5	0		13.3	11	3	10	0
	6	0		5.4	12	3	8	0
	7	0	1.08		1	4	15	0
	8	0	8.28		5	3	10	0
	9	0	6.48		4	6	11	0
	10	0		3.6	11	6	14	0

From Table 5.6, we can see that the real-time control algorithm failed eleven times out of the total 36 cases. In such a situation, other methods have to be developed to provide the information on the approaching of an EV at an intersection in advance so that the real-time control strategy can clear the way for the EV. This task can only be accomplished at the network level instead of the intersection level. Consequently, it will be a topic for future research and the simulation results for this situation will not be discussed further in the following sections.

5.4.4 Situations for EV from side road

In this section, we will simulate situations in which EV enters the system from side road (approach 2).

5.4.4.1 Simulation scenarios

EVSP scenarios are listed in Table 5.7. The EV enters the system from approach 2, and EV entry time (sec) is measured from the beginning of green(2) interval (i.e., green interval for approach 2 with the EV) during 5th to 10th cycle period for normal operation, respectively.

Table 5.7 EVSP scenarios for approach 2

scenario #	EV entry time (sec)	Remarks
0-base case	N/A	no EVSP
1	$t = 0$	beginning of G_{2w}
2	$t = \text{Green}(2)/2$	half G_{2w}
3	$t = \text{Green}(2)$	beginning of yellow
4	$t = \text{Green}(2) + \text{Yellow}(2)$	beginning of red
5	$t = \text{Green}(2) + \text{Yellow}(2) + \text{All_red}(2)$	beginning of G_{1w}
6	$t = \text{Green}(2) + \text{Yellow}(2) + \text{All_red}(2) + \text{Green}(1)/2$	half G_{1w}

5.4.4.2 Simulation of fixed held time control

The held time for the fixed time transitioning on approach 2 is fixed at $t_{f1} = 30$ and $t_{f2} = 20$ seconds, separately. Tables 5.8 and 5.9 list simulation results for $t_{f1} = 30$ and $t_{f2} = 20$ seconds, respectively.

In the tables, "more time needed" is additional time over the held time needed to clear the queue ahead of EV; "spare time" is the remaining time of the held time after the queue ahead of EV is cleared; and "queue" is undischarged queue on approach 2 using the fixed held time when failure occurs; otherwise "queue" is the queue formed at the end of the fixed held time.

More time needed > 0 means that the fixed held time is not long enough to clear the queue ahead of the EV, and the EV has to stop or reduce operating speed. Therefore the signal control strategy fails to serve the EVSP operation.

Table 5.8 Simulation results for 30-second fixed held time on approach 2

Scenario #	Cycle #	More time needed (sec)	Spare time (sec)	Queue (veh) on Approach			
				2	3	4	1
1	5		15.6	3	7	1	12
	6		18.51	5	11	0	17
	7		17.8	4	10	5	19
	8		13.1	3	5	2	24
	9		17.4	6	8	1	22
	10		13.1	5	7	6	18
2	5		30	4	10	0	20
	6		30	4	17	1	24
	7		30	4	13	1	26
	8		30	4	7	1	30
	9		28.2	5	14	0	29
	10		26.4	5	11	1	25
3	5		30	4	13	0	26
	6		30	3	20	0	29
	7		30	3	18	1	32
	8		30	5	12	0	31
	9		30	4	16	1	31
	10		30	4	14	0	29
4	5		30	4	13	0	27
	6		27.5	3	21	0	30
	7		30	4	19	1	33
	8		28.2	4	13	0	32
	9		28.2	3	16	0	32
	10		27.5	4	15	0	30
5	5		30	4	18	0	23
	6		27.5	3	14	0	27
	7		30	4	22	0	30
	8		28.2	4	20	0	33
	9		28.2	3	14	0	32
	10		27.5	4	17	0	32
6	5		22.1	2	8	0	14
	6		21.4	2	9	0	18
	7		22.1	5	12	0	24
	8		21	2	11	0	27
	9		22.1	5	5	1	27
	10		19.6	4	5	1	24

Table 5.9 Simulation results for 20-second fixed held time on approach 2

Scenario #	Cycle #	More time needed (sec)	Spare time (sec)	Queue (veh) on Approach			
				2	3	4	1
1	5		5.6	2	5	6	7
	6		8.51	3	7	3	12
	7		7.8	2	6	9	15
	8		3.1	2	4	6	19
	9		7.4	5	4	6	18
	10		3.1	4	5	9	13
2	5		20	2	8	1	16
	6		20	2	14	1	21
	7		20	3	11	3	22
	8		20	1	5	0	27
	9		18.2	3	12	1	25
	10		16.4	3	10	3	22
3	5		20	3	11	0	23
	6		20	3	18	0	26
	7		20	2	15	0	28
	8		20	3	10	0	31
	9		20	4	14	0	29
	10		20	4	13	0	27
4	5		20	4	11	0	23
	6		17.5	2	19	0	28
	7		20	2	16	0	28
	8		18.2	3	11	0	31
	9		18.2	3	14	0	30
	10		17.5	2	13	0	29
5	5		20	4	18	0	22
	6		17.5	2	11	0	24
	7		20	2	19	0	28
	8		18.2	3	16	1	29
	9		18.2	3	11	1	31
	10		17.5	2	14	1	30
6	5		12.1	1	6	0	11
	6		11.4	1	6	2	15
	7		12.1	2	10	1	22
	8		11	1	8	0	23
	9		12.1	3	5	1	22
	10		9.6	2	3	0	20

5.4.4.3 Simulation of Real-time control

Twenty three seconds were used for our real-time control transitioning and Table 5.10 lists simulation results.

In the table, "green time for main road" is the portion of the transition period allocated as green for main road; "more time needed" is additional time over the transition period needed to clear the queue ahead of the EV; "spare time" is the portion of green time for approach 2 between the moment the queue ahead of the EV clears the intersection and the instant the EV clears the intersection; and "queue" is undischarged queue on approach 2 using the transition period when failure occurs; otherwise "queue" is the queue formed at the end of the transition period.

The first objective of the real-time control strategy is to ensure that the EV can cross the intersection without stopping or reducing speed. "More time needed" is used to measure whether this objective is met or not. More time needed > 0 means that the control strategy fails to serve the EV successfully. Another objective of the real-time control strategy is to decrease the impact of the EVSP on general traffic. A longer "green time for main road" means a less impact of the EVSP on traffic on the main road. The longer the "spare time" and the "queue" the less efficient the real-time control strategy.

Table 5.10 Simulation results for 23-second real-time control on approach 2

Scenario #	Cycle #	Green time for main road (sec)	More Time needed (sec)	Spare time (sec)	Queue (veh) on approach			
					2	3	4	1
1	5	0		8.6	3	5	1	8
	6	7.48		0	4	5	6	9
	7	0		10.8	3	7	1	16
	8	0		6.1	3	4	2	21
	9	0		10.4	5	5	0	19
	10	0		6.1	5	6	3	15
2	5	15		0	2	0	10	9
	6	15		0	3	6	7	14
	7	15		0	3	4	15	15
	8	15		0	2	0	10	20
	9	0		21.2	4	13	0	26
	10	0		19.4	4	11	0	23
3	5	15		0	4	3	3	15
	6	15		0	3	10	6	19
	7	15		0	2	8	8	20
	8	15		0	4	2	2	23
	9	15		0	4	5	6	22
	10	15		0	3	4	4	20
4	5	18		0	4	2	3	14
	6	8		7.5	2	14	0	23
	7	18		0	2	6	8	20
	8	8		8.2	3	6	0	26
	9	8		8.2	3	9	0	25
	10	8		7.5	2	8	0	24

(continued on next page)

5	5	19		0	4	8	4	11
	6	9		7.5	2	8	0	19
	7	19		0	2	9	8	17
	8	9		8.2	3	12	0	25
	9	9		8.2	3	7	0	26
	10	9		7.5	2	10	0	25
6	5	9		2.1	2	1	3	8
	6	9		1.4	1	4	6	11
	7	9		2.1	3	5	6	18
	8	9		1	2	5	1	19
	9	9		2.1	3	4	6	19
	10	8.6		0	3	3	3	16

In order to test the behaviour of the real-time control algorithm in the situation when the minimum distance requirement of EV detection is not met, i.e., an EV starts suddenly from a mid-block, another simulation was conducted. The EV requested preemption at 210 metres away from the intersection on approach 2 and twelve seconds were used for the real-time control transitioning. The simulation results are tabulated in Table 5.11.

Table 5.11 Simulation results for 12-second real-time control on approach 2 (The requirement of minimum EV detection distance was not met)

Scenario #	Cycle #	Green time for main road (sec)	More Time needed (sec)	Spare time (sec)	Queue (veh) on approach			
					2	3	4	1
1	5	0	5.40		3	4	21	4
	6	0	2.52		2	4	15	8
	7	0	3.24		2	3	22	13
	8	0	7.92		5	1	22	16
	9	0	5.60		2	3	22	15
	10	0	7.92		5	3	21	10
2	5	0		5.0	1	7	16	11
	6	0		5.0	2	11	7	16
	7	0		5.0	2	10	15	18
	8	0		5.0	1	5	18	23
	9	0		3.20	1	8	19	22
	10	0		1.40	1	7	17	17
3	5	0		5.0	1	10	4	19
	6	0		5.0	1	17	2	24
	7	0		5.0	1	13	10	25
	8	0		5.0	2	7	13	28
	9	0		5.0	2	14	10	28
	10	0		5.0	1	11	9	24
4	5	0		8.0	2	10	4	20
	6	0		5.5	1	17	4	24
	7	0		8.0	1	14	10	27
	8	0		6.2	2	8	14	28
	9	0		6.2	1	14	12	29
	10	0		5.5	1	12	10	25

(continued on next page)

5	5	0		9.0	2	15	4	18
	6	0		6.5	1	10	4	20
	7	0		9.0	1	17	10	24
	8	0		7.2	2	14	14	27
	9	0		7.2	1	8	12	28
	10	0		6.5	1	14	10	29
6	5	0		1.1	1	5	11	8
	6	0		0.4	1	3	12	14
	7	0		1.1	1	8	17	17
	8	0		0	0	5	20	20
	9	0		1.1	1	1	17	19
	10	0	1.44		1	1	16	17

From Table 5.11, we can see that the real-time control algorithm failed seven times out of the total 36 cases. It is suggested again that the real-time control strategy itself could not deal with such a situation and other methods at network level need to be developed and combined with the real-time control strategy at the intersection level to solve this problem. Therefore, this situation will not be discussed further in the following sections.

5.5 Analysis and comparison of simulation results for EV from main road

Table 5.12 shows the failure numbers of control strategies for scenarios in which EV entered the system from main road. While our real-time control strategy could

successfully deal with every scenario, EVSP failed one time and seven times under 50-second and 30-second fixed held time control strategies, respectively. It means that under the two fixed held time control strategies the EV had to stop or reduce speed at the intersection before the queues ahead of it cleared the intersection.

Table 5.12 Failure number of control strategies for EV from approach 1

	Fixed time $t_f=50(s)$	Fixed time $t_f=30(s)$	Real-time $t_r=59(s)$
Failure #	1	7	0

Table 5.13 summarizes average spare times on approach 1 from which the EV entered the system. Because spare time is the green time duration between the instant the queue ahead of the EV clears the intersection and the instant the EV clears the same intersection, the longer the spare time the less efficient the signal control system. By dynamically allotting green time that is not necessary for main road to side road, our real-time control algorithm can minimize the impact of EVSP on side road. We can see from Table 5.11 the real-time control strategy had very small spare time for every scenario. Overall, it reduced the average spare time to 0.91 seconds from 29.57 seconds and 13.3 seconds under 50-second and 30-second fixed held time control strategies, respectively. Another factor worthy to be noted is the standard deviation (std) of average spare time. The smaller std for the real-time control strategy means this strategy performed steadier from cycle to cycle and from scenario to scenario than fixed held time

control strategies.

Table 5.13 Average spare (sec) time on approach 1 for transition 1

Scenario #		Fixed time $t_f=50(s)$	Fixed time $t_f=30(s)$	Real-time $t_r=59(s)$
1	Average	12.49 ⁽¹⁾	6.60 ⁽²⁾	2.87
	std ⁽³⁾	9.95	N/A	4.58
2	Average	26.84	6.84	0.00
	std	7.21	7.21	0.00
3	Average	38.24	18.24	0.57
	std	8.31	8.31	1.39
4	Average	36.74	16.74	1.13
	std	7.95	7.95	2.09
5	Average	35.84	15.84	0.87
	std	8.05	8.05	2.12
6	Average	24.41	8.31	0.00
	std	7.62	6.00	0.00
Over all scenarios	Average	29.57	13.30	0.91
	std	11.62	8.46	2.35

Notes: (1) Excluding 1 case in which control strategy failed.

(2) Excluding 7 cases in which control strategy failed.

(3) std=standard deviation.

Table 5.14 summarizes average green time for side road during transition 1 under only the real-time control strategy because the fixed held time control strategy always gives green to main road during the fixed held time transitioning. Average green time for side road ranged from 13.51 seconds for scenario 1 to 39.97 seconds for scenario 5. Not surprisingly, the shortest average green time for side road was for scenario 1 and the longest one was for scenario 5. In scenario 1 the EV entered the system at the beginning of green interval for main road, and there existed the longest average queue length which

needed the longest time to clear. Therefore, the shortest green time could be allotted to side road during the transition period. On the other hand, when the EV entered the system at the end of all red interval for main road in scenario 5, most or all of queue ahead of the EV had cleared the intersection, and signal indication could be changed to green for side road immediately. So the shortest or no additional green time was needed for main road. As a result, almost all the transition duration, minus switchover time, startup delay and safety time interval (STI), could be allotted to side road as green interval. For all scenarios, an average of 31.45 seconds was used as green time for side road during transition 1 under our real-time control strategy.

Table 5.14 Average green time (sec) for side road during transition 1

Scenario #		Fixed time $t_f=50(s)$	Fixed time $t_f=30(s)$	Real-time $t_{rt}=59(s)$
1	Average	N/A	N/A	13.51
	std			12.57
2	Average			27.84
	std			7.21
3	Average			38.67
	std			7.73
4	Average			39.61
	std			6.22
5	Average			39.97
	std			6.61
6	Average			29.08
	std			8.05
Over all scenarios	Average			31.45
	std			12.28

Summarized in Table 5.15 are average queue lengths after transition 1 for cases when the EV entered the system from main road. As expected, our real-time control strategy produced longer average queue on main road (19.25 vehicles on approach 1 and 3.03 vehicles on approach 3) and shorter average queue on side road (3.44 vehicles on approach 2 and 5.58 vehicles on approach 4), compared with average queue lengths of 16.80 vehicles (approach 1), 0.36 vehicles (approach 2), 11.08 vehicles (approach 3), and 18.19 (approach 4) under 50-second fixed held time control strategy and average queue lengths of 10.69 vehicles (approach 1), 0.83 vehicles (approach 2), 8.75 vehicles (approach 3), and 14.19 vehicles (approach 1) under 30-second fixed held time control strategy, respectively. The comparison in diagram form is shown in Figures 5.4 and 5.5. As Figure 5.5 indicates, under the real-time control, our algorithm was successful in reducing average queue over all scenarios and approaches to 7.83 vehicles from 11.61 and 8.62 vehicles under 50-second and 30-second fixed held time control strategies, respectively, because of the more efficient allotment of green time between main road and side road.

Table 5.15 Average queue lengths (veh) after transition 1 for EV from approach 1

Approach #	Scenario #		Fixed time $t_f=50(s)$	Fixed time $t_f=30(s)$	Real-time $t_r=59(s)$
1	1	Average	14.80 ⁽¹⁾	10.00 ⁽²⁾	17.00
		std	2.28	N/A	2.19
	2	Average	16.17	9.83	19.50
		std	1.83	1.72	2.26
	3	Average	18.17	10.50	20.33
		std	1.94	1.52	2.16
	4	Average	17.67	10.67	20.00
		std	1.86	1.63	2.10
	5	Average	18.00	11.00	19.67
		std	1.55	1.67	2.07
	6	Average	15.67	12.00	19.00
		std	2.66	0.82 ⁽³⁾	2.90
	over all scenarios	Average	16.80	10.69	19.25
		std	2.27	1.56	2.39
2	1	Average	6.00	4.33	5.00
		std	0.89	0.52	2.76
	2	Average	10.33	7.67	4.33
		std	2.34	1.51	1.21
	3	Average	14.33	11.67	2.50
		std	2.25	1.86	1.87
	4	Average	14.33	12.17	2.33
		std	2.25	1.47	1.97
	5	Average	14.83	11.83	3.00
		std	2.32	1.47	1.55
	6	Average	6.67	4.83	3.50
		std	1.51	1.33	1.64
	over all scenarios	Average	11.08	8.75	3.44
		std	4.16	3.60	2.01

(continues on next page)

3	1	Average	0.33	2.50	1.17
		std	0.52	2.07	2.86
	2	Average	0.33	0.00	1.33
		std	0.52	0.00	2.42
	3	Average	0.00	0.67	5.00
		std	0.00	0.52	5.29
	4	Average	0.50	0.67	4.50
		std	0.55	0.52	5.32
	5	Average	0.50	0.67	4.33
		std	0.55	0.52	5.01
	6	Average	0.50	0.50	1.83
		std	0.55	0.55	2.86
over all scenarios	Average	0.36	0.83	3.03	
	std	0.49	1.18	4.18	
4	1	Average	9.83	6.33	8.83
		std	2.48	2.80	4.22
	2	Average	16.17	11.83	5.83
		std	2.48	2.93	1.94
	3	Average	22.17	17.83	3.83
		std	2.14	2.56	2.71
	4	Average	22.50	18.50	4.67
		std	2.07	2.59	2.58
	5	Average	23.00	18.33	4.67
		std	2.37	2.66	2.42
	6	Average	15.50	12.33	5.67
		std	3.27	2.94	2.07
over all scenarios	Average	18.19	14.19	5.58	
	std	5.40	5.20	3.03	
Average queue length over all scenarios and approaches			11.61	8.62	7.83

Notes: (1) Excluding 1 case in which control strategy failed.

(2) Excluding 5 cases in which control strategy failed.

(3) Excluding 2 cases in which control strategy failed.

Fig. 5.4 Average queue for EV from approach 1 after transition 1

Fig. 5.5 Overall average queue for EV from approach 1 after transition 1

In order to convert average queue lengths over all scenarios into commonly used effective measurement, delay time, we use average discharge speed, $S_{dis} = 4.44$ m/s (or 16 km/h), and average spacing between successive vehicles in a queue, $S_{spacing} = 8$ m.

After transferring the queue lengths into equivalent delay times, we can see from Table 5.16 that all three transitioning algorithms kept overall average delay time below that of the base case, while our real-time control strategy produced least average delay time, i.e., only 61% of delay time under normal signal operation. Looking at it in detail, the two fixed held time control strategies incurred delay times on approaches 2 and 4 over those of base cases by 53% and 34% under 50-second fixed held time control strategy and by 21% and 5% under 30-second fixed held time control strategy. At the same time, only 2% average delay time over that of base case on approach 1 appeared under the real-time control strategy, and on all the other approaches our strategy produced less than 50% of average delay times incurred under normal signal operation. Figures 5.6 and 5.7 show the comparison in diagram form.

Table 5.16 Average delay time (sec) over all scenarios during transition 1 for EV from approach 1

		Base case	Fixed time $t_f=50(s)$	Fixed time $t_f=30(s)$	Real-time $t_r=59(s)$
Approach 1	Average(s)	33.87	30.27	19.26	34.68
	EVSP/Base case		0.89	0.57	1.02
Approach 2	Average(s)	13.06	19.97	15.77	6.21
	EVSP/Base case		1.53	1.21	0.48
Approach 3	Average(s)	21.26	0.65	1.50	5.46
	EVSP/Base case		0.033	0.07	0.26
Approach 4	Average(s)	24.41	32.78	25.58	10.06
	EVSP/Base case		1.34	1.05	0.41
Average on all approaches	Average(s)	23.15	20.92	15.53	14.10
	EVSP/Base case		0.90	0.67	0.61

Notes: (1) Base case=No EVSP.

(2) EVSP/Base case=ratio of average delay time under each control strategy to that of base case.

Fig. 5.6 Average delay time relative to base case for EV from approach 1 after transition 1

Fig. 5.7 Overall average delay time relative to base case for EV from approach 1 after transition 1

5.6 Analysis and comparison of simulation results for EV from side road

Table 5.17 indicates that each signal control strategy was successful in letting the EV pass through the intersection without stopping or reducing operating speed during transition 1. Actually, at most around 14.08 seconds were needed to clear the queue ahead of the EV in the most demanding scenario 1 as we can see from Table 5.18 under fixed held time control strategies.

Table 5.17 Failure number of control strategies for EV from approach 2

	Fixed time $t_f=30(s)$	Fixed time $t_f=20(s)$	Real-time $t_{rt}=23(s)$
Failure #	0	0	0

Table 5.18 Average spare time (sec) on approach 2 for transition 1

Scenario #		Fixed time $t_f=30(s)$	Fixed time $t_f=20(s)$	Real-time $t_{rt}=23(s)$
1	Average	15.92	5.92	7.00
	std ⁽¹⁾	2.38	2.38	3.98
2	Average	29.10	19.10	6.77
	std	1.51	1.51	10.50
3	Average	30.00	20.00	0.00
	std	0.00	0.00	0.00
4	Average	28.57	18.57	5.23
	std	1.15	1.15	4.07
5	Average	28.57	18.57	5.23
	std	1.15	1.15	4.07
6	Average	21.38	11.38	1.45
	std	0.99	0.99	0.85
over all scenarios	Average	25.59	15.59	4.28
	std	5.40	5.40	5.48

Note: (1) std= standard deviation.

Similar to observation in the last section, our real-time control strategy produced the least overall average spare time among the three control strategies (see Table 5.18). However, the overall average spare time (4.28 seconds) is relatively large compared with the spare time (0.91 seconds in Table 5.13) incurred when the EV entered the system from main road (approach 1), in which the average transition period was longer at 59 seconds. The reason behind it probably resulted from the selected short transition period (23 seconds). Since the yellow (amber) intervals for main road and side road are 4 seconds and the minimal green time for main road is 7 seconds, only if the queue on side road can be cleared in 8 seconds there is opportunity for main road to get green time during the transition period. If the transition period is selected below 15 seconds, few benefits will be obtained from the real-time control strategy. The longer the transition period, the more benefit can be achieved. Usually, this situation will happen at an intersection with heavy traffic on both crossing roads and there exists less than average queue length ahead when the EV enters the system.

Table 5.19 shows average green time for main road during transition 1 while the EV entered the system from the side road. The green times for main road under the real-time control strategy ranged from 1.25 seconds for scenario 1 to 12.33 seconds for scenario 5. Similar to the situation in which the EV entered from main road, when the EV entered from side road the shortest average green time for main road appeared in scenario 1 and the longest average green time appeared in scenario 5. For all the six scenarios, our

real-time control strategy extracted 9.81 seconds in green time for main road during transition 1.

Table 5.19 Average green time (sec) for main road during transition 1

Scenario #		Fixed time $t_f=30(s)$	Fixed time $t_f=20(s)$	Real-time $t_R=23(s)$
1	Average	N/A	N/A	1.25
	std			3.05
2	Average			10.00
	std			7.75
3	Average			15.00
	std			0.00
4	Average			11.33
	std			5.16
5	Average			12.33
	std			5.16
6	Average			8.93
	std			0.16
over all scenarios	Average			9.81
	std			6.03

Average queue lengths at the end of transition 1 are shown in Table 5.20. While keeping the queue lengths on side road at low levels (2.97 vehicles on approach 2 and 3.67 vehicles on approach 4), our real-time control strategy reduced queue lengths to 18.33 vehicles from 26.08 vehicles on approach 1, 6.08 vehicles from 12.92 vehicles on approach 3 under 30-second fixed held time control strategy. In the case of under 20-second fixed held time control strategy, queue lengths were reduced from 22.97 vehicles on approach 1 and 10.64 vehicles on approach 3. The comparison in diagram form is shown in Figure 5.8.

Table 5.20 Average queue lengths (veh) after transition 1 for EV from approach 2

Approach #	Scenario #		Fixed time $t_f=30(s)$	Fixed time $t_f=20(s)$	Real-time $t_r=23(s)$
2	1	Average	4.33	3.00	3.83
		std	1.21	1.26	0.98
	2	Average	4.33	2.33	3.00
		std	0.52	0.82	0.89
	3	Average	3.83	3.17	3.33
		std	0.75	0.75	0.82
	4	Average	3.67	2.67	2.67
		std	0.52	0.82	0.82
	5	Average	3.67	2.67	2.67
		std	0.52	0.82	0.82
	6	Average	3.33	1.67	2.33
		std	1.51	0.82	0.82
	over all scenarios	Average	3.86	2.58	2.97
		std	0.93	0.97	0.94
3	1	Average	8.00	5.17	5.33
		std	2.19	1.17	1.03
	2	Average	12.00	10.00	5.67
		std	3.46	3.16	5.47
	3	Average	15.50	13.50	5.33
		std	3.08	2.88	3.08
	4	Average	16.17	14.00	7.50
		std	3.25	3.10	3.99
	5	Average	17.50	14.83	9.00
		std	3.21	3.43	1.79
	6	Average	8.33	6.33	3.67
		std	2.94	2.42	1.51
	over all scenarios	Average	12.92	10.64	6.08
		std	4.74	4.63	3.44

(continued on next page)

4	1	Average	2.50	6.50	2.17
		std	2.43	2.26	2.14
	2	Average	0.67	1.50	7.00
		std	0.52	1.22	6.00
	3	Average	0.33	0.00	4.83
		std	0.52	0.00	2.23
	4	Average	0.17	0.00	1.83
		std	0.41	0.00	3.25
	5	Average	0.00	0.50	2.00
		std	0.00	0.55	3.35
	6	Average	0.33	0.67	4.17
		std	0.52	0.82	2.14
	Over all scenarios	Average	0.67	1.53	3.67
		std	1.31	2.54	3.73
1	1	Average	18.67	14.00	14.67
		std	4.18	4.38	5.24
	2	Average	25.67	22.17	17.83
		std	3.61	3.76	6.31
	3	Average	29.67	27.33	19.83
		std	2.16	2.73	2.79
	4	Average	30.67	28.17	22.00
		std	2.16	2.79	4.43
	5	Average	29.50	27.33	20.50
		std	3.83	3.56	5.92
	6	Average	22.33	18.83	15.17
		std	5.24	4.79	4.62
	Over all scenarios	Average	26.08	22.97	18.33
		std	5.60	6.33	5.40
Average queue length over all scenarios and approaches			10.88	9.43	7.76

The control algorithm performance is indicated by the average queue length over all scenarios and approaches. The real-time control strategy demonstrated the best control result with 7.76 vehicles, compared with 10.88 vehicles and 9.43 vehicles under 30-second and 20-second fixed held time control strategies, respectively. Figure 5.9 shows the comparison in diagram form.

Fig. 5.8 Average queue for EV from approach 2 after transition 1

Fig. 5.9 Overall average queue for EV from approach 2 after transition 1

In order to convert average queue lengths over all scenarios into delay time, we use average discharge speed, $S_{dis} = 4.44$ m/s (or 16 km/h), and average spacing between successive vehicles in a queue, $S_{spacing} = 8$ m.

Table 5.21 shows the equivalent average delay times under normal operation and EVSP operations. Similarly, all three control algorithms kept overall average delay time below that of base case, and our real-time control strategy achieved best result with 60% of average delay time under normal operation. Under our real-time control strategy, only on approach 1 the average delay time reached 98% of corresponding value under normal operation. On the other hand, 30-second fixed time control strategy produced 9% and 39% average delay times over those of base cases on approaches 1 and 3, respectively. The 20-second fixed held time control strategy also produced 22% average delay time over that of base case on approach 1, although its 20 seconds held time was less than the 23 seconds for real-time transitioning. Figures 5.10 and 5.11 compare average delay time in diagram form.

Table 5.21 Average delay time (sec) over all scenarios during transition 1 for EV from approach 2

		Base case	Fixed time $t_f=30(s)$	Fixed time $t_f=20(s)$	Real-time $t_r=23(s)$
Approach 2	Average(s)	13.06	6.96	4.65	5.36
	EVSP/Base case		0.53	0.36	0.41
Approach 3	Average(s)	21.26	23.27	19.17	10.96
	EVSP/Base case		1.09	0.90	0.52
Approach 4	Average(s)	24.41	1.20	2.75	6.61
	EVSP/Base case		0.05	0.11	0.27
Approach 1	Average(s)	33.87	47.00	41.39	33.03
	EVSP/Base case		1.39	1.22	0.98
Average on all approaches	Average(s)	23.15	19.61	16.99	13.99
	EVSP/Base case		0.85	0.73	0.60

Notes: (1) Base case=No EVSP.

(2) EVSP/Base case=ratio of average delay time under each control strategy to that of base case.

Fig. 5.10 Average delay time relative to base case for EV from approach 2 after transition 1

Fig. 5.11 Overall average delay time relative to base case for EV from approach 2 after transition 1

Chapter 6 Simulation and Comparison of Control Strategies for Transition 2

6.1 Situations for EV from main road

As in the case of transition 1, we first deal with situations in which the EV enters the system from main road (approach 1). Then in the next section we will simulate situations in which the EV enters the system from side road (approach 2).

Queue lengths on individual approaches at the beginning of transition 2 are those formed at the end of fixed held time control strategy with 50 or 20 seconds held time on main road and side road, respectively.

6.1.1 Simulation of smooth control strategy for transition 2

The task of transition 2 is to get back to normal operation with least possible impact on general traffic. In order to compare the effectiveness of our optimal control strategy with that of the smooth control strategy on an obvious and simple base, we choose fixed held time ($t_f = 50$ sec for EV from main road or $t_f = 20$ sec for EV from side road) strategy for transition 1.

The following parameters are used for this transition:

- Normal cycle length = $(60 + 40 + 3 \times 2 + 1 \times 2) = 108$ sec;
- Maximum addition: 20% of cycle length = 21.6 sec; and
- Maximum subtraction: 17% of cycle length = 18.36 sec.

Let t_{fin} be the finishing point of EV operation; t_{more} the time difference between the finishing point and the beginning of green interval of background cycle; and t_{less} the time difference between the finishing point and the end of green interval of background cycle. t_{fin} can fall in green interval or in red interval, as shown in Fig. 6.1 and Fig. 6.2. Accordingly, the algorithm to determine the recovery process is described in Fig. 6.3.

When the EV entered the system from main road (approach 1), simulation results of queue lengths on individual approaches at the end of transition 2 are listed in Table 6.1. Since the objective of the optimal control strategy is to minimize the queue lengths at the intersection during a shortest possible time after the EV has crossed the intersection, recovery time and queue length are used as measures of effectiveness for the simulation.

Fig. 6.1 Situation 1: t_{fin} falls in $green(1)$

Fig. 6.2 Situation 2: t_{fin} falls in $red(1)$

Fig. 6.3 Flow chart for determination of smooth transitioning recovery process

(continued on next page)

"B" (t_{fin} falls in red(1))

Fig. 6.3 Flow chart for determination of smooth transitioning recovery process

(continued on next page)

Fig. 6.3 Flow chart for determination of smooth transitioning recovery process

Table 6.1 Queue lengths for EV from approach 1 under smooth transitioning

Scenario #	Cycle #	Recovery Time	Queue (veh) on approach			
			1	2	3	4
1	5	2	39	0	15	4
	6		43	1	12	0
	7		36	1	5	3
	8		35	1	13	2
	9		30	0	11	1
	10		20	1	10	3
2	5	2	34	0	14	1
	6		33	1	11	4
	7		35	1	5	3
	8		31	1	12	5
	9		30	0	10	4
	10		20	1	9	10
3	5	1	24	0	15	18
	6		22	8	12	22
	7		21	5	5	20
	8		17	9	13	22
	9		17	7	11	21
	10		11	9	10	23
4	5	1	25	0	15	18
	6		23	8	12	22
	7		22	5	5	20
	8		19	9	13	22
	9		17	7	11	21
	10		11	9	10	23
5	5	1	25	2	15	14
	6		22	0	12	18
	7		22	8	5	22
	8		19	5	13	20
	9		17	9	11	22
	10		11	7	10	21
6	5	2	26	0	12	7
	6		29	1	5	6
	7		22	1	13	12
	8		17	1	11	13
	9		12	0	10	11
	10		15	1	12	16

Notes: (1) Recovery time=the number of cycles to get back to normal operation.

(2) Queue = queue length at the end of recovery period.

6.1.2 Simulation of optimal control strategy for transition 2

The following procedure is used in this simulation.

- 1 Select $P=P'=R=I_{4 \times 4}$ (4x4 identity) and $S=I_{2 \times 2}$ (2x2 identity).
- 2 Average green intervals and average queue lengths from normal operation, $g_1 = 60$ (sec), $g_2 = 40$ (sec), $q_1 = 18.8$ (veh), $q_2 = 7.25$ (veh), $q_3 = 11.8$ (veh), $q_4 = 13.55$ (veh).
- 3 If we select yellow interval $Y(1)=Y(2) = 3$ (sec), all red interval $AR(1) = AR(2) = 1$ (sec), and then the cycle length = 108 (sec), saturation discharge rate (veh/sec) for green time $s_{g1}= s_{g2}= s_{g3}= s_{g4}=2000/3600$, and saturation discharge rate (veh/sec) for yellow time $s_{yi}= s_{gi} / 2$ ($i=1,2,3$,and 4), we can get

$$W_u = (g_1, g_2)_{1 \times 2}$$

$$W'_u = [(g_1, 0), (0, g_2), (g_1, 0), (0, g_2)]^T_{4 \times 2}$$

$$W_x = \text{diag} (1/q_1, 1/q_2, 1/q_3, 1/q_4)_{4 \times 4}$$

$$S_g = \text{diag} (s_{g1}(i), s_{g2}(i), s_{g3}(i), s_{g4}(i))_{4 \times 4}$$

$$S_y = \text{diag} (s_{y1}(i), s_{y2}(i), s_{y3}(i), s_{y4}(i))_{4 \times 4}$$

- 4 Decide time step duration, T , and the total number of time step, N (listed in Table 6.2);

Table 6.2 Time parameters of optimal control for EV from approach 1

Scenario #	RNC (sec)	N (sec)	T (sec)	Remarks
1	58	2	29	EV enters at beginning of green(1)
2	28	1	28	EV enters at half green(1)
3	106	4	26.5	EV enters at end of green(1)
4	103	4	25.75	EV enters at end of yellow(1)
5	102	4	25.5	EV enters at beginning of green(2)
6	82	3	27.3	EV enters at half green(2)

Notes: (1) Cycle length, $C=108(\text{sec})$; and Fixed held time, $t_f=50(\text{sec})$ for transition 1.

(2) Additional number of cycles used for recovery, $n=0$.

If queue lengths are still too long to get back to normal operation, another normal cycle length will be used for recovery. This procedure can proceed until recovery is completed.

- 5 Generate vehicle arrivals, $A(i)$, for each time step;
- 6 Produce combination of $\delta_1(i)$ and $\delta_2(i)$ for current M-step sequences;
- 7 Set other $\delta_1(i) = \delta_2(i) = 1$, $g_1(i) = g_1 * (T - y(1) - y(2) - AR(1) - AR(2)) / (g_1 + g_2)$ and $g_2(i) = g_2 * (T - y(1) - y(2) - AR(1) - AR(2)) / (g_1 + g_2)$;
- 8 For each combination of $\delta_1(i)$ and $\delta_2(i)$, solve the linear system of equations for control variables;
- 9 Calculate performance indices, and then select the optimal one to implement the first time step in the M-step sequences;
- 10 Produce combination of $\delta_1(i)$ and $\delta_2(i)$ for next M-step sequences, and repeat steps 7

through 9.

Table 6.3 summarizes simulation results under the optimal control strategy for cases when the EV entered the system from approach 1. Since the objective of the optimal control strategy is to minimize the queue lengths at the intersection during a shortest possible time after the EV has crossed the intersection, recovery time and queue length are used as measures of effectiveness for the simulation. In Table 6.4, a detailed description of the control process for one transition period is provided.

Table 6.3 Queue lengths for EV from approach 1 under optimal control strategy

Scenario #	Cycle #	Recovery time (sec)	Queue (veh) on approach			
			1	2	3	4
1	5	58	14.35	1.18	0	4.2
	6		18.67	1.18	0	12.88
	7		15.92	3.79	0	9.29
	8		18.04	1.17	0	8.32
	9		21	2.27	0	13.33
	10		15.39	2.89	0	7.74
2	5	28	20.48	1.24	0	3.24
	6		25.28	0	4.48	12.56
	7		22.3	5.35	3.74	10.48
	8		23.7	3.35	0	11.48
	9		27.61	2.52	9.23	11
	10		24.82	2.88	6.65	8.98
3	5	106	20.59	0.17	0	9.57
	6		30.62	0.18	3.22	22.58
	7		21.94	2.61	0	15.62
	8		26.47	0.33	0	15.32
	9		22.47	3.99	1.19	20.18
	10		22.63	3.15	0	15.2

(continued on next page)

4	5	103	15.53	1.4	0	9.83
	6		30.15	0	3.53	22.7
	7		21.97	2.27	0	15.26
	8		27.57	0.87	0	15.86
	9		23.23	3.99	0.9	20.08
	10		21.26	2.72	0	15.16
5	5	102	19.3	1.5	0	9.08
	6		28.02	0.91	3.31	18.39
	7		23.7	1.5	0.13	19.2
	8		27.39	0	0	16.63
	9		22.08	4.01	0.73	16.79
	10		23.32	3.68	0	15.51
6	5	82	13.09	2.89	0	8.88
	6		20.24	0	0	14.82
	7		20.7	0.25	0	16.63
	8		16.42	0	0	12.15
	9		21.99	2.7	1.29	17.37
	10		18.64	3.37	0	13.78

Notes: Queue=queue length at the end of recovery period.

Table 6.4 An example of control process for transition 2

		Approach				Remarks
		1	2	3	4	
Initial queue (veh)		16	9	1	17	
During 1 st time step	Vehicle arrivals (veh)	7	5	5	5	Signal indication sequence=seq(3) and switchover time=4 sec
	Green time (sec)	7	16.3	7	16.3	
During 2 nd time step	Vehicle arrivals (veh)	8	4	5	4	Signal indication sequence=seq(7) and switchover time=4 sec
	Green time (sec)	7.4	15.9	7.4	15.9	
During 3 rd time step	Vehicle arrivals (veh)	12	3	7	6	Signal indication sequence=seq(1) and switchover time=0 sec
	Green time (sec)	27.3	0	27.3	0	
Final queue (veh)		19	3	0	14	
Queue discharged (veh)		24	18	18	18	

Notes: 1. Total transition period = 82 seconds.

2. Total number of time steps = 3.

3. Duration for each time step = 27.3 seconds.

4. Switchover time (yellow and all-red intervals) between two successive green intervals = 4 seconds.

6.2 Situations for EV from side road

In this section we will simulate situations in which the EV enters the system from side road (approach 2). Queue lengths on individual approaches at the beginning of transition 2 are those formed at the end of fixed held time control strategy with 20 seconds held time.

Recovery algorithms are unchanged except the EV enters the system from side road (approach 2). Traffic parameters for geometric layout of intersection and operation also remain the same as for the EV from main road unless specified in the following sections.

6.2.1 Simulation of smooth control strategy for transition 2

When the EV entered the system from side road (approach 2), simulation results of queue lengths on each approach at the end of transition 2 are listed in Table 6.5.

Table 6.5 Queue lengths for EV from approach 2 under smooth transitioning

Scenario #	Cycle #	Recovery time	Queue (veh) on approach			
			2	3	4	1
1	5	2	6	0	13	19
	6		9	0	11	16
	7		7	1	11	17
	8		9	1	15	15
	9		9	2	13	10
	10		5	2	15	5

(continued on next page)

2	5	0	2	8	1	16
	6		2	14	1	21
	7		3	11	3	22
	8		1	5	0	27
	9		3	12	1	25
	10		3	10	3	22
3	5	2	6	0	13	20
	6		9	0	11	17
	7		7	1	10	19
	8		9	1	15	17
	9		2	2	12	12
	10		5	2	14	7
4	5	2	6	0	12	22
	6		9	0	11	19
	7		7	1	10	21
	8		9	1	14	19
	9		2	2	11	14
	10		5	2	14	8
5	5	2	6	2	12	10
	6		9	0	11	22
	7		6	0	10	19
	8		8	1	14	21
	9		9	1	11	19
	10		5	2	14	14
6	5	2	6	2	13	18
	6		9	4	11	30
	7		7	7	11	26
	8		10	1	16	28
	9		10	1	12	26
	10		5	2	14	21

Notes: (1) Recovery time=the number of cycles to get back to normal operation.

(2) Queue=queue length at the end of recovery period, or at the end of fixed held time from transition 1 when recovery time=0.

6.2.2 Simulation of optimal control strategy for transition 2

The first step is to determine recovery time step duration, T , and the total number of time step, N , and they are listed in Table 6.6. Then, optimization and simulations are conducted to get signal timing and queue lengths on approaches when the EV enters the system from side road (approach 2). Simulation results are summarized in Table 6.7

Table 6.6 Time parameters of optimal control for EV from approach 2

Scenario #	RNC (sec)	N	T (sec)	Remarks
1	88	3	29.3	EV enters at beginning of green(2)
2	68	3	22.7	EV enters at half of green(2)
3	48	2	24	EV enters at end of green(2)
4	45	2	22.5	EV enters at end of yellow(2)
5	44	2	22	EV enters at beginning of green(1)
6	122	5	24.4	EV enters at half of green(1)

Notes: (1) Cycle length, $C=108(s)$; and fixed held time, $t_f=20(s)$ for transition 1.

(2) Additional number of cycles used for recovery, $n=0$ for scenario 1 through 5, and $n=1$ for scenario 6.

Table 6.7 Queue lengths for EV from approach 2 under optimal control strategy

Scenario #	Cycle #	Recovery time (sec)	Queue (veh) on approach			
			2	3	4	1
1	5	88	0	0	2.5	3.5
	6		0	1.7	7.4	11
	7		2	0	10	13
	8		0	0	8.8	19
	9		1.8	0	9.9	19
	10		0	0	8.3	13

(continued on next page)

2	5	68	1.7	0	2	12
	6		2.1	8.2	7.3	19
	7		3.7	4.4	4.4	21
	8		3.3	0	5.4	23
	9		5.8	2.4	8.8	22
	10		5.5	0.4	6.9	22
3	5	48	4.1	0	2.5	19
	6		5.5	8.6	7.3	23
	7		8.6	2.9	5.2	20
	8		4.1	0	6.3	28
	9		7.2	5	8.5	23
	10		6	4	5.9	23
4	5	45	4.3	0	1.7	20
	6		4.9	10	6.5	25
	7		8.7	5.6	5.2	21
	8		6.1	0	8.3	26
	9		7.2	3.2	8.6	22
	10		6.5	3.1	7.5	24
5	5	44	5.3	2.1	3.6	16
	6		4.3	1.9	6.2	22
	7		8.8	9.1	5.3	21
	8		5.3	0	8.4	24
	9		7.1	0.8	8.5	23
	10		7.6	2.2	8.6	22
6	5	122 (=14+108)	1	0	1.5	5.9
	6		0.51	0	9.8	17
	7		3.7	0	8.4	19
	8		1.2	0	8.7	21
	9		0.37	1.5	16	20
	10		2.7	0	5.6	14

Notes: Queue=queue length at the end of recovery period.

6.3 Analysis and comparison of simulation results for EV from main road

An important index to evaluate the performance of the traffic signal control system for EVSP is the recovery time to get back to the signal normal operation. Another index

of the same importance as last one is queue lengths or delay times caused by EVSP. Generally speaking, an EVSP system has to compromise between the two factors. If two control strategies incur the same impact on general traffic, we take the strategy with shorter recovery time as the better one. From the perspective of queue or delay time caused by EVSP, we will choose from two control algorithms the one with shorter queue or delay time if they have the same recovery time. What will we do if one algorithm can produce shorter queues and furthermore it has shorter recovery time? The answer is that we have to implement it for EVSP. This situation happened under our optimal control strategy for transition 2.

Table 6.8 shows the average recovery times for the smooth control strategy and our optimal control strategy. The optimal control strategy is better than the smooth transitioning in this performance index with 79.83 seconds versus 162 seconds. The smaller standard deviation (std), 31.24, also indicates the optimal control strategy is steadier than the smooth control strategy, whose standard deviation is 59.15.

Table 6.8 Average recovery time for EV from approach 1

		Smooth transitioning	Optimal control transitioning
Recovery time ⁽¹⁾	Average (sec)	162	79.83
	std	59.15	31.24

Note: (1) Average recovery time over all scenarios.

Now let us look into another control performance index, the average queue lengths

after transition 2 when the EV entered the system from approach 1, shown in Table 6.9. On main road, our optimal control strategy produced shorter average queues with 21.86 vehicles and 1.07 vehicles, compared with 23.67 vehicles and 10.78 vehicles on approaches 1 and 3, respectively, under the smooth control strategy. This resulted from more green time allotted for main road with heavier traffic under the optimal control strategy. Meanwhile, the smooth algorithm allotted green time to main road and side road on a proportion basis, based on normal operation, regardless of actual traffic condition at current time.

Somewhat surprisingly, reducing queue lengths on main road did not sacrifice the interest of traffic on side road under the optimal control strategy. Compared with queues incurred by the smooth control strategy, only 58.9% (1.95 vehicles versus 3.31 vehicles) and 103.3% (13.61 vehicles versus 13.17 vehicles) queue lengths were produced by the optimal control algorithm on approaches 2 and 4, respectively. Obviously, the green time diverted to main road from side road came from durations least demanded by side road.

Actually, our optimal control strategy allotted green time to each road only when green time was most needed according to current traffic condition. Because of the random feature of vehicle arrivals at the intersection the optimal control strategy could successfully allot green time to each road in a dynamic mode. From the average queue length over all scenarios and approaches shown in Table 6.9, we can see that the optimal control algorithm reduced average queue by 22.2% at the intersection to 9.62 vehicles

from 12.37 vehicles under the smooth control algorithm.

Figures 6.4 and 6.5 compare the results in diagram form.

Table 6.9 Average queue lengths (veh) after transition 2 for EV from approach 1

Approach #	Scenario #		Smooth transitioning	Real-time transitioning
1	1	Average	33.83	17.23
		std	8.04	2.46
	2	Average	30.50	24.03
		std	5.47	2.48
	3	Average	18.67	24.12
		std	4.68	3.74
	4	Average	19.50	23.29
		std	5.05	5.13
	5	Average	19.33	23.97
		std	4.93	3.29
	6	Average	20.17	18.51
		std	6.62	3.28
	Over all scenarios	Average	23.67	21.86
		std	8.26	4.35
2	1	Average	0.67	2.08
		std	0.52	1.10
	2	Average	0.67	2.56
		std	0.52	1.83
	3	Average	6.33	1.74
		std	3.44	1.71
	4	Average	6.33	1.88
		std	3.44	1.42
	5	Average	5.17	1.93
		std	3.54	1.58
	6	Average	0.67	1.54
		std	0.52	1.61
	Over all scenarios	Average	3.31	1.95
		std	3.55	1.48

(continued on next page)

3	1	Average	11.00	0.00
		std	3.41	0.00
	2	Average	10.17	4.02
		std	3.06	3.65
	3	Average	11.00	0.74
		std	3.41	1.31
	4	Average	11.00	0.74
		std	3.41	1.41
	5	Average	11.00	0.70
		std	3.41	1.31
	6	Average	10.50	0.22
		std	2.88	0.53
	Over all scenarios	Average	10.78	1.07
		std	3.04	2.14
4	1	Average	2.17	9.29
		std	1.47	3.42
	2	Average	4.50	9.62
		std	3.02	3.34
	3	Average	21.00	16.41
		std	1.79	4.52
	4	Average	21.00	16.48
		std	1.79	4.46
	5	Average	19.50	15.93
		std	3.08	3.61
	6	Average	10.83	13.94
		std	3.76	3.12
	Over all scenarios	Average	13.17	13.61
		std	8.27	4.68
Average queue length over all scenarios and approaches			12.37	9.62

Note: queue length is that at the end of recovery period.

Fig. 6.4 Average queue for EV from approach 1 after transition 2

Fig. 6.5 Overall average queue for EV from approach 1 after transition 2

To convert average queue lengths over all scenarios into delay time, we use average discharge speed, $S_{dis} = 4.44$ m/s (or 16 km/h), and average spacing between successive vehicles in a queue, $S_{spacing} = 8$ m.

From Table 6.10, in terms of delay time the optimal control strategy saved 3.26 seconds (i.e., $42.64 - 39.38 = 3.26$ sec), 2.44 seconds (i.e., $5.96 - 3.52 = 2.44$ sec), 17.5 seconds (i.e., $19.42 - 1.92 = 17.5$ sec), and -0.81 seconds (i.e., $23.72 - 24.53 = -0.81$ sec) on approaches 1, 2, 3, and 4, respectively. Table 6.10 also shows that 22.39 seconds (i.e., $4 \times 22.94 - 4 \times 17.34 = 22.39$ sec) saving in average delay time during each period of transition 2 was achieved by the optimal control strategy. Comparison diagrams are shown in Figures 6.6 and 6.7.

Table 6.10 Average delay time (sec) over all scenarios for EV from approach 1

		Base case	Smooth transitioning	Optimal control transitioning
Approach 1	Average(s)	33.87	42.64	39.38
	EVSP/Base case		1.26	1.16
Approach 2	Average(s)	13.06	5.96	3.52
	EVSP/Base case		0.46	0.27
Approach 3	Average(s)	21.26	19.42	1.92
	EVSP/Base case		0.91	0.09
Approach 4	Average(s)	24.41	23.72	24.53
	EVSP/Base case		0.97	1.00
Average on all approaches	Average(s)	23.15	22.94	17.34
	EVSP/Base case		0.99	0.75

Notes: (1) Base case=No EVSP.

(2) EVSP/Base case=ratio of average delay time under each control strategy to that of base case.

Fig. 6.6 Average delay time relative to base case for EV from approach 1 after transition 2

Fig. 6.7 Overall average delay time relative to base case for EV from approach 1 after transition 2

6.4 Analysis and comparison of simulation results for EV from side road

When the EV entered from side road (approach 2), the average recovery time for transition 2 was 69.17 seconds under the optimal control algorithm, versus 180 seconds under the smooth transitioning, as shown in Table 6.11.

Table 6.11 Average recovery time for EV from approach 2

		Smooth transitioning	Optimal control transitioning
Recovery time ⁽¹⁾	Average (sec)	180	69.17
	std	88.18	31.03

Note: (1) Average recovery time over all scenarios.

In terms of average queue lengths produced after transition 2, the optimal control strategy succeeded again. As shown in Table 6.12, the overall average length was reduced by 15% ($8.1/9.53=85\%$) at the intersection for each occurrence of EVSP under the optimal control strategy, compared with the smooth control strategy. Our control strategy benefited all approaches except approach 1. The average queue lengths were about 65% (i.e., $4.08/6.25=65.28\%$), 76% (i.e., $2.14/2.81=76.16\%$), and 64% (i.e., $6.83/10.64=64.2\%$) on approaches 2, 3, and 4, compared with those caused by the smooth control strategy. Only on approach 1 the optimal control algorithm incurred longer queue over that of the smooth control strategy by 5% (i.e., $19.34/18.44=105\%$). This light penalty was imposed for the total interest of the whole intersection. The comparisons in

diagram form are shown in Figures 6.8 and 6.9.

Table 6.12 Average queue lengths (veh) after transition 2 for EV from approach 2

Approach #	Scenario #		Smooth transitioning	Optimal control transitioning
2	1	Average	7.50	0.63
		std	1.76	0.98
	2	Average	2.33	3.68
		std	0.82	1.70
	3	Average	6.33	5.92
		std	2.66	1.77
	4	Average	6.33	6.28
		std	2.66	1.59
	5	Average	7.17	6.40
		std	1.72	1.70
	6	Average	7.83	1.58
		std	2.14	1.33
	Over all scenarios	Average	6.25	4.08
		std	2.67	2.74
3	1	Average	1.00	0.28
		std	0.89	0.69
	2	Average	10.00	2.57
		std	3.16	3.26
	3	Average	1.00	3.42
		std	0.89	3.27
	4	Average	1.00	3.65
		std	0.89	3.78
	5	Average	1.00	2.68
		std	0.89	3.26
	6	Average	2.83	0.25
		std	2.32	0.61
	Over all scenarios	Average	2.81	2.14
		std	3.71	2.94

(continued on next page)

4	1	Average	13.00	7.82
		std	1.79	2.78
	2	Average	1.50	5.80
		std	1.22	2.41
	3	Average	12.50	5.95
		std	1.87	2.05
	4	Average	12.00	6.30
		std	1.67	2.57
	5	Average	12.00	6.77
		std	1.67	2.08
1	6	Average	12.83	8.33
		std	1.94	4.80
	Over all scenarios	Average	10.64	6.83
		std	4.45	2.88
	1	Average	13.67	13.08
		std	5.20	5.77
	2	Average	22.17	19.83
		std	3.76	4.07
	3	Average	15.33	22.67
		std	4.93	3.14
	4	Average	17.17	23.00
		std	5.27	2.37
	5	Average	17.50	21.33
		std	4.59	2.80
	6	Average	24.83	16.15
		std	4.49	5.60
	Over all scenarios	Average	18.44	19.34
		std	5.88	5.32
	Average queue length over all scenarios and approaches		9.53	8.10

Note: queue length is that at the end of recovery period.

Fig. 6.8 Average queue for EV from approach 2 after transition 2

Fig. 6.9 Overall average queue for EV from approach 2 after transition 2

To convert average queue lengths over all scenarios into delay time, we use average discharge speed, $S_{dis} = 4.44$ m/s (or 16 km/h), and average spacing between successive vehicles in a queue, $S_{spacing} = 8$ m.

After transferring the average queue lengths into average delay times, shown in Table 6.13, the optimal control algorithm reduced average delay time by 3.9 seconds (i.e., $11.26 - 7.36 = 3.9$ sec), 1.2 seconds (i.e., $5.06 - 3.86 = 1.2$ sec), and 6.87 seconds (i.e., $19.17 - 12.3 = 6.87$ sec) on approach 2, 3, and 4, respectively, while the average delay time on approach 1 was increased by 1.62 seconds (i.e., $33.23 - 34.85 = -1.62$ sec). Overall saving in average delay time was 10.36 seconds (i.e., $4 \times 17.18 - 4 \times 14.59 = 10.36$ sec) at the intersection. Figures 6.10 and 6.11 show in diagram form the comparison results.

Table 6.13 Average delay time over all scenarios for EV from approach 2

		Base case	Smooth transitioning	Optimal control transitioning
Approach 2	Average (sec)	13.06	11.26	7.36
	EVSP/Base case		0.86	0.56
Approach 3	Average (sec)	21.26	5.06	3.86
	EVSP/Base case		0.24	0.18
Approach 4	Average (sec)	24.41	19.17	12.30
	EVSP/Base case		0.79	0.50
Approach 1	Average (sec)	33.87	33.23	34.85
	EVSP/Base case		0.98	1.03
Average on all approaches	Average (sec)	23.15	17.18	14.59
	EVSP/Base case		0.74	0.63

Notes: (1) Base case=No EVSP.

(2) EVSP/Base case=ratio of average delay time under each control strategy to that of base case.

Fig. 6.10 Average delay time relative to base case for EV from approach 2 after transition 2

Fig. 6.11. Overall average delay time relative to base case for EV from approach 2 after transition 2

Only about half saving (10.36 sec) in delay time was obtained for cases when the EV entered from side road, compared with 22.39 seconds saving in delay time for cases of the EV entering from main road. This can be explained from two aspects. First, for cases when the EV entered from main road, a longer average queue (11.61 vehicles, see Table 5.15) was formed after the 50-second fixed held time period for transition 1, while for cases when the EV entered from side road, a shorter average queue (9.43 vehicles, see Table 5.20) was formed after the 20-second fixed held time period for transition 1. The longer traffic queue base for the former case determined that the optimal control strategy could extract more saving in absolute delay time.

Second, traffic lights are always green for the road from which the EV entered the traffic control system when the EV has just passed through the intersection, and from here transition 2 begins under the optimal control strategy. For cases when the EV entered from main road, our optimal control algorithm usually kept the green indication to main road because of long queues on this road with heavier traffic. It did not have to give green to side road if there was no benefit to do so. On the other hand, for cases when the EV entered from side road, the optimal control algorithm usually had to give green to main road because longer queues had formed on main road immediately after the EV cleared the intersection. A period of time consisting of yellow and all red intervals was wasted during transition 2 since no traffic on either main road or side road could make use of it.

This analysis is supported by simulation results. The relative saving in average delay time can be indicated by the ratio of average absolute saving in delay time to average recovery time. For cases when the EV entered from the main road, the ratio is 28% (i.e., $22.39 \text{ sec} / 79.83 \text{ sec} = 28\%$), and for cases when the EV entered from side road, the ratio is only 15% (i.e., $10.35 \text{ sec} / 69.17 \text{ sec} = 15\%$).

Chapter 7 Simulation and Comparison of Control Strategies for a Coordinated Route

Up to this point, only one single intersection has been simulated. Are the control strategies still feasible for coordinated traffic network? The answer is "yes" because the background signal cycle is maintained after the transitioning. The real-time control strategy and the optimal control strategy are applicable whether normal traffic control system is coordinated or not.

To illustrate the effectiveness and efficiency of our control strategies, the simulation of a coordinated route is described in this chapter, and then the results are analyzed against simulation results of the fixed held time control algorithm for transition 1 and the smooth control algorithm for transition 2, respectively.

7.1 Given conditions

Figure 7.1 illustrates the layout of the subject coordinated route. There are four signalized intersections, 1, 2, 3, and 4, along the route, and the EV is going to travel from intersection 1 to 4. The distance between intersection i and j is denoted by L_{ij} . The vehicular volumes at an intersection i are represented by V_{iE} , V_{iN} , V_{iS} , and V_{iW} for

eastbound, northbound, southbound, and westbound approaches, respectively. Table 7.1 lists all the given traffic conditions associated with the subject route.

Fig. 7.1 Layout of subject coordinated route

Table 7.1 Traffic parameters for subject EV route

		Intersection #			
		1	2	3	4
L_{ij} (m)		500	300	400	N/A
Volume (veh/h)	V_{iE}	900	V_{2E}	V_{3E}	V_{4E}
	V_{iN}	750	50	300	400
	V_{iS}	400	300	50	650
	V_{iW}	V_{1W}	V_{2W}	V_{3W}	800
Turning percentage	LT	4%			
	TH	90%			
	RT	6%			
Turning factor	LT	1.4			
	TH	1.0			
	RT	1.2			

Notes: (1) LT = left-turn; TH = through; and RT = right-turn.

(2) V_{2E} , V_{3E} , V_{4E} , V_{1W} , V_{2W} , and V_{3W} vary and are to be determined by traffic conditions and signal control at corresponding upstream intersections.

Other traffic parameters include:

- General vehicle operating speed, $S_{\text{mean}} = 12 \text{ m/s}$;
- EV operating speed, $S_{\text{EV}} = 15 \text{ m/s}$;
- Saturation flow = 2000 passenger cars/hour of green time/approach;
- Discharge rate during yellow interval = saturation flow/2;
- Two-phase signal system (see Fig. 5.3) at each intersection; and
- Lost time per phase = 3 sec.

7.2 Simulation of normal operation

We will use the same normal signal control parameters as for the single intersection in Chapter 5, i.e.,

- total green time for each phase: $G_{1W} = 60 \text{ sec}$; $G_{2W} = 40 \text{ sec}$;
- yellow interval = 3 sec;
- all red interval = 2 sec (1 sec. before green intervals for each phase).

In practical operation, signal control parameters should be determined according to actual traffic volumes. However, in this chapter we intentionally choose a traffic signal timing plan not very suitable for the subject route because we want to see how and to what extent an inappropriate timing plan will affect the results of our control strategies.

Another reason to do so is that it is easier to compare simulation results for the single intersection (in Chapters 5 and 6) and an intersection in a coordinated route (in this chapter) than using different signal timing plans for them.

The signal offset between two intersections is simply determined using the quotient of link length and general vehicle operating speed. In the following simulations we use 0, 40, 25, and 30 seconds as offsets at intersection 1, 2, 3, and 4, respectively.

After 20 cycles were simulated, the average queue lengths were calculated and listed in Table 7.2.

Table 7.2 Average queue length under normal operation

Intersection #		Queue (veh) on approach			
		EB	NB	SB	WB
1	mean	11.45	28.7	7.8	11.05
	std	2.11	15.08	1.51	7.44
	mean+2std	15.7	58.9	10.8	25.9
2	mean	15.95	0.3	5.95	10.65
	std	4.12	0.47	1.82	2.43
	mean+2std	23.2	1.2	9.6	15.5
3	mean	11.8	5.35	0.7	10.3
	std	9.91	1.14	0.47	3.40
	mean+2std	31.6	7.6	1.6	17.1
4	mean	14.45	8.35	18.15	10.35
	std	9.27	1.60	4.17	2.37
	mean+2std	33.0	11.6	26.5	15.1

Notes: 1. queue lengths are measured at the end of each red interval.

2. mean = mean of queue length.

3. std = standard deviation of queue length.

4. EB = eastbound; NB = northbound; SB = southbound; and WB = westbound.

7.3 Simulation and comparison of control strategies for transition 1

For comparison, a 50-second fixed held time control strategy is first simulated as reference, and then followed by the simulation for our real-time control strategy. EVSP scenarios are listed in Table 7.3. The EV enters the system from eastbound approach to intersection 1, and EV entry time (sec) is measured from the beginning of green(1) interval during 5th to 10th cycle period for normal operation, respectively. Simulation results are to be averaged over the four EV scenarios and the six cycles simulated.

Table 7.3 EVSP scenarios for subject route

scenario #	EV entry time (sec)	Remarks
0-base case	N/A	no EVSP
1	$t = 0$	beginning of G_{1w}
2	$t = \text{Green}(1)/2$	half G_{1w}
3	$t = \text{Green}(1)$	beginning of yellow
4	$t = \text{Green}(1) + \text{Yellow}(1) + \text{All_red}(1) + \text{Green}(2)/2$	half G_{2w}

7.3.1 Simulation of fixed held time control strategies

A period of 50 seconds was used for the fixed held time control algorithm. Table 7.4 summarizes the simulation results for comparison with our real-time control algorithm.

For all scenarios and all intersections, 50 seconds were long enough to clear queues ahead of the EV. It means that the 50-second fixed held time control strategy was

successful in EVSP for the subject route.

Table 7.4 Simulation results of 50-second fixed held time control strategy

Intersection #		Spare time(sec)	Queue (veh) on Approach			
			EB	NB	SB	WB
1	mean	41.71	12.3	22.5	8.4	0.4
	std	9.38	2.49	8.81	3.43	0.58
2	mean	38.18	10.4	0.7	5.54	0.5
	std	11.95	5.98	0.82	3.05	0.51
3	mean	42.19	9.8	6.0	1.0	0.4
	std	8.27	5.75	2.70	0.81	0.72
4	mean	41.72	11.5	8.8	21.2	0.4
	std	9.46	7.82	3.82	6.30	0.50

Notes: 1. queue lengths are measured at the end of each transitioning.

2. mean = mean of spare time or queue length.

3. std = standard deviation of spare or queue length.

4. EB = eastbound; NB = northbound; SB = southbound; and WB = westbound.

7.3.2 Simulation of real-time control strategy

The same parameters for the single intersection control are used for the route control, and the (average queue length + 2 times standard deviation) under normal operation is used to determine EV detection distances away from intersections. For our subject route, 450, 600, 770, and 785 meters are used for intersection 1, 2, 3, and 4, respectively.

Please note, since link lengths between two successive intersections are shorter than corresponding required EV detection distances (i.e., $500\text{m} < 600\text{m}$, $300\text{m} < 700\text{m}$, and $400\text{m} < 785\text{m}$ for intersections 2, 3, 4, respectively), the EV detection point for an intersection will be beyond its upstream intersection.

Based on the real-time control algorithm for coordinated route, simulations were conducted and results are summarized in Table 7.5.

For all scenarios and all intersections, the real-time control algorithm was successful in clearing the way for the EV to traverse the route without any delay.

Table 7.5 Simulation results of real-time control strategy

Intersection #		Transition period (sec)	Green for side road (sec)	Spare time (sec)	Queue (veh) on approach			
					EB	NB	SB	WB
1	mean	30	14.45	2.26	7.6	10.7	2.0	5.96
	std	/	9.14	3.86	2.06	6.94	1.60	4.76
2	mean	40	20.06	2.79	8.08	0.17	1.25	5.83
	std	/	10.98	4.30	5.96	0.38	1.45	4.90
3	mean	51.3	37.30	0.22	10.0	1.0	0.13	8.9
	std	/	6.32	1.06	5.59	1.06	0.34	7.78
4	mean	52.3	38.05	0	11.83	1.67	3.96	9.42
	std	/	7.69	0	7.89	1.43	2.42	4.41

- Notes: 1. queue lengths are measured at the end of each transitioning
 2. mean = mean of spare time or queue length.
 3. std = standard deviation of spare time or queue length.
 4. EB = eastbound; NB = northbound; SB = southbound; and WB = westbound.

7.3.3 Analysis and comparison of simulation results for transition 1

Simulation results show that both the 50-second fixed held time control and the real-time control were successful in clearing the queue at each intersection before the EV

crossed the intersection. It means that the EV crossed the route smoothly and no delay occurred during the EVSP.

Table 7.6 lists average spare times on each eastbound approach along the route. As indicated in the table, the real-time control wasted little green times, and an average of only 1.32 seconds was not used by the EV route or by the cross road at each intersection. On the other hand, the fixed held time control strategy incurred much longer spare times at all intersections and the average spare time over the whole route was 40.95 seconds. It means that an average of only 9.05 seconds was actually used to clear queue at an intersection for each EVSP occurrence.

Table 7.6 Average spare time (sec) under control strategies for transition 1

Intersection #		Fixed time $t_f = 50$ sec	Real-time
1	mean	41.71	2.26
	std	9.38	3.86
2	mean	38.18	2.79
	std	11.95	4.30
3	mean	42.19	0.22
	std	8.27	1.06
4	mean	41.72	0
	std	9.46	0
Overall average		40.95	1.32

Note: Real-time transitioning period = 30, 40, 51.3, and 52.3 seconds for intersection 1, 2, 3, and 4, respectively.

Green time for side road is only applicable to the real-time control strategy since the fixed held time control strategy always gives green to EV route during the whole transition duration. We can see from Table 7.5, the real-time control strategy extracted

14.45, 20.06, 37.30, and 38.05 seconds to clear queues on the side roads during the EVSP duration for intersection 1, 2, 3, and 4, respectively. For all the four intersections along the route, an average of 109.86 seconds was extracted as green time for the side roads during each EVSP occurrence

Figures 7.2 to 7.6 compare queue lengths under the fixed held time control and the real-time control in diagram form. On almost all eastbound, northbound, and southbound approaches, the real-time control strategy produced shorter queues except on eastbound approaches to intersections 3 and 4 with nearly equal queues of 10 and 11.83 vehicles to 9.8 and 11.5 vehicles produced by the fixed held time control strategy. On all westbound approaches, the real-time control strategy incurred longer queues because it diverted some green time from the EV route to the cross road. In our simulation, westbound traffic was allowed to pass through the intersection during EVSP duration. This assumption implies that the longer the EVSP duration and thereafter the green time for side road, the longer the red interval westbound traffic will confront. If traffic on side road is much lighter than that on EV route, the diversion does not necessarily benefit the whole intersection. As shown in Fig. 7.6, the average queue at intersection 2 under the real-time control strategy approached to that under the fixed held time control strategy (3.83 vs. 4.29). At intersection 3, the ratio even exceeded 1.0 (5.01 vs. 4.30). When traffic on side road was relative heavy, the benefit obtained from the diversion also increased, as

Fig. 7.2 Average queues at intersection 1 after transition 1

Fig. 7.3 Average queues at intersection 2 after transition 1

Fig. 7.4 Average queues at intersection 3 after transition 1

Fig. 7.5 Average queues at intersection 4 after transition 1

Fig. 7.6 Average queues along the route after transition 1

indicated in Fig. 7.6 for intersections 1 and 4. Nevertheless, over all the four intersections, the real-time control strategy reduced average queue to 5.53 from 7.49 vehicles under the fixed held time control strategy, i.e., a 26.2% ($= (7.49 - 5.53) / 7.49 = 0.262$) rises in control efficiency.

7.4 Simulation and comparison of control strategies for transition 2

We will first obtain simulation results under the smooth control strategy as reference, and then the optimal control strategy will be applied to the same traffic situations to show its effectiveness and efficiency. Similar to the single intersection control, in order to focus on the comparison between these two control strategies, a 50-second fixed held time control strategy for transition 1 is used to produce queue lengths on each approach in the subject route at the end of transition 1. From here, control strategies for transition 2 take the responsibility to recover the traffic signal to normal operation in a quick and efficient manner.

7.4.1 Simulation of smooth transitioning algorithm

In the following simulation of route control, we use the parameters and algorithm of smooth transitioning that were used for the single intersection control (described in

Chapter 6). The EV enters the route at the beginning, middle, and end of green interval for the EV route, and at the middle of red interval for the EV route, respectively. In order to consider the randomness of traffic situations, six occurrences for each EVSP scenario were conducted separately and then simulation results were averaged. Table 7.7 summarizes simulation results under the smooth control strategy.

Table 7.7 Simulation results of smooth transitioning algorithm

Intersection #		Recovery time (sec)	Queue (veh) on approach			
			EB	NB	SB	WB
1	mean	189	14.63	17.71	0.92	11.58
	std	47.77	4.60	11.34	1.10	2.90
2	mean	189	15.33	0	0.67	12.54
	std	47.77	5.83	0	0.87	0.98
3	mean	189	10.79	0.5	0	9.38
	std	47.77	4.01	0.51	0	3.27
4	mean	189	7.29	1.38	12.79	10.46
	std	47.77	5.77	1.53	9.86	2.59

- Notes: 1. queue lengths are measured at the end of each transitioning.
 2. mean = mean of recovery time or queue length.
 3. std = standard deviation of recovery time or queue length.
 4. EB = eastbound; NB = northbound; SB = southbound; and WB = westbound.

7.4.2 Simulation of optimal control strategy

Transitioning parameters and procedure for the single intersection described in Chapter 6 are also used for the coordinated route control, except that average queue

lengths from normal operation are different, and they are obtained from the simulation of the route control. From Table 7.2 we get average queue lengths on each approach in the route and we will use these values for the optimal control algorithm.

There are also four EVSP scenarios, i.e., the EV enters the route at the beginning, middle, and end of green interval for the EV route, and at the middle of red interval for the EV route, respectively. In order to consider the randomness of traffic situations, six occurrences for each EVSP scenario were studied separately and then simulation results were averaged for consideration of traffic fluctuation. Table 7.8 summarizes simulation results under the optimal control strategy.

Table 7.8 Simulation results of optimal control strategy

Intersection #		Recovery time(sec)	Queue (veh) on approach			
			EB	NB	SB	WB
1	mean	68.5	7.36	24.87	0.14	3.33
	std	29.51	3.38	11.02	0.52	2.63
2	mean	68.5	8.36	0.01	0	3.40
	std	29.51	4.36	0.04	0	2.40
3	mean	68.5	7.99	0.67	0	1.82
	std	29.51	4.63	1.64	0	1.91
4	mean	68.5	5.19	4.81	13.01	4.05
	std	29.51	6.74	3.85	7.65	1.99

- Notes: 1. queue lengths are measured at the end of each transitioning.
 2. mean = mean of recovery time or queue length.
 3. std = standard deviation of recovery time or queue length.
 4. EB = eastbound; NB = northbound; SB = southbound; and WB = westbound.

7.4.3 Analysis and comparison of simulation results

Table 7.9 presents a comparison of recovery time. It shows that average recovery times and standard deviations for every intersection were the same under each control strategy. This resulted from the specific transitioning algorithm for transition 1 on which the smooth control and the optimal control are based. The 50-second fixed held time control strategy determined that every intersection was treated in the same way in terms of starting instant and end instant of transition 1.

From Table 7.9 we can see that the optimal control strategy needed only 36.2% ($=68.5/189$) of recovery time under the smooth control algorithm to get traffic signal back to normal operation. While our optimal control strategy obviously outperformed the smooth control algorithm in terms of recovery time, we cannot reach a conclusion until another factor in assessment of control algorithm, namely average queue length experienced, has been studied.

Figures 7.7 to 7.11 compare average queue lengths on individual approaches to each intersection in the route in diagram form.

For all the 16 approaches, there were four that had longer queue lengths under the optimal control than under the smooth control. While all of them can be explained by sacrificing approach(es) for the benefit of the whole intersection under the optimal

control strategy, another reason for the longer queue lengths on northbound approach to intersection 1 (Fig. 7.7) and southbound approach to intersection 4 (Fig. 7.10) is that the average queue lengths under normal operation were relatively too long (28.7 and 18.15 vehicles respectively, see Table 7.2), which resulted from the inappropriate normal operation timing plan that we intentionally chose. It indicates that normal signal plan does have impact on the efficiency of the optimal control algorithm because the optimal control algorithm uses the average queue length under normal operation as one of its control parameters. An optimized normal operation plan is expected to provide a better base for the application of the optimal control strategy.

Table 7.9 Average recovery time (sec) under control strategies for transition 2

Intersection #		Smooth transitioning	Optimal control
1	mean	189	68.5
	std	47.77	29.51
2	mean	189	68.5
	std	47.77	29.51
3	mean	189	68.5
	std	47.77	29.51
4	mean	189	68.5
	std	47.77	29.51
Overall average		189	68.5

Notes: mean = average recovery time; and std = standard deviation.

Fig. 7.7 Average queues at intersection 1 after transition 2

Fig. 7.8 Average queues at intersection 2 after transition 2

Fig. 7.9 Average queues at intersection 3 after transition 2

Fig. 7.10 Average queues at intersection 4 after transition 2

Fig. 7.11 Average queues along the route after transition 2

Although the normal signal timing plan was not appropriate for the subject route, the optimal control strategy performed very well compared with the smooth control algorithm. As shown in Fig. 7.11, the optimal control strategy produced shorter average queues on every intersection in the route. Over the whole route it produced 5.31 vehicles of average queue on each approach compared with 7.87 vehicles incurred by the smooth transitioning, and that was a 32.5% (i.e., $(7.87-5.31)/7.87=0.325$) rise in control efficiency.

Based on the above comparison of recovery time and average queue length, it is suggested the optimal control strategy is potentially a better choice as a recovery procedure for traffic signal operation from EVSP back to normal operation.

Chapter 8 Control Efficiency at Different Saturation levels

Although we must use absolute traffic volumes and signal timings for simulation at a certain intersection, it is relative values of traffic volume compared with saturation flow that is of interest for obtaining a general guideline to assess control strategies and select control parameters. Therefore, after simulations are conducted for certain combinations of traffic volumes and signal timings, results will be presented in terms of ratios of traffic volumes to saturation flows and ratios of queues to corresponding queues formed under normal signal operation, respectively.

8.1 Normal signal operation

The simplest four-leg intersection with two-phase signal system is still used for simulation in this chapter (see Figure 8.1). All lanes on each approach are in one lane group. Because critical approach volumes determine normal signal timing plan and EVSP efficiency, we will fix the traffic volume on an opposing approach at 0.8 times volume on its corresponding critical approach. As shown in Figure 8.1, approaches 1 and 2 are critical approaches, approaches 3 and 4 are opposing approaches, and the EV route is from approach 1 to approach 3

Fig. 8.1 Critical approaches and relative traffic volumes

Ratios, r_1 and r_2 , of traffic volumes on approaches 1 and 2 to saturation flows vary from scenario to scenario for each simulation, and they are listed in Table 8.1.

For simulation, we will use a fixed saturation flow, 2000 passenger cars/hour of green time for each approach, and then traffic flows on individual approaches can be calculated as follows.

$$\text{Traffic flow on approach 1} = 2000 * r_1 ;$$

$$\text{Traffic flow on approach 2} = 2000 * r_2 ;$$

$$\text{Traffic flow on approach 3} = 0.8 * 2000 * r_1 ;$$

$$\text{Traffic flow on approach 4} = 0.8 * 2000 * r_2 ;$$

Table 8.1 Simulation scenarios and ratios of traffic volumes to saturation flows

Scenario #	Approach 1: r_1	Approach 2: r_2	$r_2 + r_1$	Remarks
1	0.3	0.1	0.4	
2		0.2	0.5	
3		0.3	0.6	
4		0.4	0.7	
5	0.4	0.1	0.5	
6		0.2	0.6	
7		0.3	0.7	
8		0.4	0.8	
9	0.5	0.1	0.6	
10		0.2	0.7	
11		0.3	0.8	
12		0.4	0.9	
13	0.6	0.1	0.7	
14		0.2	0.8	
15		0.3	0.9	
16		0.4	1.0	
17	0.7	0.1	0.8	
18		0.2	0.9	
19		0.3	1.0	
20		0.4	1.1	

In practice, signal timing for normal operation should be determined by actual traffic flows. However, in order to focus on the impact of traffic volumes at different saturation levels on EVSP control strategies, we will use the fixed signal timing plan obtained in Chapter 5, i.e.,

$$\text{green}(1) = 60 \text{ sec}$$

$$\text{green}(2) = 40 \text{ sec}$$

$$\text{yellow}(1) = \text{yellow}(2) = 3 \text{ sec}$$

$$\text{all_red}(1) = \text{all_red}(2) = 1 \text{ sec}$$

$$\text{cycle length} = 108 \text{ sec}$$

From the above traffic flows and traffic signal timing plan, capacities and volume/capacity ratios (or degrees of saturation) can be calculated using the following equations.

$$\text{capacity} = \text{saturation flow} * \text{green interval} / \text{cycle length}$$

$$\text{volume/capacity ratio} = \text{traffic flow} / \text{capacity}$$

Therefore, we get volume/capacity ratio, x_i , for each approach.

$$x_1 = 2000 * r_1 / (2000 * 60 / 108) = 1.8 * r_1$$

$$x_2 = 2000 * r_2 / (2000 * 40 / 108) = 2.7 * r_2$$

$$x_3 = 0.8 * x_1$$

$$x_4 = 0.8 * x_2$$

Volume/capacity ratios for each approach are then listed in Table 8.2.

Table 8.2 Volume/capacity ratios, x_i , for each approach

r_1	Approach 1: x_1	Approach 3: x_3	r_2	Approach 2: x_2	Approach 4: x_4
0.3	0.54	0.432	0.1	0.27	0.216
0.4	0.72	0.576	0.2	0.54	0.432
0.5	0.9	0.72	0.3	0.81	0.648
0.6	1.08	0.864	0.4	1.08	0.864
0.7	1.26	1.008			

Note: $x_i > 1.0$ means that approach i has a traffic flow greater than its capacity.

Simulations under normal operation were first conducted, and simulation results for 20 cycles are averaged and summarized in Table 8.3.

Table 8.3 Average queue lengths (veh) under normal operation

Scenario #		Queue (veh) on approach			
		1	2	3	4
1	mean	7.2	3.3	7	3
	std	1.7	1.5	1.8	1.2
	mean+2std	11	6.2	11	5.3
2	mean	7.3	7.8	5.8	6
	std	1.5	1.9	1.7	1.8
	mean+2std	10	12	9.2	9.6
3	mean	8.2	12	7.3	9.9
	std	2	2.6	1.8	1.7
	mean+2std	12	17	11	13
4	mean	7.8	30	6.5	13
	std	1.8	6.4	1.6	4.7
	mean+2std	11	43	9.6	23
5	mean	11	3.3	8.8	2.9
	std	1.6	1.8	2.5	1.5
	mean+2std	14	6.9	14	5.8
6	mean	11	7.3	8.4	6.5
	std	3.5	1.7	1.8	2.1
	mean+2std	18	11	12	11
7	mean	12	12	8.9	13
	Std	2.3	3.2	1.6	1.9
	mean+2std	16	19	12	13
8	Mean	11	28	8.9	13
	Std	2	5.3	2.3	3
	mean+2std	15	38	14	19
9	Mean	16	3.9	11	3
	Std	4.3	0.88	2.5	1.5
	mean+2std	25	5.6	16	6

(continued on next page)

10	Mean	14	7.6	11	6.5
	Std	2.4	2.2	2	1.5
	mean+2std	18	12	15	9.4
11	mean	17	12	9.8	9.2
	std	4.5	2.4	2.6	2.4
	mean+2std	26	17	15	14
12	mean	12	28	11	13
	std	1.9	9.2	2.8	2.6
	mean+2std	16	46	17	19
13	mean	60	3.6	13	3
	std	24	1.3	2.4	1.2
	mean+2std	110	6.3	18	5.5
14	mean	36	8	13	5.9
	std	12	1.9	2.9	1.5
	mean+2std	61	12	18	8.9
15	mean	57	13	14	9.2
	std	15	1.9	3.5	2.3
	mean+2std	86	17	21	14
16	mean	43	44	14	13
	std	13	18	4.4	2.7
	mean+2std	68	80	23	19
17	mean	130	3.6	39	3.4
	std	60	1.4	13	1.1
	mean+2std	250	6.5	64	5.7
18	mean	100	8	35	5.8
	std	45	2	14	2.2
	mean+2std	190	12	64	10
19	mean	100	11	39	9.3
	std	56	2.4	11	2.4
	mean +2std	220	16	61	14
20	mean	100	35	28	13
	std	51	18	11	2.2
	mean+2std	200	70	50	17

- Notes: 1. Queue lengths are measured at the end of each red interval.
2. mean = mean of queue length.
3. std = standard deviation of queue length.

From the above table, we can see that queue lengths on approach 1 increased rapidly for scenarios 17 through 20. Obviously the signal timing plan was not suitable for these traffic scenarios and was not able to keep queues under a reasonable length. Therefore, we will not further discuss these scenarios in the following sections.

The average queues for individual traffic ratios are calculated and listed in Table

8.4

Table 8.4 Average queue lengths (veh) for each traffic ratio under normal operation

r_1	Approach 1 q_{N1}	Approach 3 q_{N3}	r_2	Approach 2 q_{N2}	Approach 4 q_{N4}
0.3	7.625	6.65	0.1	3.525	2.975
0.4	11.25	8.75	0.2	7.675	6.225
0.5	14.75	10.7	0.3	12.25	10.325
0.6	49	13.5	0.4	32.5	13

8.2 Simulation of real-time control strategy for transition 1

Based on mean queue lengths under normal operation, EV detection distances on approach 1 are calculated using equations established in Chapter 3, and they are listed in Table 8.5.

Four EV occurrence scenarios are listed in Table 7.3 (EV entry time = 0, green(1)/2, green(1), and green(1)+yellow(1)+all_red(1)+green(2)/2, respectively). Simulation results are averaged over the four EV scenarios and six (5^{th} to 10^{th}) cycles.

Table 8.6 summarizes simulation results.

The control strategy failed in one case for scenario 12 and one case for scenario 14. The former failed because the queue on approach 1 was too long to be discharged before the EV arrived at the intersection, and the later failed because the time needed to discharge the queue on approach 1 was greater than the 100-second pre-determined maximum green time.

Table 8.5 EV detection distances (m) on approach 1 away from the intersection

Scenario #	EV detection distances (m)	Remark
1	430	mean+2std = 11 veh
2		
3		
4		
5	570	mean+2std = 16 veh
6		
7		
8		
9	700	mean+2std = 21 veh
10		
11		
12		
13	2320	mean+2std = 81 veh
14		
15		
16		

Table 8.6 Simulation results of real-time control strategy

Scenario #		Transition period (sec)	Green for side road (sec)	time Spare (sec)	Queue (veh) on approach			
					1	2	3	4
1	mean	28.7	16.5	1.4	5.38	0.75	2.5	0.38
	std	/	6.15	3.69	1.53	0.94	1.87	0.58
2	mean	28.7	15.9	1.3	4.88	1.75	3.25	1.25
	std	/	6.93	3.70	1.75	1.94	2.23	1.26
3	mean	28.7	15.1	2.1	5.04	5.29	3.13	2.33
	std	/	8.17	4.05	1.73	3.21	2.66	1.93
4	mean	28.7	16.0	1.4	5.13	9.75	3.38	3.96
	std	/	6.89	3.39	1.57	5.42	2.02	2.60
5	mean	38	25.4	0.2	7.71	0.45	4.50	0.54
	std	/	5.31	1.00	1.78	0.66	2.87	0.59
6	mean	38	22.6	1.1	9.38	1.54	3.88	1.5
	std	/	7.87	2.53	1.79	1.50	2.85	1.44
7	mean	38	23.2	1.0	8.58	2.38	4.25	2.13
	std	/	8.29	3.04	1.79	1.28	3.40	1.94
8	mean	38	23.2	0.9	8.21	9.83	4.75	2.42
	std	/	7.70	2.60	2.26	4.87	3.29	1.79
9	mean	46.7	30.4	1.0	12.21	1.29	7.42	0.75
	std	/	10.06	2.99	2.11	1.33	3.99	0.79
10	mean	46.7	27.7	0.5	12.08	2.21	4.46	1.29
	std	/	9.51	1.43	2.65	1.89	4.25	1.16
11	mean	46.7	27.9	0.6	13.21	3.08	5.5	2.13
	std	/	9.13	1.61	2.26	1.77	4.51	1.39
12 ⁽⁴⁾	mean	46.7	28.8	1.3	12.83	19.58	6.25	3.38
	std	/	10.38	3.00	2.51	7.40	4.09	2.81
13	mean	154.7	98.0	15.4	46.54	2.83	4.33	2.67
	std	/	4.24	14.60	3.84	1.20	3.70	1.24
14 ⁽⁵⁾	mean	154.7	83.3	3.4	53.0	7.75	0.54	5.92
	std	/	15.57	6.91	12.24	2.45	1.61	2.45

(continued on next page)

15	mean	154.7	93.13	6.2	47.75	9.67	4.38	8.54
	std	/	9.15	10.22	3.67	1.79	3.68	2.70
16	mean	154.7	89.97	6.6	52.63	14.71	3.25	9.96
	std	/	12.90	10.24	5.12	4.79	3.33	2.74

- Notes: 1. Queue lengths are measured at the end of each transition.
2. mean = mean of spare time or queue length.
3. std = standard deviation of spare time or queue length.
4. The control strategy failed in one case.
5. The control strategy failed in one case.

The above simulation results are averaged for individual traffic ratios and are listed in Table 8.7. In the table, average queues on approaches 2 and 4 are for traffic ratio $r_2 = 0.1, 0.2, 0.3$ and 0.4 , respectively; and other measurements are for traffic ratio $r_1 = 0.3, 0.4, 0.5$ and 0.6 , respectively.

Table 8.7 Averages of simulation results for each traffic ratio under real-time control strategy

r_1/r_2	Green for side road g_{side} (sec)	Spare time t_{spare} (sec)	Queue (veh) on approach			
			1 q_{R1}	2 q_{R2}	3 q_{R3}	4 q_{R4}
0.3/0.1	15.9	1.6	5.1	1.33	3.1	1.09
0.4/0.2	23.6	0.8	8.5	3.31	4.3	2.49
0.5/0.3	28.7	0.9	12.6	5.11	5.9	3.78
0.6/0.4	91.1	7.9	50.0	13.47	3.1	4.93

Note: r_2 is used for queues on approaches 2 and 4, and r_1 is used for other measurements.

8.3 Simulation of optimal control strategy for transition 2

In this section, we will use identities as weighting matrices in order to focus on the

impact of traffic volumes on the efficiency of the optimal control strategy. The difference from previous chapters is that we will adopt the real-time control strategy and the optimal control strategy for transition 1 and transition 2 sequentially during a whole EVSP process. That is, immediately after an EV passes the intersection under the real-time control strategy, the optimal control strategy starts its recovery procedure. The purpose to combine them is to assess their control efficiencies for EVSP in a real traffic environment.

Following simulations for individual EV scenarios and cycles under the real-time control strategy, the optimal control strategy continues the transition process until traffic signal gets back to normal operation. Average queue lengths from the normal operation are to be used as control parameters for the optimal control strategy, and they are listed in Table 8.3. Simulation results are averaged over the four EV scenarios and the six (5th to 10th) cycles, and they are summarized in Table 8.8.

Table 8.8 Simulation results of optimal control strategy

Scenario #		Recovery time(sec)	Queue (veh) on approach			
			1	2	3	4
1	mean	89.8	0.14	0.17	0.20	0.41
	std	29.5	0.51	0.26	0.97	0.71
2	mean	89.8	0.06	0.06	0	0.05
	std	29.5	0.25	0.20	0	0.22

(continued on next page)

3	mean	89.8	1.22	0.49	0.70	0.01
	std	29.5	1.00	0.86	0.80	0.05
4	mean	89.8	1.54	12.06	1.25	2.08
	std	29.5	1.93	6.80	1.39	2.60
5	mean	80.5	0.65	0.24	0.10	0.15
	std	29.51	1.30	0.51	0.28	0.23
6	mean	80.5	0.68	1.44	0	0.27
	std	29.51	1.15	1.71	0	1.00
7	mean	80.5	1.33	1.41	0.08	0.28
	std	29.51	1.57	1.79	0.16	0.81
8	mean	80.5	2.88	16.59	0.51	2.97
	std	29.51	2.27	5.67	0.92	2.98
9	mean	71.8	5.49	0.78	0.86	0.36
	std	29.51	2.77	1.03	1.96	0.71
10	mean	71.8	7.56	2.75	0.57	1.54
	std	29.51	4.72	2.04	1.58	1.67
11	mean	71.8	8.12	6.20	0.44	2.65
	std	29.51	3.56	3.79	1.14	2.74
12	mean	71.8	9.38	16.32	2.03	6.23
	std	29.51	4.74	5.12	2.00	3.94
13	mean	71.8	51.25	1.17	4.63	0.76
	std	29.51	5.49	1.23	5.41	0.89
14	mean	71.8	52.39	3.09	3.78	0.92
	std	29.51	6.05	2.20	3.96	1.20
15	mean	71.8	53.69	7.56	4.37	2.72
	std	29.51	8.30	2.99	3.84	1.94
16	mean	71.8	54.29	17.00	4.43	8.21
	std	29.51	5.01	6.47	3.55	3.47

Notes: 1. queue lengths are measured at the end of each transitioning.

2. mean = mean of recovery time or queue length.

3. std = standard deviation of recovery time or queue length.

The above simulation results are averaged for individual traffic ratios and are listed in Table 8.9. In the table, average queues on approaches 1 and 3 are for traffic ratio $r_1 = 0.3, 0.4, 0.5$ and 0.6 , respectively; and average queues on approaches 2 and 4

are for traffic ratio $r_2 = 0.1, 0.2, 0.3$, and 0.4 respectively.

Table 8.9 Averages of simulation results for each traffic ratios under optimal control strategy

r_1/r_2	Queue (veh) on approach			
	1 q_{01}	2 q_{02}	3 q_{03}	4 q_{04}
0.3/0.1	0.74	0.59	0.54	0.12
0.4/0.2	1.39	1.84	0.17	0.70
0.5/0.3	7.64	3.92	0.98	1.42
0.6/0.4	52.91	15.49	4.30	4.87

Note: r_1 is used for queues on approaches 2 and 4, and r_2 is used for queues on approaches 1 and 3.

8.4 Analysis of simulation results

The real-time control strategy failed in two cases. One was for scenario 12 ($r_1 = 0.5$ and $r_2 = 0.4$) and another one was for scenario 14 ($r_1 = 0.6$ and $r_2 = 0.2$). Since there were 24 cases (four EV occurrences and six cycles for each EV occurrence) for each scenario, the failure ratio was less than 5% ($1/24 = 4.2\%$). Therefore, we conclude that the real-time control strategy is still reliable.

It should also be noticed that the reasons for these two failures were different. The former failed because the EV arrived at the intersection before the queue on approach 1 was discharged, and the later failed because the 100-second pre-determined maximum green time was not long enough to discharge the queue on approach 1. However, for the

latter, the EV still could pass through the intersection without any delay during the transition period under the real-time control strategy. This difference probably resulted in the rapid increase in queue on approach 2 for scenario 12 (see Table 8.6).

In order to obtain a general conclusion, we represent the previous simulation results in relative values as follows. In Tables 8.10 through 8.14,

g_{side} = average green time for side road;

t_{spare} = average spare time;

TP = average transition period for transition 1;

q_{N1} , q_{N2} , q_{N3} , q_{N4} , = average queue lengths on approaches 1 through 4 under normal operation, respectively;

q_{R1} , q_{R2} , q_{R3} , q_{R4} , = average queue lengths on approaches 1 through 4 under the real-time control strategy, respectively; and

q_{O1} , q_{O2} , q_{O3} , q_{O4} , = average queue lengths on approaches 1 through 4 under the optimal control strategy, respectively;

We can see in Table 8.10 that only 1.9% to 5.6% of transition periods were wasted as spare times and more than half transition periods (from 55.4% to 62.1%) were diverted to the side road as green time. Since the EV detection point is determined such that nearly the longest possible queue (average +2 times standard deviation) at the intersection can be discharged before the EV passes the intersection without delay, and

given that under normal traffic conditions the queue is at about the average value, it is reasonable that about half a transition period is needed to discharge the average queue and another half can be diverted to the side road.

Table 8.10 Relative measurements for each traffic ratio under real-time control strategy

r_1	TP (sec)	g_{side} (sec)	$g_{\text{side}} / \text{TP}$ (%)	t_{spare} (sec)	$t_{\text{spare}} / \text{TP}$ (%)
0.3	28.7	15.9	55.4	1.6	5.6
0.4	38	23.6	62.1	0.8	2.1
0.5	46.7	28.7	61.5	0.9	1.9
0.6	154.7	91.1	58.9	7.9	5.1

Notes: g_{side} = average green time for side road over $r_2 = 0.1, 0.2, 0.3$, and 0.4 ; and

t_{spare} = average spare time over $r_2 = 0.1, 0.2, 0.3$, and 0.4 .

Since one of our objectives for the optimal control strategy is to control the queues at the intersection that are not longer than the average queues under normal operation so that traffic signal can resume to normal operation immediately after transition 2, we will compare the queues after transitions 1 and 2 with average queues under normal operation. Thereafter, the ratios of queues after transition 1 or transition 2 to average queues under normal operation will be used as relative queues to assess the efficiencies of our control strategies.

Tables 8.11 to 8.14 summarize the relative queues on individual approaches after transition 1 and transition 2, respectively. The relative queues are also illustrated in Figures 8.2 to 8.9.

Table 8.11 Relative queue lengths on EV route under real-time control strategy

Scenario #	Approach 1			Approach 3		
	q_{R1}	q_{N1}	q_{R1}/q_{N1} (%)	q_{R3}	q_{N3}	q_{R3}/q_{N3} (%)
1	5.38	7.625	70.6	2.5	6.65	37.6
2	4.88	7.625	64.0	3.25	6.65	48.9
3	5.04	7.625	66.1	3.13	6.65	47.1
4	5.13	7.625	67.3	3.38	6.65	50.8
5	7.71	11.25	68.5	4.50	8.75	51.4
6	9.38	11.25	83.4	3.88	8.75	44.3
7	8.58	11.25	76.3	4.25	8.75	48.6
8	8.21	11.25	73.0	4.75	8.75	54.3
9	12.21	14.75	82.8	7.42	10.7	69.3
10	12.08	14.75	81.9	4.46	10.7	41.7
11	13.21	14.75	89.6	5.5	10.7	51.4
12 ⁽¹⁾	12.83	14.75	87.0	6.25	10.7	58.4
13	46.54	49	95.0	4.33	13.5	32.1
14 ⁽²⁾	53.0	49	108.2	0.54	13.5	4.0
15	47.75	49	97.4	4.38	13.5	32.4
16	52.63	19	107.4	3.25	13.5	24.1

Notes: 1. The control strategy failed in one case.

2. The control strategy failed in one case.

Table 8.12 Relative queue lengths on side road under real-time control strategy

Scenario #	Approach 2			Approach 4		
	q_{R2}	q_{N2}	q_{R2}/q_{N2} (%)	q_{R4}	q_{N4}	q_{R4}/q_{N4} (%)
1	0.75	3.525	21.3	0.38	2.975	12.8
2	1.75	7.675	22.8	1.25	6.225	20.1
3	5.29	12.25	43.2	2.33	10.325	22.6
4	9.75	32.5	30.0	3.96	13	30.5
5	0.45	3.525	12.8	0.54	2.975	18.2
6	1.54	7.675	20.1	1.5	6.225	24.1
7	2.38	12.25	19.4	2.13	10.325	20.6
8	9.83	32.5	30.2	2.42	13	18.6
9	1.29	3.525	36.6	0.75	2.975	25.2
10	2.21	7.675	28.8	1.29	6.225	20.7

(continued on next page)

11	3.08	12.25	25.1	2.13	10.325	20.6
12 ⁽¹⁾	19.58	32.5	60.2	3.38	13	26.0
13	2.83	3.525	80.3	2.67	2.975	89.7
14 ⁽²⁾	7.75	7.675	101.0	5.92	6.225	95.1
15	9.67	12.25	78.9	8.54	10.325	82.7
16	14.71	32.5	45.3	9.96	13	76.6

Notes: 1. The control strategy failed in one case.

2. The control strategy failed in one case.

Table 8.13 Relative queue lengths on EV route under optimal control strategy

Scenario #	Approach 1			Approach 3		
	q ₀₁	q _{N1}	q ₀₁ / q _{N1} (%)	q ₀₃	q _{N3}	q ₀₃ / q _{N3} (%)
1	0.14	7.625	1.84	0.2	6.65	3.01
2	0.06	7.625	0.79	0	6.65	0.00
3	1.22	7.625	16.00	0.7	6.65	10.53
4	1.54	7.625	20.20	1.25	6.65	18.80
5	0.65	11.25	5.78	0.1	8.75	1.14
6	0.68	11.25	6.04	0	8.75	0.00
7	1.33	11.25	11.82	0.08	8.75	0.91
8	2.88	11.25	25.60	0.51	8.75	5.83
9	5.49	14.75	37.22	0.86	10.7	8.04
10	7.56	14.75	51.25	0.57	10.7	5.33
11	8.12	14.75	55.05	0.44	10.7	4.11
12	9.38	14.75	63.59	2.03	10.7	18.97
13	51.25	49	104.59	4.63	13.5	34.30
14	52.39	49	106.92	3.78	13.5	28.00
15	53.69	49	109.57	4.37	13.5	32.37
16	54.29	49	110.80	4.43	13.5	32.81

Table 8.14 Relative queue lengths on side road under optimal control strategy

Scenario #	Approach 2			Approach 4		
	q _{O2}	q _{N2}	q _{O2} / q _{N2} (%)	q _{O4}	q _{N4}	q _{O4} / q _{N4} (%)
1	0.17	3.525	4.82	0.41	2.975	13.78
2	0.06	7.675	0.78	0.05	6.225	0.80
3	0.49	12.25	4.00	0.01	10.325	0.10
4	12.06	32.5	37.11	2.08	13	16.00
5	0.24	3.525	6.81	0.15	2.975	5.04
6	1.44	7.675	18.76	0.27	6.225	4.34
7	1.41	12.25	11.51	0.28	10.325	2.71
8	16.59	32.5	51.05	2.97	13	22.85
9	0.78	3.525	22.13	0.36	2.975	12.10
10	2.75	7.675	35.83	1.54	6.225	24.74
11	6.2	12.25	50.61	2.65	10.325	25.67
12	16.32	32.5	50.22	6.23	13	47.92
13	1.17	3.525	33.19	0.76	2.975	25.55
14	3.09	7.675	40.26	0.92	6.225	14.78
15	7.56	12.25	61.71	2.72	10.325	26.34
16	17	32.5	52.31	8.21	13	63.15

Fig. 8.2 Relative queues on approach 1 after transition 1

Fig. 8.3 Relative queue on approach 3 after transition 1

Fig. 8.4 Relative queue on approach 2 after transition 1

Fig. 8.5 Relative queue on approach 4 after transition 1

Fig. 8.6 Relative queues on approach 1 after transition 2

Fig. 8.7 Relative queue on approach 3 after transition 2

Fig. 8.8 Relative queue on approach 2 after transition 2

Fig. 8.9 Relative queue on approach 4 after transition 2

(1) The real-time control strategy

The relative queues under the real-time control strategy are presented in Tables 8.11 and 8.12, and illustrated in Figures 8.2 to 8.5.

On approach 1 (the approach with the EV), the relative queues increased with the increase in traffic ratio r_1 . They were at about 70% of average queues of normal operation for $r_1=0.3$ and about 100% for $r_1=0.6$. The normal operation timing plan was determined according to traffic volumes 1000 (veh/hr) on approach 1 and 650 (veh/hr) on approach 2 (see Chapter 5 for detail), i.e., $r_1=0.5$ and $r_2=0.325$. Consequently, if the traffic is heavier than 1000 (veh/hr) on approach 1, the normal operation timing plan will not be able to discharge the queue on approach 1 during many individual cycles, and traffic queue will be accumulated between cycles. That is why 154.7 seconds were needed for the real-time control strategy to complete the transition 1 for $r_1=0.6$. This long transition period affected more vehicles and long queues were formed on approach 1 after transition 1.

On approach 3 (the opposite approach), relative queues were very short, especially for $r_1=0.6$. This can be explained from two perspectives. First, the traffic volume on approach 3 was selected as 0.8 times the volume on approach 1. Second, vehicles on approach 3 were allowed to pass through the intersection during the transition period whenever they faced green signal, and the long transition period for $r_1=0.6$ gave them a

long green interval to discharge the queue immediately before the completion of transition 1.

On approach 2 (the cross road), relative queues were shorter than 60.2% average queues of normal operation for $r_1=0.3$, 0.4, and 0.5. For $r_1=0.6$, long relative queues were formed on approach 1 and a long transition period was then needed. This long transition period caused long relative queues on approach 2. However, the relative queues for $r_1=0.6$ changed with traffic volumes on approach 2. When r_2 increased from 0.1 to 0.2, relative queues changed from 80.3% to 101%. Surprisingly, when r_2 continued to increase to 0.3 and then 0.4, relative queues decreased to 78.9% and then 45.3%. One explanation for the decrease is that the diversion of green time for the side road (about 90 seconds) relieved the accumulation of queues on approach 2 since only a 40-second green interval was available during each normal operation cycle.

On approach 4 (the cross road), everything was the same as on approach 2 except that the traffic volume on approach 4 was selected as 0.8 times the volume on approach 2. Therefore, the simulation results and their explanations are similar for approaches 2 and 4. For $r_1=0.3$, 0.4, and 0.5, the relative queues were shorter than 30.5%. For $r_1=0.6$, long relative queues on approach 1 and consequently a long transition period caused long relative queues on approach 4. After relative queues increased from 89.7% to 95.1% as r_2 changed from 0.1 to 0.2, they decreased to 82.7% and then 76.6% as r_2 changed to 0.3 and then to 0.4 because of the diversion of green time for the side road.

which in turn relieved the accumulation of queues on approach 4.

(2) The optimal control strategy

The relative queues under the optimal control strategy are shown in Tables 8.13 and 8.14, and illustrated in Figures 8.6 to 8.9.

On approach 1 (the EV approach), the relative queues increased with both r_2 and r_1 , and they were primarily affected by r_1 , which was the ratio of traffic volume to saturation flow on approach 1. For scenarios 1 through 12, the queues were controlled at levels shorter than their corresponding lengths of normal operation, and scenarios 12 ($r_1 = 0.5$ and $r_2 = 0.4$) had the longest one with 63.6% of queue length under normal operation. When traffic conditions were oversaturated for scenarios 13 through 16 ($r_1 = 0.6$), the optimal control strategy also produced longer relative queues by 4.6%, 6.9%, 9.6%, and 10.8% for scenarios 13, 14, 15, and 16, respectively, compared with their corresponding values of normal operation. From Fig. 8.6, one conclusion that can be drawn is that the difference in relative queues was primarily caused by traffic volumes (indicated by r_1 and r_2) instead of recovery period, because relative queues were longer for $r_1=0.6$ than for $r_1=0.5$ while the recovery period (71.8 seconds) was the same for both $r_1=0.5$ and $r_1=0.6$.

On approach 3 (the opposite approach), relative queues were very short with the longest at 34.3% of average queue for normal operation. This probably resulted from two factors. First, there existed short relative queues on approach 3 after transition 1.

from which the optimal control strategy began the recovery process. Second, traffic was lighter on approach 3 than on approach 1 while both approaches got the same green intervals. We can observe again from Fig. 8.7 that traffic volumes (indicated by r_1 and r_2) instead of recovery period primarily determined the relative queues, since there were big differences in relative queues for scenarios with $r_1=0.5$ and for those with $r_1=0.6$ while the recovery period was the same (71.8 seconds) for all the scenarios.

On approach 2 (the cross road), relative queues were controlled below 61.7% for all scenarios. From Fig. 8.8, we can see there were rapid increases in relative queues when traffic volumes approached or exceeded the control capacity of the normal operation timing plan. For scenarios with light traffic on approach 1 ($r_1=0.3$ or 0.4), the rapid increases occurred at $r_2=0.4$ (corresponding volume/capacity ratio = 1.08, see Table 8.2). In the case of scenarios with heavy traffic on approach 1 ($r_1=0.5$ or 0.6), the rapid increases occurred at $r_2=0.3$ (corresponding volume/capacity ratio = 0.81, see Table 8.2). It suggests that traffic conditions on the EV route were important factors to influence the relative queues on the cross road. However, when volume/capacity ratio on approach 2 was greater than 1.0 ($r_2=0.4$), the relative queues on approach 2 were primarily determined by traffic volume on this approach. As we can see from Table 8.14 and Fig 8.8, the relative queues on approach 2 had no major differences for scenarios with $r_2=0.4$, for all values of r_1 (i.e., $r_1=0.3, 0.4, 0.5$, or 0.6). Furthermore, the queues were shorter than their corresponding lengths under normal operation for all scenarios.

This means that even though traffic on the cross road was at an oversaturated level ($r_2=0.4$), it did not cause any problem for the optimal control strategy to get the traffic signal back to normal operation in an efficient manner.

On approach 4 (the cross road), relative queues were controlled at short levels for all scenarios, and the highest one (63.2%) appeared for scenario 16 with $r_1=0.6$ and $r_2=0.4$. Similar to results on approach 2, relative queues on approach 4 had rapid increases when traffic volume on this road approached its control capability of the normal operation timing plan, i.e., when volume/capacity ratio on this approach was equal to 0.864 ($r_2=0.4$, see Table 8.14 and Figure 8.9).

Chapter 9 Impact of Weighting Matrices on Efficiency of Optimal Control Strategy

In previous simulations, we used identities as weighting matrices in the optimal control algorithm for transition 2. In this section, we will investigate the impact of different weighting matrices on control efficiency.

Weighting matrices include P for fixed final state, P' for free final state, R for intermediate states, and S for control variables. As mentioned in Section 4.9, fixed final state corresponds to open-loop control, and our M-step model is not suitable for this situation. Therefore, the weighting matrix, P , for fixed final state will not be discussed further in this study.

9.1 Simulation of optimal control strategy

A set of combinations of weighting matrices is selected for simulation and they are listed in Table 9.1.

The optimal control strategy with each combination of weighting matrices will be applied to different traffic conditions, which are described by 16 scenarios listed in Table 8.1. Similar to the case described in Section 8.2, there are four EVSP scenarios (listed in Table 7.3) for each weighting matrix combination and each traffic scenario. Six cycles (5th to 10th) will be simulated to take into account traffic fluctuation. Simulation results are to be averaged over the four EVSP scenarios and the six cycles

Table 9.1 Combinations of weighting matrices

Combination #	P'	R	S
1	$I_{4 \times 4}$	$I_{4 \times 4}$	$I_{2 \times 2}$
2			5 $I_{2 \times 2}$
3			10 $I_{2 \times 2}$
4		5 $I_{4 \times 4}$	$I_{2 \times 2}$
5			5 $I_{2 \times 2}$
6			10 $I_{2 \times 2}$
7		10 $I_{4 \times 4}$	$I_{2 \times 2}$
8			5 $I_{2 \times 2}$
9			10 $I_{2 \times 2}$
10	5 $I_{4 \times 4}$	$I_{4 \times 4}$	$I_{2 \times 2}$
11			5 $I_{2 \times 2}$
12			10 $I_{2 \times 2}$
13		5 $I_{4 \times 4}$	$I_{2 \times 2}$
14			5 $I_{2 \times 2}$
15			10 $I_{2 \times 2}$
16		10 $I_{4 \times 4}$	$I_{2 \times 2}$
17			5 $I_{2 \times 2}$
18			10 $I_{2 \times 2}$
19	10 $I_{4 \times 4}$	$I_{4 \times 4}$	$I_{2 \times 2}$
20			5 $I_{2 \times 2}$
21			10 $I_{2 \times 2}$
22		5 $I_{4 \times 4}$	$I_{2 \times 2}$
23			5 $I_{2 \times 2}$
24			10 $I_{2 \times 2}$
25		10 $I_{4 \times 4}$	$I_{2 \times 2}$
26			5 $I_{2 \times 2}$
27			10 $I_{2 \times 2}$

Note: $I_{4 \times 4}$ =4 by 4 identity, and $I_{2 \times 2}$ =2 by 2 identity.

for each combination of weighting matrices and each traffic scenario.

Since normal operation and the real-time control strategy are the same as those in Chapter 8, the queue lengths at the end of the real-time control for transition 1 obtained in this chapter are also the same as those in Chapter 8. Therefore, the average queue lengths at the end of transition 1 under the real-time control described in Chapter 8 will be used in this chapter as initial states for transition 2 under the optimal control. As noted in the previous chapters, the optimal control begins immediately after the real-time control. The average queue lengths are listed in Table 9.2 (See also Table 8.7).

Table 9.2 Average queue lengths after transition 1

Scenario #	r_1	r_2	Average queue (veh) on approach			
			1	2	3	4
1	0.3	0.1	5.1	1.33	3.1	1.09
2	0.4	0.2	8.5	3.31	4.3	2.49
3	0.5	0.3	12.6	5.11	5.9	3.78
4	0.6	0.4	50.0	13.47	3.1	4.93

Notes: (1) r_1 = ratio of traffic volume to saturation flow for approach 1.

(2) r_2 = ratio of traffic volume to saturation flow for approach 2.

(3) Approaches 1 & 3 are critical approaches, and traffic volumes on their opposite approaches 2 & 4 are fixed at 0.8 times volumes on corresponding critical approaches.

Simulation results under the optimal control for transition 2 are summarized in Tables 9.3 to 9.6. The recovery time is not listed in these tables because the change in weighting matrices will only affect the distribution of green time between the EV road and the cross road and the recovery time will not be affected. The simulation results are also illustrated in Figures 9.1 to 9.8.

Table 9.3 Average queue lengths (veh) after transition 2 for traffic scenario 1
(simulation 1)

Weighting matrix combination #		Queue on approach				Average queue over all approaches
		1	2	3	4	
1	mean	0.27	0.50	0.00	0.02	0.20
	std	0.67	0.55	0.00	0.06	
2	mean	0.00	0.57	0.00	0.00	0.14
	std	0.00	0.46	0.00	0.00	
3	mean	0.00	0.49	0.00	0.00	0.12
	std	0.00	0.41	0.00	0.00	
4	mean	0.02	0.00	0.00	0.00	0.01
	std	0.05	0.00	0.00	0.00	
5	mean	0.03	0.00	0.00	0.00	0.01
	std	0.07	0.00	0.00	0.00	
6	mean	0.13	0.00	0.00	0.00	0.03
	std	0.31	0.00	0.00	0.00	
7	mean	0.55	0.00	0.00	0.00	0.14
	std	0.91	0.00	0.00	0.00	
8	mean	0.58	0.08	0.00	0.00	0.17
	std	0.93	0.20	0.00	0.00	
9	mean	0.57	0.06	0.00	0.00	0.16
	std	0.82	0.15	0.00	0.00	
10	mean	1.07	0.21	0.14	0.00	0.35
	std	1.22	0.51	0.33	0.00	
11	mean	0.00	0.87	0.00	0.03	0.23
	std	0.00	0.68	0.00	0.07	
12	mean	0.00	0.87	0.00	0.02	0.22
	std	0.00	0.67	0.00	0.06	
13	mean	0.00	0.31	0.00	0.00	0.08
	std	0.00	0.34	0.00	0.00	
14	mean	0.00	0.30	0.00	0.00	0.08
	std	0.00	0.33	0.00	0.00	
15	mean	0.00	0.30	0.00	0.00	0.07
	std	0.00	0.33	0.00	0.00	
16	mean	0.00	0.07	0.00	0.00	0.02
	std	0.00	0.12	0.00	0.00	
17	mean	0.00	0.06	0.00	0.00	0.02
	std	0.00	0.10	0.00	0.00	
18	mean	0.00	0.06	0.00	0.00	0.01
	std	0.00	0.09	0.00	0.00	

(continued on next page)

19	mean	0.14	0.76	0.00	0.16	0.27
	std	0.35	0.50	0.00	0.22	
20	mean	0.00	0.91	0.00	0.24	0.29
	std	0.00	0.55	0.00	0.38	
21	mean	0.00	1.03	0.00	0.17	0.30
	std	0.00	0.60	0.00	0.33	
22	mean	0.56	0.15	0.07	0.00	0.19
	std	0.71	0.38	0.16	0.00	
23	mean	0.67	0.15	0.12	0.02	0.24
	std	0.87	0.37	0.30	0.04	
24	mean	0.66	0.16	0.12	0.00	0.23
	std	0.87	0.38	0.30	0.00	
25	mean	0.00	0.34	0.00	0.00	0.08
	std	0.00	0.29	0.00	0.00	
26	mean	0.00	0.33	0.00	0.00	0.08
	std	0.00	0.29	0.00	0.00	
27	mean	0.00	0.33	0.00	0.00	0.08
	std	0.00	0.28	0.00	0.00	

Notes: 1. queue lengths are measured at the end of each transitioning.

2. mean = mean of queue length.

3. std = standard deviation of queue length.

Table 9.4 Average queue lengths (veh) after transition 2 for traffic scenario 2
(simulation 1)

Weighting matrix combination #		Queue on approach				Average queue over all approaches
		1	2	3	4	
1	mean	3.22	1.90	0.42	0.53	1.52
	std	2.02	1.06	0.73	0.55	
2	mean	3.67	2.69	0.75	0.85	1.99
	std	1.97	1.42	1.22	0.47	
3	mean	3.61	2.74	0.69	0.89	1.98
	std	2.02	1.44	1.22	0.54	
4	mean	2.03	1.07	0.60	0.18	0.97
	std	1.34	1.29	0.87	0.45	
5	mean	2.18	1.05	0.79	0.19	1.05
	std	1.62	1.38	1.00	0.47	
6	mean	2.31	1.37	0.47	0.30	1.11
	std	1.39	1.38	0.84	0.48	

(continued on next page)

7	mean	2.30	0.67	0.60	0.27	0.96
	std	1.89	0.72	1.13	0.65	
8	mean	2.25	0.69	0.59	0.27	0.95
	std	1.85	0.75	1.11	0.65	
9	mean	2.43	0.74	0.58	0.00	0.94
	std	1.78	1.05	1.10	0.00	
10	mean	1.60	2.52	0.12	0.47	1.18
	std	1.21	1.28	0.30	0.63	
11	mean	3.26	3.09	0.56	1.18	2.02
	std	2.03	1.39	1.07	0.67	
12	mean	3.26	3.09	0.55	1.18	2.02
	std	2.05	1.40	1.07	0.68	
13	mean	1.40	2.05	0.13	0.05	0.91
	std	1.16	1.23	0.31	0.11	
14	mean	1.38	2.06	0.13	0.05	0.90
	std	1.17	1.23	0.31	0.11	
15	mean	2.74	2.29	0.52	0.31	1.47
	std	1.58	1.41	0.66	0.41	
16	mean	1.64	1.34	0.65	0.23	0.96
	std	1.22	1.22	1.11	0.35	
17	mean	1.90	1.33	0.64	0.18	1.01
	std	1.10	1.13	1.09	0.29	
18	mean	1.89	1.33	0.61	0.14	0.99
	std	1.07	1.14	1.05	0.22	
19	mean	0.70	2.37	0.09	0.19	0.84
	std	0.82	1.21	0.22	0.21	
20	mean	1.39	2.57	0.10	0.21	1.07
	std	1.67	1.56	0.24	0.23	
21	mean	3.19	3.17	0.53	1.25	2.04
	std	2.01	1.39	1.04	0.71	
22	mean	0.48	2.45	0.07	0.25	0.81
	std	0.81	1.06	0.18	0.22	
23	mean	0.78	2.37	0.07	0.22	0.86
	std	0.92	1.19	0.18	0.20	
24	mean	1.08	2.29	0.10	0.17	0.91
	std	1.22	1.17	0.24	0.19	
25	mean	0.87	2.52	0.05	0.43	0.97
	std	1.10	1.12	0.13	0.28	

(continued on next page)

26	mean	0.87	2.51	0.06	0.41	0.96
	std	1.08	1.13	0.13	0.27	
27	mean	0.88	2.48	0.06	0.37	0.95
	std	1.08	1.17	0.14	0.26	

Notes: 1. queue lengths are measured at the end of each transitioning.
 2. mean = mean of queue length.
 3. std = standard deviation of queue length.

Table 9.5 Average queue lengths (veh) after transition 2 for traffic scenario 3
(simulation 1)

Weighting matrix combination #		Queue on approach				Average queue over all approaches
		1	2	3	4	
1	mean	8.79	6.57	1.41	4.11	5.22
	std	0.84	1.33	1.44	1.42	
2	mean	9.09	6.27	1.64	3.80	5.20
	std	0.82	1.37	1.50	1.50	
3	mean	9.02	6.42	1.81	3.96	5.30
	std	1.10	0.66	1.55	1.39	
4	mean	10.94	5.39	3.59	3.09	5.75
	std	3.83	3.52	3.05	3.69	
5	mean	6.99	8.37	0.44	5.91	5.43
	std	0.82	1.37	0.68	1.50	
6	mean	7.70	7.65	0.70	5.19	5.31
	std	0.83	1.41	1.00	1.56	
7	mean	10.74	4.72	2.81	2.42	5.17
	std	0.96	1.58	1.86	1.79	
8	mean	10.73	4.74	2.78	2.45	5.17
	std	0.96	1.58	1.86	1.79	
9	mean	12.81	4.06	4.81	1.79	5.87
	std	0.95	1.57	1.86	1.81	
10	mean	9.30	6.06	1.84	3.60	5.20
	std	0.85	1.33	1.55	1.40	
11	mean	9.35	6.01	1.87	3.55	5.19
	std	0.84	1.33	1.56	1.42	
12	mean	9.02	6.37	1.91	3.91	5.30
	std	1.31	0.59	1.57	1.32	
13	mean	8.70	6.66	1.32	4.20	5.22
	std	0.85	1.33	1.44	1.40	

(continued on next page)

14	mean	8.79	6.57	1.41	4.11	5.22
	std	0.84	1.33	1.44	1.42	
15	mean	8.88	6.48	1.48	4.01	5.21
	std	0.83	1.34	1.45	1.45	
16	mean	7.95	7.41	0.84	4.94	5.29
	std	0.86	1.33	1.18	1.40	
17	mean	8.09	7.27	0.91	4.81	5.27
	std	0.84	1.33	1.23	1.42	
18	mean	8.23	7.13	1.02	4.66	5.26
	std	0.83	1.34	1.25	1.45	
19	mean	9.37	5.99	1.90	3.53	5.20
	std	0.85	1.33	1.57	1.40	
20	mean	9.39	5.97	1.91	3.51	5.19
	std	0.85	1.33	1.57	1.41	
21	mean	9.02	6.36	1.93	3.90	5.30
	std	1.37	0.59	1.58	1.32	
22	mean	9.07	6.29	1.64	3.83	5.21
	std	0.86	1.33	1.50	1.40	
23	mean	9.10	6.26	1.67	3.79	5.20
	std	0.85	1.33	1.51	1.41	
24	mean	9.14	6.22	1.70	3.76	5.20
	std	0.84	1.33	1.51	1.42	
25	mean	8.69	6.67	1.31	4.21	5.22
	std	0.85	1.33	1.44	1.40	
26	mean	8.74	6.63	1.37	4.16	5.22
	std	0.85	1.33	1.43	1.41	
27	mean	8.79	6.57	1.41	4.11	5.22
	std	0.84	1.33	1.44	1.42	

Notes: 1. queue lengths are measured at the end of each transitioning.

2. mean = mean of queue length.

3. std = standard deviation of queue length.

Table 9.6 Average queue lengths (veh) after transition 2 for traffic scenario 4
(simulation 1)

Weighting matrix combination #		Queue on approach				Average queue over all approaches
		1	2	3	4	
1	mean	48.34	14.18	1.67	6.57	17.69
	std	1.89	2.58	1.67	1.55	
2	mean	48.04	14.94	1.54	7.34	17.97
	std	1.99	2.33	1.59	1.22	
3	mean	47.71	16.20	1.24	8.59	18.44
	std	1.88	2.46	1.46	1.55	
4	mean	46.54	15.98	0.56	8.37	17.86
	std	1.89	2.58	1.10	1.55	
5	mean	47.12	15.40	0.87	7.80	17.80
	std	1.88	2.40	1.25	1.56	
6	mean	47.38	15.14	1.07	7.53	17.78
	std	1.89	2.32	1.30	1.58	
7	mean	48.93	14.51	3.11	7.02	18.39
	std	3.75	1.29	2.73	3.09	
8	mean	45.85	16.66	0.33	9.05	17.98
	std	1.88	2.40	0.82	1.56	
9	mean	46.56	15.95	0.57	8.35	17.86
	std	1.89	2.32	1.06	1.58	
10	mean	48.85	13.67	2.07	6.06	17.66
	std	1.90	2.66	1.81	1.57	
11	mean	48.65	14.56	1.85	6.95	18.00
	std	2.15	2.18	1.85	1.52	
12	mean	48.16	15.75	1.42	8.14	18.37
	std	1.98	2.88	1.76	1.63	
13	mean	48.42	14.09	1.71	6.49	17.68
	std	1.90	2.66	1.72	1.56	
14	mean	48.34	14.18	1.67	6.57	17.69
	std	1.89	2.58	1.67	1.55	
15	mean	48.26	14.26	1.63	6.65	17.70
	std	1.88	2.51	1.63	1.55	
16	mean	47.89	14.62	1.31	7.02	17.71
	std	1.90	2.66	1.59	1.56	
17	mean	47.89	14.63	1.33	7.03	17.72
	std	1.89	2.58	1.56	1.55	

(continued on next page)

18	mean	47.88	14.64	1.34	7.03	17.72
	std	1.88	2.51	1.53	1.55	
19	mean	48.93	13.59	2.13	5.98	17.66
	std	1.90	2.67	1.83	1.57	
20	mean	48.81	14.40	1.93	6.80	17.99
	std	2.19	2.26	1.92	1.53	
21	mean	48.31	15.59	1.48	7.99	18.34
	std	2.05	3.04	1.87	1.71	
22	mean	47.56	14.78	1.95	6.79	17.77
	std	1.97	2.39	1.79	1.32	
23	mean	48.64	13.88	1.90	6.27	17.67
	std	1.90	2.63	1.75	1.56	
24	mean	48.56	13.96	1.86	6.35	17.68
	std	1.89	2.58	1.72	1.55	
25	mean	47.30	15.03	1.72	7.05	17.77
	std	1.95	2.37	1.73	1.31	
26	mean	48.39	14.13	1.70	6.52	17.68
	std	1.89	2.63	1.70	1.56	
27	mean	48.34	14.18	1.67	6.57	17.69
	std	1.89	2.58	1.67	1.55	

Notes: 1. queue lengths are measured at the end of each transitioning.
2. mean = mean of queue length.
3. std = standard deviation of queue length.

Fig. 9.1 Average queues on approaches 1 to 4 for traffic scenario 1 (simulation 1)

Fig. 9.2 Average queues over all approaches for traffic scenario 1 (simulation 1)

Fig. 9.3 Average queues on approaches 1 to 4 for traffic scenario 2 (simulation 1)

Fig. 9.4 Average queues over all approaches for traffic scenario 2 (simulation 1)

Fig. 9.5 Average queues on approaches 1 to 4 for traffic scenario 3 (simulation 1)

Fig. 9.6 Average queues over all approaches for traffic scenario 3 (simulation 1)

Fig. 9.7 Average queues on approaches 1 to 4 for traffic scenario 4 (simulation 1)

Fig. 9.8 Average queues over all approaches for traffic scenario 4 (simulation 1)

9.2 Analysis of simulation results

For ease of presentation in the following sections, q_{mean} is used to represent the average queue length over all the approaches to the subject intersection.

1. Traffic scenario 1

Simulation results are summarized in Table 9.3 and illustrated in Figures 9.1 and 9.2. As shown in Figure 9.2, for weighting matrix combinations with $P' = 1$, q_{mean} was minimized when $R = 5I$ (or $R = 5P'$), which means that R was more important than P' in the control algorithm. For combinations with $P' = 5I$, minimal q_{mean} appeared when $R = 10I$ (or $R = 2P'$), which suggests again that a higher R relative to P' could benefit the optimal control algorithm. For combinations with $P' = 10I$, minimal q_{mean} appeared when $R = 10I$ (or $R = P'$), which indicates that P' should be less than or equal to R .

A trend is observed that the higher the ratio of R to P' , the lower the q_{mean} . The reason behind it probably is that the final queues were very short (the longest one was 1.07 veh. On approach 1 for weighting matrix combination 10), and therefore their corresponding weighting matrix P' had less impact on reducing q_{mean} than the weighting matrix R for the intermediate queue lengths.

2. Traffic scenario 2

Simulation results are summarized in Table 9.4 and illustrated in Figures 9.3 and 9.4. As shown in Figure 9.4, all longest queues appeared when $R = I$, especially when $S > R$ (combinations 2, 3, 11, 12, and 21). Another rapid increase in q_{mean} was observed for combination 15 with $P' = 5I$, $R = 5I$, and $S = 10I$. q_{mean} became more stable with increase of R . The simulation results suggest that a large R makes q_{mean} short and stable.

and R is more important in reducing q_{mean} than weighting matrix S for control variables.

3. Traffic scenario 3

Simulation results are summarized in Table 9.5 and illustrated in Figures 9.5 and 9.6. In Figure 9.6, all rapid changes in q_{mean} happened for combinations with $P^* = I$. When P^* increased to $5I$ and $10I$, q_{mean} became stable for all corresponding weighting matrix combinations. It should be noted that for combinations 12 ($P^* = 5I$, $R = I$, and $S = 10I$) and 21 ($P^* = 10I$, $R = I$, and $S = 10I$), q_{mean} were not so stable as for other combinations. It is suggested that P^* should not be too small relative to R and S ; large P^* and R make q_{mean} short and stable; and S should not be greater than R for stable behaviour of the optimal control algorithm even if P^* and R are large.

4. Traffic scenario 4

Simulation results are summarized in Table 9.6 and illustrated in Figures 9.7 and 9.8. In Figure 9.8, for combinations with $R = I$ (combinations 1, 2, 3, 10, 11, 12, 19, 20, and 21) q_{mean} increased with S , and q_{mean} was minimized when $S = I$. It is suggested that R and S have the same order in affecting the efficiency of the optimal control algorithm. When R increased to $5I$ (combinations 4, 5, 6, 13, 14, 15, 22, 23, and 24) q_{mean} was not sensitive to P^* and S , and q_{mean} was short and stable for all corresponding combinations. For combinations with $R = 10I$, q_{mean} was short and stable when $P^* = 5I$ and $P^* = 10I$ (combinations 16, 17, 18, 25, 26, and 27), and unstable when $P^* = I$ (combinations 7, 8, and 9). For combinations 7, 8, and 9, q_{mean} decreased with S , and minimal q_{mean} appeared for combination 9 with $P^* = I$, $R = 10I$, and $S = 10I$. It is suggested again that R and S have the same order of importance in affecting control efficiency.

In summary, for stable behaviour of the optimal control algorithm and short q_{mean} ,

weighting matrix combinations should be selected as follows.

- (1) Weighing matrix S for control variables should be less than or equal to weighting matrix P' for final states and weighting matrix R for intermediate states.
- (2) When traffic volumes are very light at an intersection, R should be greater than P' .

9.3 Another simulation of optimal control strategy

In the last section, the simulation results indicate that no great differences exist in q_{mean} for different weighting matrix combinations. The maximum differences in q_{mean} were

$$0.35 - 0.01 = 0.34 \text{ (veh) for traffic scenario 1,}$$

$$2.04 - 0.81 = 1.23 \text{ (veh) for traffic scenario 2,}$$

$$5.87 - 5.17 = 0.70 \text{ (veh) for traffic scenario 3, and}$$

$$18.44 - 17.66 = 0.78 \text{ (veh) for traffic scenario 4, respectively.}$$

In this section another simulation is conducted in order to investigate whether greater differences in weighting matrices will cause rapid changes in simulation results.

The combinations of weighting matrices for the second simulation are listed in Table 9.7. The simulation results are summarized in Tables 9.8 to 9.11 and illustrated in Figures 9.9 to 9.16.

Table 9.7 Combinations of weighting matrices for the second simulation

Combination #	P	R	S
1	$I_{4 \times 4}$	$I_{4 \times 4}$	$I_{2 \times 2}$
2			50 $I_{2 \times 2}$
3			100 $I_{2 \times 2}$
4		50 $I_{4 \times 4}$	$I_{2 \times 2}$
5			50 $I_{2 \times 2}$
6			100 $I_{2 \times 2}$
7		100 $I_{4 \times 4}$	$I_{2 \times 2}$
8			50 $I_{2 \times 2}$
9			100 $I_{2 \times 2}$
10	50 $I_{4 \times 4}$	$I_{4 \times 4}$	$I_{2 \times 2}$
11			50 $I_{2 \times 2}$
12			100 $I_{2 \times 2}$
13		50 $I_{4 \times 4}$	$I_{2 \times 2}$
14			50 $I_{2 \times 2}$
15			100 $I_{2 \times 2}$
16		100 $I_{4 \times 4}$	$I_{2 \times 2}$
17			50 $I_{2 \times 2}$
18			100 $I_{2 \times 2}$
19	100 $I_{4 \times 4}$	$I_{4 \times 4}$	$I_{2 \times 2}$
20			50 $I_{2 \times 2}$
21			100 $I_{2 \times 2}$
22		50 $I_{4 \times 4}$	$I_{2 \times 2}$
23			50 $I_{2 \times 2}$
24			100 $I_{2 \times 2}$
25		100 $I_{4 \times 4}$	$I_{2 \times 2}$
26			50 $I_{2 \times 2}$
27			100 $I_{2 \times 2}$

Note: $I_{4 \times 4}$ =4 by 4 identity, and $I_{2 \times 2}$ =2 by 2 identity.

Table 9.8 Average queue lengths (veh) after transition 2 for traffic scenario 1
(simulation 2)

Weighting matrix combination #		Queue on approach				Average queue over all approaches
		1	2	3	4	
1	mean	0.00	0.62	0.00	0.29	0.23
	std	0.00	0.47	0.00	0.53	
2	mean	0.00	0.07	0.00	0.07	0.03
	std	0.00	0.17	0.00	0.17	
3	mean	0.00	0.04	0.00	0.03	0.02
	std	0.00	0.10	0.00	0.07	
4	mean	0.20	0.00	0.05	0.00	0.06
	std	0.26	0.00	0.13	0.00	
5	mean	0.23	0.00	0.00	0.00	0.06
	std	0.26	0.00	0.00	0.00	
6	mean	0.21	0.11	0.00	0.00	0.08
	std	0.34	0.26	0.00	0.00	
7	mean	0.21	0.00	0.06	0.00	0.07
	std	0.27	0.00	0.13	0.00	
8	mean	0.12	0.00	0.00	0.00	0.03
	std	0.22	0.00	0.00	0.00	
9	mean	0.38	0.00	0.14	0.00	0.13
	std	0.49	0.00	0.33	0.00	
10	mean	0.00	0.90	0.00	0.13	0.26
	std	0.00	0.53	0.00	0.24	
11	mean	0.00	1.29	0.00	0.21	0.37
	std	0.00	0.54	0.00	0.37	
12	mean	0.00	1.44	0.00	0.25	0.42
	std	0.00	0.62	0.00	0.39	
13	mean	0.07	0.27	0.00	0.00	0.08
	std	0.18	0.30	0.00	0.00	
14	mean	0.11	0.28	0.04	0.00	0.11
	std	0.27	0.28	0.09	0.00	
15	mean	0.11	0.26	0.03	0.00	0.10
	std	0.27	0.29	0.09	0.00	
16	mean	0.20	0.09	0.13	0.00	0.10
	std	0.49	0.14	0.31	0.00	
17	mean	0.23	0.08	0.15	0.00	0.12
	std	0.55	0.13	0.37	0.00	
18	mean	0.00	0.13	0.00	0.15	0.07
	std	0.00	0.14	0.00	0.38	

(continued on next page)

19	mean	0.00	0.91	0.00	0.13	0.26
	std	0.00	0.54	0.00	0.24	
20	mean	0.00	1.32	0.00	0.21	0.38
	std	0.00	0.55	0.00	0.38	
21	mean	0.00	1.48	0.00	0.26	0.44
	std	0.00	0.62	0.00	0.41	
22	mean	0.54	0.03	0.08	0.14	0.20
	std	0.67	0.09	0.20	0.33	
23	mean	0.70	0.04	0.21	0.18	0.28
	std	0.84	0.07	0.34	0.38	
24	mean	0.70	0.06	0.21	0.17	0.29
	std	0.84	0.09	0.35	0.37	
25	mean	0.07	0.27	0.00	0.00	0.08
	std	0.17	0.30	0.00	0.00	
26	mean	0.11	0.28	0.04	0.00	0.11
	std	0.28	0.28	0.10	0.00	
27	mean	0.11	0.28	0.04	0.00	0.11
	std	0.27	0.28	0.09	0.00	

- Notes: 1. queue lengths are measured at the end of each transitioning.
 2. mean = mean of queue length.
 3. std = standard deviation of queue length.

Table 9.9 Average queue lengths (veh) after transition 2 for traffic scenario 2 (simulation 2)

Weighting matrix combination #		Queue on approach				Average queue over all approaches
		1	2	3	4	
1	mean	2.86	1.68	0.46	1.16	1.54
	std	1.22	1.35	0.77	1.19	
2	mean	3.86	1.51	0.20	1.11	1.67
	std	1.68	1.36	0.41	1.42	
3	mean	3.98	1.46	0.30	1.02	1.69
	std	1.57	1.33	0.52	1.39	
4	mean	3.78	0.52	0.59	0.00	1.22
	std	2.19	1.01	0.65	0.00	
5	mean	3.02	0.32	0.25	0.16	0.94
	std	2.18	0.44	0.39	0.32	
6	mean	2.88	0.17	0.15	0.12	0.83
	std	1.88	0.42	0.18	0.20	

(continued on next page)

7	mean	4.36	0.06	0.48	0.38	1.32
	std	2.13	0.16	0.55	0.92	
8	mean	3.50	0.28	0.19	0.08	1.02
	std	1.88	0.44	0.25	0.20	
9	mean	2.57	0.52	0.08	0.35	0.88
	std	2.01	0.58	0.13	0.60	
10	mean	0.30	1.31	0.00	0.49	0.53
	std	0.47	1.23	0.00	0.62	
11	mean	3.13	2.21	0.00	1.79	1.78
	std	1.84	1.42	0.00	1.27	
12	mean	3.10	2.18	0.00	1.73	1.75
	std	1.82	1.42	0.00	1.32	
13	mean	0.94	1.55	0.00	0.70	0.80
	std	1.18	1.38	0.00	0.79	
14	mean	1.17	1.54	0.00	0.90	0.90
	std	1.05	1.26	0.00	1.31	
15	mean	1.25	1.38	0.00	0.85	0.87
	std	1.09	1.23	0.00	1.26	
16	mean	1.77	1.41	0.16	0.66	1.00
	std	1.06	1.51	0.29	0.56	
17	mean	1.74	1.34	0.09	0.67	0.96
	std	1.02	1.49	0.21	0.59	
18	mean	1.54	1.48	0.08	0.66	0.94
	std	1.13	1.48	0.19	0.62	
19	mean	0.31	1.29	0.00	0.47	0.52
	std	0.48	1.24	0.00	0.60	
20	mean	3.11	2.23	0.00	1.81	1.79
	std	1.83	1.42	0.00	1.27	
21	mean	3.07	2.21	0.00	1.76	1.76
	std	1.82	1.42	0.00	1.31	
22	mean	0.19	1.44	0.00	0.65	0.57
	std	0.47	1.27	0.00	1.03	
23	mean	0.70	1.38	0.00	0.78	0.71
	std	0.70	1.30	0.00	1.02	
24	mean	1.28	1.34	0.00	0.82	0.86
	std	1.13	1.32	0.00	1.14	
25	mean	0.94	1.56	0.00	0.70	0.80
	std	1.19	1.38	0.00	0.79	

(continued on next page)

26	mean	0.81	1.62	0.00	0.94	0.84
	std	0.57	1.32	0.00	1.31	
27	mean	1.18	1.54	0.00	0.90	0.90
	std	1.05	1.26	0.00	1.31	

- Notes: 1. queue lengths are measured at the end of each transitioning.
 2. mean = mean of queue length.
 3. std = standard deviation of queue length.

Table 9.10 Average queue lengths (veh) after transition 2 for traffic scenario 3
(simulation 2)

Weighting matrix combination #		Queue on approach				Average queue over all approaches
		1	2	3	4	
1	mean	10.45	6.24	0.16	3.51	5.09
	std	1.63	2.05	0.33	1.39	
2	mean	10.84	7.24	0.30	4.37	5.69
	std	1.61	2.16	0.63	1.66	
3	mean	11.16	6.92	0.46	4.07	5.65
	std	1.61	2.19	0.77	1.59	
4	mean	12.18	4.77	0.88	2.42	5.06
	std	1.62	2.22	0.89	1.33	
5	mean	12.06	4.89	0.74	2.57	5.06
	std	1.62	2.23	0.86	1.33	
6	mean	8.20	8.95	0.00	6.22	5.84
	std	1.27	2.22	0.00	1.33	
7	mean	12.18	4.79	0.86	2.45	5.07
	std	1.62	2.23	0.89	1.33	
8	mean	12.12	4.85	0.78	2.53	5.07
	std	1.62	2.23	0.87	1.33	
9	mean	12.05	4.92	0.71	2.61	5.07
	std	1.62	2.23	0.85	1.33	
10	mean	9.13	6.07	0.34	2.13	4.42
	std	1.59	1.82	0.43	1.10	
11	mean	9.30	8.78	0.00	5.82	5.97
	std	1.72	1.94	0.00	2.09	
12	mean	9.47	8.61	0.00	5.66	5.93
	std	1.70	1.95	0.00	2.04	
13	mean	7.75	7.45	0.00	3.91	4.78
	std	1.68	1.54	0.00	1.54	
14	mean	9.49	6.61	0.02	3.55	4.92
	std	1.46	1.94	0.06	1.41	

(continued on next page)

15	mean	10.53	6.16	0.19	3.43	5.08
	std	1.62	2.06	0.38	1.39	
16	mean	9.59	7.10	0.00	4.38	5.27
	std	1.63	2.04	0.00	1.41	
17	mean	9.75	6.94	0.02	4.21	5.23
	std	1.63	2.05	0.05	1.40	
18	mean	9.89	6.80	0.05	4.07	5.20
	std	1.62	2.07	0.12	1.39	
19	mean	8.79	6.09	0.36	1.99	4.31
	std	1.62	2.08	0.49	1.16	
20	mean	9.21	8.87	0.00	5.90	5.99
	std	1.73	1.93	0.00	2.12	
21	mean	9.31	8.77	0.00	5.81	5.97
	std	1.72	1.94	0.00	2.09	
22	mean	7.90	6.95	0.00	2.97	4.45
	std	1.61	2.10	0.00	1.40	
23	mean	8.50	6.66	0.04	2.87	4.52
	std	1.62	1.71	0.09	1.26	
24	mean	9.48	6.29	0.07	2.90	4.69
	std	0.72	1.96	0.17	1.21	
25	mean	7.76	7.43	0.00	3.89	4.77
	std	1.68	1.55	0.00	1.53	
26	mean	8.37	7.12	0.00	3.71	4.80
	std	0.89	1.85	0.00	1.49	
27	mean	9.50	6.60	0.02	3.54	4.92
	std	1.45	1.94	0.06	1.40	

- Notes: 1. queue lengths are measured at the end of each transitioning.
2. mean = mean of queue length.
3. std = standard deviation of queue length.

Table 9.11 Average queue lengths (veh) after transition 2 for traffic scenario 4
(simulation 2)

Weighting matrix combination #		Queue on approach				Average queue over all approaches
		1	2	3	4	
1	mean	48.29	14.83	0.40	5.33	17.21
	std	2.30	1.65	0.70	1.57	
2	mean	47.41	17.10	0.13	7.59	18.06
	std	2.07	1.38	0.32	1.67	
3	mean	54.15	12.90	5.07	4.02	19.03
	std	2.04	1.35	2.11	1.71	
4	mean	48.21	14.90	0.30	5.89	17.33
	std	2.04	1.35	0.44	1.71	
5	mean	45.39	17.73	0.00	8.22	17.83
	std	2.05	1.36	0.00	1.70	
6	mean	46.35	16.77	0.00	7.26	17.60
	std	2.05	1.36	0.00	1.71	
7	mean	48.21	14.90	0.25	5.96	17.33
	std	2.04	1.35	0.43	1.72	
8	mean	50.26	14.22	1.56	5.47	17.88
	std	2.04	1.35	1.19	1.72	
9	mean	45.25	17.88	0.00	8.37	17.87
	std	2.04	1.35	0.00	1.71	
10	mean	44.23	18.29	0.41	3.97	16.73
	std	2.15	1.43	0.71	1.58	
11	mean	48.75	15.76	0.74	6.25	17.88
	std	2.74	2.16	0.91	1.74	
12	mean	48.52	15.98	0.58	6.48	17.89
	std	2.58	1.98	0.79	1.65	
13	mean	42.81	19.30	0.00	6.41	17.13
	std	2.09	1.38	0.00	1.57	
14	mean	43.25	18.97	0.00	5.69	16.98
	std	2.09	1.39	0.00	1.57	
15	mean	44.45	18.76	0.14	5.48	17.21
	std	1.66	1.41	0.23	1.37	
16	mean	45.20	17.31	0.30	6.75	17.39
	std	5.05	3.43	0.55	0.77	
17	mean	44.23	18.13	0.26	6.88	17.37
	std	3.79	3.46	0.50	1.13	

(continued on next page)

18	mean	44.29	18.13	0.03	6.57	17.26
	std	2.43	2.22	0.08	1.66	
19	mean	44.38	18.19	0.60	3.65	16.70
	std	2.15	1.43	0.82	1.58	
20	mean	48.89	15.62	0.83	6.11	17.86
	std	2.81	2.28	0.99	1.81	
21	mean	48.76	15.75	0.75	6.24	17.87
	std	2.74	2.16	0.91	1.74	
22	mean	43.48	18.82	0.08	5.06	16.86
	std	2.12	1.40	0.19	1.57	
23	mean	43.76	18.61	0.15	4.67	16.80
	std	2.12	1.41	0.37	1.57	
24	mean	44.59	18.50	0.36	4.65	17.03
	std	1.76	1.46	0.49	1.28	
25	mean	42.84	19.28	0.00	6.32	17.11
	std	2.09	1.38	0.00	1.57	
26	mean	43.06	19.11	0.00	5.94	17.03
	std	2.09	1.39	0.00	1.57	
27	mean	43.29	18.93	0.00	5.58	16.95
	std	2.10	1.39	0.00	1.57	

Notes: 1. queue lengths are measured at the end of each transitioning.
 2. mean = mean of queue length.
 3. std = standard deviation of queue length.

Fig. 9.9 Average queues on approaches 1 to 4 for traffic scenario 1 (simulation 2)

Fig. 9.10 Average queues over all approaches for traffic scenario 1 (simulation 2)

Fig. 9.11 Average queues on approaches 1 to 4 for traffic scenario 2 (simulation 2)

Fig. 9.12 Average queues over all approaches for traffic scenario 2 (simulation 2)

Fig. 9.13 Average queues on approaches 1 to 4 for traffic scenario 3 (simulation 2)

Fig. 9.14 Average queues over all approaches for traffic scenario 3 (simulation 2)

Fig. 9.15 Average queues on approaches 1 to 4 for traffic scenario 4 (simulation 2)

Fig. 9.16 Average queues over all approaches for traffic scenario 4 (simulation 2)

As shown in Figures 9.10, 9.12, 9.14, and 9.16, the changes in q_{mean} had the same tendency as in Figures 9.2, 9.4, 9.6, and 9.8, respectively. The simulation results in this section confirm the conclusions obtained in the last section. That is:

- (1) large weighting matrices P' and R relative to weighting matrix S will benefit the stability of the optimal control strategy and the reduction in q_{mean} ; and
- (2) for light traffic situations, R for intermediate states is more important than P' for final states in terms of control stability and efficiency.

In the second simulation, the maximum differences in q_{mean} for different weighting matrix combinations had the following values:

$$0.44 - 0.02 = 0.42 \text{ (veh) for traffic scenario 1}$$

$$1.79 - 0.52 = 1.27 \text{ (veh) for traffic scenario 2}$$

$$5.99 - 4.31 = 1.68 \text{ (veh) for traffic scenario 3, and}$$

$$19.03 - 16.70 = 2.33 \text{ (veh) for traffic scenario 4, respectively.}$$

Compared with corresponding values of maximum differences in q_{mean} found in simulation 1, i.e., 0.34, 1.23, 0.70, 0.78 (veh) for scenarios 1, 2, 3, and 4, respectively, there were obvious changes for scenarios 3 and 4 with heavier traffic due to the increases in difference of weighting matrices P' , R , and S .

Looking at the details, the minimum average queues in the second simulation were 0.02 (combination 3), 0.52 (combination 19), 4.31(combination 19), and 16.70 (combination 19) for traffic scenarios 1, 2, 3, and 4, respectively. Compared with corresponding values in the first simulation, i.e., 0.01 (combination 4), 0.81 (combination 22), 5.17 (combination 7), and 17.66 (combination 10) for traffic

scenarios 1, 2, 3, and 4, respectively, all scenarios except scenario 1 in the second simulation benefited from the increases in differences of weighting matrices P' , R , and S . The gains are summarized in Table 9.12.

Table 9.12 Changes in minimum average queues due to changes of P' , R , and S

Traffic scenario #	$q_{\text{mean-1}}$ (veh)	$q_{\text{mean-2}}$ (veh)	$\Delta = q_{\text{mean-1}} - q_{\text{mean-2}}$ (veh)	$\delta = \Delta / q_{\text{mean-1}}$ (%)
1	0.01	0.02	-0.01	-100
2	0.81	0.52	0.29	35.8
3	5.17	4.31	0.86	16.6
4	17.66	16.70	0.96	5.4

Notes: $q_{\text{mean-1}}$ = minimum average queue for the first simulation; and
 $q_{\text{mean-2}}$ = minimum average queue for the second simulation;
 Δ = absolute change in minimum average queues from the first simulation to the second simulation; and
 δ = relative change in minimum average queues from the first simulation to the second simulation.

As shown in Table 9.12, the absolute changes in q_{mean} were small for light traffic scenarios 1 and 2, although their relative changes are obvious (i.e., -100% and 35.8%). For heavier traffic scenarios 3 and 4, both absolute changes and relative changes in q_{mean} were not so rapid as weighting matrices P' , R , and S . It means that the shortest average queue under the optimal control algorithm was not sensitive to weighting matrices P' , R , and S , and high differences in P' , R , and S did not cause significant improvement of control efficiency. This analysis leads to the following possibility:

For a certain traffic situation at an intersection, it is possible to search a small space of weighting matrix combinations for an optimal combination so that a shortest average queue length can be obtained under the optimal control algorithm.

Chapter 10 Control Efficiencies for Multiple EVSP Occurrences

If two successive EVs request signal preemptions with a long time interval between such requests, they can be treated as two separate signal preemption occurrences, and control strategies discussed in previous chapters are still applicable to this situation. However, if multiple EVSPs occur within a short timed interval, there exists the possibility that another EV can approach an intersection before our control strategies complete the process of signal transition for a preceding EV. Therefore, our control strategies have to be enhanced under such a circumstance.

10.1 Control strategy for multiple EVSP occurrences

Let D be the distance of EV detection position away from the subject intersection, which will be used for multiple EVSP occurrence control strategy. We define

$$D = L \quad \text{for the first intersection in a coordinated route, and} \quad (10-1a)$$

$$D = \min(L, L_{i,i+1}) \quad \text{for other intersections} \quad (10-1b)$$

where

L = distance of EV detection away from an intersection, which is determined by queue length under normal operation (see Chapter 3); and

$L_{i,i+1}$ = link length between intersection i and $i+1$

let Δt be the time headway between two successive EVs, denoted as i -th EV and $(i+1)$ -th EV, which are going to pass through the subject intersection. We have

$$\Delta t = t_{EV}(i+1) - t_{EV}(i) \quad (10-2)$$

where

$t_{EV}(i+1)$, $t_{EV}(i)$ = arrival time of i -th EV and $(i+1)$ -th EV at D from the subject intersection, respectively.

Denote transition period of the real-time control and recovery period of the optimal control for the i -th EV as $TP(i)$ and $RP(i)$, respectively. We have the following scenarios based on the headway Δt relative to $TP(i)$ and $RP(i)$.

1. $\Delta t < TP(i)$

In this scenario, the $(i+1)$ -th EV arrives at D before the i -th EV crosses the intersection, as illustrated in Fig. 10.1

Because our first priority is to ensure that the EV can pass through the intersection without delay, we will continue the real-time control for the i -th EV until it crosses the intersection. After the i -th EV crosses the intersection, the real-time control for the $(i+1)$ -th EV will start immediately, and there is no optimal control period for the i -th EV.

When the i -th EV has just crossed the intersection, $[TP(i) - \Delta t]$ has passed since the $(i+1)$ -th EV has been detected, as illustrated in Fig. 10.2.

Fig. 10.1 EV(i+1) arrives before EV(i) crosses intersection

Fig. 10.2 Remaining transition time for EV(i+1)

If the (i+1)-th EV is the only requesting preemption, it would need a period of

TP(i+1) for real-time control.

$$TP(i+1) = \max[D/S_{EV}, (L_{QS}(i+1) + L_{QM}(i+1))/S_{dis}] \quad (10-3)$$

where

$L_{QS}(i+1)$, $L_{QM}(i+1)$ = standing queue length, moving queue length ahead of (i+1)-th EV, respectively (please note they are current queue lengths here instead of statistical values); and

S_{EV} , S_{dis} = EV operating speed, and discharge speed, respectively.

In order to ensure the (i+1)-th EV can cross the intersection without delay, we need to discharge the queue ahead of the (i+1)-th EV before it arrives at the intersection, i.e.,

$$t_{clear}(i+1) \leq t_{EV} \quad (10-4)$$

while $t_{clear}(i+1)$ and t_{EV} can be determined by

$$t_{EV} = D/S_{EV} \quad (10-5)$$

$$t_{clear}(i+1) = \max[(L_{QS}(i+1) + L_{QM}(i+1)) / S_{mean} , (L_{QS}(i+1) + L_{QM}(i+1)) / S_{dis}] \quad (10-6)$$

where

S_{mean} = general traffic mean speed.

Of course, there also exists the possibility for the queue between the two EVs to

pull over and stop at road side. However, this scenario is less demanding than letting the queue cross the intersection in terms of transition time needed and control complexity of signal transitioning. Therefore, we will focus on the later scenario to discharge the queue before the second EV arrives at the intersection, as formulated in the Equation (10-4).

Since the i -th EV exists, the real-time control process for the $(i+1)$ -th EV cannot start until the real-time control for the i -th EV is completed. After the i -th EV has just crossed the intersection, the remaining transition time for the $(i+1)$ -th EV (see Fig. 10.2) will be

$$TP_remaining(i+1) = TP_ (i+1) - [TP_ (i) - \Delta t] \quad (10-7)$$

and the time needed to discharge the queue between the two EVs will be

$$t_{clear_remaining}(i+1) = t_{clear_}(i+1) - [TP_ (i) - \Delta t] \quad (10-8)$$

If the queue between the two EVs is short, some remaining transition period can be diverted to side roads as green time. In order to achieve such a diversion, the following procedure has to be followed.

- A safety time interval (STI) must be kept between the i -th EV and its following general traffic
- Signal for the EV route turns yellow for T_{yellow} , and then turns red (for $t_{red} \geq T_{red-1}$)
- Signal for side road turns green at $t_{red} = T_{red-1}$ for t'_{green}
- Signal for side road turns yellow for T'_{yellow} , and then turns red (for $t'_{red} \geq T_{red-2}$)

- Signal for the EV route turns green at $t'_{red} = T_{red-2}$ for t_{clear} to clear the queue
- A safety time interval (STI) must be kept between the (i+1)-th EV and its leading general traffic before the (i+1)-th EV is served

Thus we obtain the equation to determine the green time available for side road as follows.

$$t'_{green}(i+1) = TP_remaining(i+1) - [T_{yellow} + T_{red-1} + T'_{yellow} + T_{red-2} + t_{clear_remaining}(i+1) + 2*STI] \quad (10-9)$$

$$2. TP(i) \leq \Delta t < TP(i) + RP(i)$$

In this scenario, the (i+1)-th EV arrives at D after the i-th EV has crossed the intersection, and before the optimal control process for the i-th EV is completed, as illustrated in Fig. 10.3.

We have to interrupt the optimal control for the i-th EV and start the real-time control for the (i+1)-th EV, in order to ensure that the (i+1)-th EV can cross the intersection without delay. It means that the transition period of optimal control for the i-th EV will be cut to

$$RP'(i) = \Delta t - TP(i) \quad (10-10)$$

Fig. 10.3 EV(i+1) arrives during recovery period for EV(i)

The optimal control for the i -th EV will start at $t_{EV(i)} + TP(i)$, and will end at $t_{EV(i+1)}$. After this optimal control period for the i -th EV, the real-time control process begins for the $(i+1)$ -th EV.

3. $\Delta t \geq TP(i) + RP(i)$

In this scenario, the $(i+1)$ -th EV arrives at D after both transition 1 and transition 2 have been completed for the i -th EV. The EVSP operation for the i -th EV is an isolated EVSP occurrence, and control algorithms described in previous chapters are applicable to the i -th EV.

After the transition process for the i -th EV and then a period of normal operation, the $(i+1)$ -th EV arrives at EV detection position D . At this point, the real-time control starts to clear the way for the $(i+1)$ -th EV. The interference between the $(i+1)$ -th EV and

the (i+2)-th EV will be considered according to headway $\Delta t = t_{EV}(i+2) - t_{EV}(i+1)$ relative to transition periods $TP(i+1)$ and $RP(i+1)$, and the control strategies discussed previously for scenarios 1, 2, and 3 are to be put in place.

10.2 Flow chart of control algorithm and simulation for multiple EVSP occurrences

In summary, for the whole transition process of EVSP, consisting of transition 1 and transition 2, the algorithm of real-time control for transition 1 and the optimal control for transition 2 is illustrated in Figure 10.4.

Fig 10.4. Flow chart of control algorithm for multiple EVSP occurrences (continued on next page)

**Fig 10.4. Flow chart of control algorithm for multiple EVSP occurrences
(continued on next page)**

Fig 10.4. Flow chart of control algorithm for multiple EVSP occurrences

In Figure 10.4, the following symbols are used for convenience.

t_N = starting time of a period for normal operation;

queue_1 = queue lengths on individual approaches to the subject intersection at the beginning of t_N ;

queue_2 = queue lengths on individual approaches to the subject intersection at the end of t_N ;

queue_3 = queue lengths on individual approaches to the subject intersection at the end of a period for real-time control; and

queue_4 = queue lengths on individual approaches to the subject intersection at the end of a period for optimal control.

10.3 Simulation of control strategy for multiple EVSP occurrences

The four-leg intersection with two-phase signal system is used in this chapter. The intersection layout, traffic volumes, pre-timed signal plan for normal operation, and average queue lengths under normal operation are the same as in Chapter 5. Only the real-time control strategy and the optimal control strategy will be different from Chapter 5.

EVs are produced on approach 1 with the heaviest traffic among the four approaches. Three EVSPs at closely timed intervals are simulated, and the time headways between two successive EVs are 15, 30, 60, and 80 seconds, respectively. The entry time of the first EV is at beginning of green, half green, beginning of red, and half red for approach 1, respectively, in order to consider randomness of EV arrival times. For each headway and each EV entry time, six different periods are used to consider the fluctuation in general traffic at the intersection. Simulation results are then averaged over the four entry times and the six cycles for each headway EVSP scenario. EVSP scenarios are listed in Table 10.1.

Simulations were performed and results are summarized in Tables 10.2 to 10.6.

Table 10.1 Multiple EVSP scenarios

Scenario #	Δt (sec)	EV(1) (sec)	EV(2) (sec)	EV(3) (sec)	Remarks
1	15	0, $g(1)/2$ $g(1)+y(1)$ $g(1)+y(1)+r(1)/2$	EV(1) $+ \Delta t$	EV(2) $+ \Delta t$	$\Delta t \approx TP(1)/4$
2	30				$\Delta t \approx TP(1)/2$
3	60				$\Delta t \approx TP(1)$
4	80				$TP(1) < \Delta t < TP(1) + RP(1)$

Notes: Δt =time headway between two successive EVs;
 EV(i)=entry time of i-th EV in a row;
 $g(1)$ =green interval for approach 1;
 $y(1)$ =yellow interval for approach 1; and
 $r(1)$ =red interval for approach 1.

Table 10.2 Failure number of control strategy for multiple EVSPs

Scienario #	1	2	3	4
Failure #	1	7	15	5

Note: A failure was recorded if the queue ahead of an EV had not been discharged completely when the EV arrived at the intersection.

Table 10.3 Average green time for side road during multiple EVSPs

Scienario #	1	2	3	4
Average (sec)	35.85	25.30	13.84	24.20
std	11.02	17.37	17.44	17.36

Note: std=standard deviation.

Table 10.4 Average spare time during multiple EVSPs

Scienario #	1	2	3	4
Average (sec)	0.42	1.04	2.63	1.83
std	1.28	2.77	4.81	3.95

Note: std=standard deviation.

Table 10.5 Average queue lengths after real-time control for multiple EVSPs

Scenario #		queue on approach (veh)			
		1	2	3	4
1	mean	20.54	3.14	8.20	10.14
	std	6.63	2.89	6.38	10.65
2	mean	21.52	3.63	6.43	7.49
	std	8.43	3.39	6.03	5.34
3	mean	19.77	10.40	3.33	17.71
	std	13.26	10.28	5.52	16.37
4	mean	22.64	6.01	10.27	14.70
	std	8.48	6.07	9.93	12.63
Average over four scenarios		10.51	9.77	12.80	13.41

Notes: 1. mean = average queue length.

2. std = standard deviation of queue length.

3. queue lengths were measured after the 3rd EV had just crossed the intersection.

Table 10.6 Average queue lengths after optimal control for multiple EVSPs

Scenario #		queue on approach (veh)			
		1	2	3	4
1	mean	15.31	1.48	1.68	13.39
	std	9.73	1.42	3.47	10.25
2	mean	16.70	1.28	0.97	11.11
	std	7.10	1.34	2.21	3.39
3	mean	17.69	4.85	4.90	19.35
	std	8.36	7.53	5.60	11.98
4	mean	22.06	1.20	4.68	21.54
	std	7.65	1.65	5.46	8.58
Average over four scenarios		7.97	7.52	11.70	12.37

Notes: 1. mean = average queue length.

2. std = standard deviation of queue length.

3. queue lengths were measured at the end of transition 2 for the 3rd EV.

10.4 Analysis of simulation results

1. Failure number of control strategy

Table 10.2 lists failure number of control strategy for multiple EVSPs. From the table, we can see that number of failures increased with time headway Δt between two successive EVs for scenarios 1 through 3, in which Δt was less than the transition period for the real-time control (about 60 seconds). When Δt was 15 seconds, the queue between two successive EVs was short, and the real-time control strategy could easily discharge the queue before the second EV reached the intersection. When Δt increased to 30 seconds, the queue between two successive EVs increased accordingly, and more cases were observed, in which the real-time control algorithm could not clear the way before the second EV reached the intersection. The number of failures increased to 7 for cases with 30 seconds of Δt . For scenario 3 with 60 seconds of Δt , which was almost the same as the transition period for the real-time control, the longest queues were produced between two successive EVs before the first EV crossed the intersection. These longest queues caused 15 failures in the real-time control. The second EV had to stop and reduce speed before it reached the intersection and then crossed the intersection after the queue ahead of it had been discharged.

For scenario 4, in which Δt was greater than the transition period for the real-time control, there was a recovery period under the optimal control between the instant an

EV had crossed the intersection and the moment a second EV was requesting signal preemption. Although these recovery periods (about $80-60 = 20$ seconds) were not long enough to resume queues to normal traffic conditions, they did reduce queues on approach 1, from which EVs entered the system, and the number of failures decreased to 5. It is expected that the number of failures will further decrease with longer recovery periods, and for cases with full recovery periods the real-time control algorithm shall be able to clear the way for an EV in these cases, as demonstrated in the simulation results of single EVSP (see Chapter 5).

Considering all the headway scenarios, the failure rate was still in acceptable range after the total number of EVSP occurrences in the simulation was taken into consideration. Each of the four EVSP scenarios had three EVs in a row, and each EV was simulated with four entry times and at six different cycles. The total number of EVSP occurrences could be calculated as $4 \times 3 \times 4 \times 6 = 288$ and the total number of failures was 28 (i.e., $1+7+15+5=28$). These result in 9.7% failure rate (i.e., $28/288=9.7\%$).

2. Average green time for side road during multiple EVSPs

Table 10.3 lists the average green time for side road during multiple EVSPs. Among the four scenarios, the longest green time (35.85 seconds) for side road appeared in scenario 1 with the shortest Δt (15 seconds). With increase of Δt , the queue between two successive EVs also became longer, and it required longer time to clear the way for the second EV. Therefore, when Δt increased from 30 to 60 seconds, green time

for side road decreased from 25.30 to 13.84 seconds. However, for scenario 4 with 80 seconds of Δt , the queue ahead of the second EV was reduced by the recovery period for the optimal control. As a result, the green time increased to 24.20 seconds, compared with 13.84 seconds for scenario 3 with 60 seconds of Δt (about the same as the transition period for the real-time control).

3. Average spare time during multiple EVSP

Table 10.4 lists the average spare time during multiple EVSPs. As expected, the average spare time was closely associated with the average green time for side road. A long green time for side road means that the queue ahead of an EV was short, and a large portion of transition period for the real-time control could be diverted to side road as green time. It was not difficult for the real-time control algorithm to allocate just enough green time to clear the way for an EV after it gave green indication to side road. On the other hand, a short green time for side road indicates that a long queue existed ahead of an EV, and most portion of transition period for the real-time control had to be used to discharge the queue. Only a small remaining portion of the transition period could be diverted to side road as green time. In such a situation, it was more likely to happen that the small remaining portion of the transition period was shorter than the minimum time to switch over traffic signals between EV route and side road. The minimum switchover time was required to switch green indication from EV route to side road and then back to EV route after a minimum period of green time for side road.

If such a situation did happen, the real-time control algorithm would not divert the small remaining portion of transition period to side road, and all the transition period would be used as green for EV route. As a result, the small remaining portion would be wasted as spare time, because it was not necessary for EV route and it could not be used by the side road. From Table 10.4, we can see that spare time was 0.42, 1.04, 1.83, and 2.63 seconds, associated with 35.85, 25.30, 24.20, and 13.84 seconds of green time for the side road, respectively.

4. Average queue lengths after the real-time control

To assess the efficiency of the control strategy for multiple EVSP occurrences against base case without EVSP, the average queue lengths after the real-time control are converted to relative values of queue lengths under normal operation (i.e., ratios of queue lengths under the real-time control and corresponding values under normal operation). Table 10.7 lists the relative queue lengths after the real-time control. The calculation results are also illustrated in Fig. 10.5

As shown in Table 10.7 and Fig. 10.5, on approach 1 (the EV approach), relative queue lengths after the real-time control were between 105% and 120%. The small differences among individual scenarios indicates that the real-time control algorithm had a steady behaviour for all the four EVSP scenarios, although it produced slighter longer queues than corresponding average queue under normal operation.

On approach 3 (the opposite approach), relative queue lengths were very short.

The longest one (87%) appeared in scenario 4, and the shortest one (28%) appeared in scenario 3. An explanation for the results is that approach 3 got the same green time as approach 1, while approach 3 had a lighter traffic than approach 1.

Table 10.7 Relative queue lengths after real-time control

Scenario		Approach				Over all approaches
		1	2	3	4	
Base case	q_N (veh)	18.8	7.25	11.8	13.55	51.4
EVSP scenario 1	q_R (veh)	20.54	3.14	8.20	10.14	42.02
	q_R/q_N (%)	109	43	69	75	82
EVSP scenario 2	q_R (veh)	21.52	3.63	6.43	7.49	39.07
	q_R/q_N (%)	114	50	54	55	76
EVSP scenario 3	q_R (veh)	19.77	10.40	3.33	17.71	51.21
	q_R/q_N (%)	105	143	28	131	100
EVSP scenario 4	q_R (veh)	22.64	6.01	10.27	14.70	53.62
	q_R/q_N (%)	120	83	87	108	104

Notes: 1. q_N = average queue length under normal operation; and

2. q_R = average queue length after real-time control.

On approach 4 (the cross road), relative queue lengths were similar to those on approach 2. They were 75%, 55%, 131%, and 108% in scenario 1, 2, 3, and 4, respectively. The longest relative queue appeared in scenario 3, which caused the longest queues on approach 1 among the four EVSP scenarios, and most portion of a transition period had to be used to discharge the queue on approach 1. In scenario 4, although the recovery period under the optimal control was not long enough to control queue length below normal condition, it did relieve the queue to a slightly longer length than corresponding value under normal operation by 8%.

Fig. 10.5 Relative queue lengths after real-time control

A promising result was observed in average relative queue lengths over all approaches. The real-time control algorithm for multiple EVSP occurrences kept the overall relative queue lengths below or about 100% in four EVSP scenarios. These were 82%, 76%, 100%, and 104% for scenario 1, 2, 3, and 4, respectively.

5. Average queue lengths after the optimal control

Similar to average queue lengths following the real-time control, the average queue lengths after the optimal control were also converted to relative values of queue lengths under normal operation (i.e., ratios of queue lengths under the optimal control and corresponding values under normal control). Table 10.8 lists the relative queue lengths after the optimal control. The calculation results are illustrated in Fig. 10.6.

As shown in Table 10.8 and Fig. 10.6, on approach 1 (the EV approach), relative queue lengths after the optimal control were below 100% in all scenarios except scenario 4 (i.e., 81%, 89%, 94% for scenario 1, 2, and 3, respectively). Somewhat surprisingly, scenario 4, the only one with about 20 seconds of recovery period between two successive EVs, had an average queue length that was 17% higher than corresponding average queue length under normal operation. One probable explanation is that a longer recovery period did not necessarily produce a better control result. For certain traffic volumes at an intersection, it is possible that extending the recovery period beyond some value will not benefit the optimal control algorithm any more.

On approach 3 (the opposite approach), relative queue lengths were very short.

These were 14%, 8%, 42%, and 40% in scenario 1, 2, 3, and 4, respectively. This is because approach 3 got the same green time as approach 1 and the traffic was lighter on approach 3 than on approach 1.

On approach 2 (the cross road), relative queue lengths were also very short (i.e., 20%, 18%, 67%, and 17% for scenario 1, 2, 3, and 4, respectively). Similar to approach 3, approach 2 received the same green time as approach 4, while traffic volume was lighter on approach 2 than on approach 4. Consequently, there were rather short relative queue lengths on approach 2.

Table 10.8 Relative queue lengths after optimal control

Scenario		Approach				Over all approaches
		1	2	3	4	
Base case	q_N (veh)	18.8	7.25	11.8	13.55	51.4
EVSP scenario 1	q_O (veh)	15.31	1.48	1.68	13.39	31.86
	q_O/q_N (%)	81	20	14	99	62
EVSP scenario 2	q_O (veh)	16.70	1.28	0.97	11.11	30.06
	q_O/q_N (%)	89	18	8	82	58
EVSP scenario 3	q_O (veh)	17.69	4.85	4.90	19.35	46.79
	q_O/q_N (%)	94	67	42	143	91
EVSP scenario 4	q_O (veh)	22.06	1.20	4.68	21.54	49.48
	q_O/q_N (%)	117	17	40	159	96

Notes: 1. q_N (veh)= average queue length under normal operation; and

2. q_O (veh)=average queue length after optimal control.

Fig. 10.6 Relative queue lengths after optimal control

On approach 4 (the cross road), average queues following the optimal control were shorter than corresponding values under normal operation in scenarios 1 and 2, and the relative queue lengths were 99% and 82%, respectively. In scenarios 3 and 4, the optimal control strategy could not control the average queues below corresponding values under normal operation, and the relative queue lengths reached 143% and 159% for scenario 3 and 4, respectively. Compared with relative queues on approach 1, one possible reason for the result is that approach 1 had a preferential position in getting green time during the recovery period. At the beginning of the recovery period, the green indication always faced approach 1 (the EV approach). Switching green indication to approach 4 (the cross road) would happen only if a much longer relative queue existed on approaches 2 or 4 than on approaches 1 and/or 3, so that a net gain could be obtained after taking into account the waste of resource due to switchover.

Another promising result is obtained in terms of average relative queue lengths over all approaches. Under the optimal control strategy for multiple EVSP occurrences, the overall relative queue lengths after the recovery period were controlled at 62%, 58%, 91%, and 96% for scenario 1, 2, 3, and 4, respectively. This result is not surprising because it coincides with the goal of the optimal control algorithm, i.e., minimization of the overall relative queue length instead of individual approaches at an intersection (see Equation (4-26) for control performance index definition).

Chapter 11 Summary, Conclusions and Recommendations

11.1 Summary of control strategy concept for EVSP

Emergency Vehicle Signal Preemption (EVSP) process consists of two periods of time. The first one, transition 1, begins at the instant an EV approaching an intersection is detected, and ends at the moment the EV crosses the intersection. The traffic signal phase has to be changed from normal operation to EVSP operation. The second one, transition 2, is the time duration from the instant the EV has just crossed the intersection to the moment the traffic signal resumes normal operation.

The two transition periods, namely from normal operation to EVSP operation and then back to normal operation, have different objectives and requirements for the EVSP system. For transition 1, the EVSP system should ensure that the EV can pass through the intersection at operating speed without stopping or reducing speed, while minimizing the impact of EVSP on general traffic. For transition 2, the EVSP system should recover the traffic signal phase to normal operation, which is usually a coordination pattern. This should be done as quickly as possible, and also should minimize the impact of EVSP on general traffic.

The real-time signal control strategy suggested in this research for transition 1 is

accomplished in two steps. The first step determines a suitable distance of EV detection away from the intersection in order to ensure that the EV can pass through the intersection at operating speed under almost every possible traffic condition. The second step allocates dynamically green time that is unnecessary for the EV to traffic on cross road to minimize the impact of EVSP on general traffic.

For transition 2 the traffic signal control problem is formulated as a time-invarying dynamic system, and an optimal control model is developed. A two-phase algorithm is then introduced to solve the optimal control problem because of the interrelationship between sequences of greens, yellows, and all red periods in successive time intervals.

A stepwise search strategy is also adopted to reduce the computation complexity in the two-phase algorithm, considering the requirement of real-time traffic signal control. From the total N discrete time steps for the whole procedure of transition 2, only M time steps are considered at a time. The M -step model makes it possible for EVSP system to control the signal timing transition in real-time mode, and minimize the traffic queue caused by EVSP.

11.2 Summary of simulation results for proof of concept

To demonstrate the effectiveness of the real-time control strategy for transition 1 and the optimal control strategy for transition 2, numerical simulations were conducted.

The fixed held time control strategy and the smooth control strategy were selected as reference strategies for transition 1 and transition 2, respectively. Computer programs were developed in MATLAB environment for simulation of traffic signal timing transitioning under the fixed time control strategy and the real-time control strategy for transition 1, and the smooth control strategy and the optimal control strategy for transition 2.

In Chapters 5 and 6, the EVSP at a single intersection was first studied. To evaluate the impact of EVSP in different traffic conditions, the EV was introduced separately into the signal control system from main road with heavier traffic and from side road with lighter traffic. The simulation for each control strategy consisted of six scenarios in which the EV entered the traffic signal control system at different instant of time in normal signal operation cycles. While the randomness of EV arrivals at an intersection was considered by the six scenarios, the randomness of general traffic arrivals was considered by six normal signal operation cycles for each scenario.

The pre-timed plan was chosen for normal signal operation before an EV is detected at an intersection, and the fixed held time control strategy was chosen for transition 1 as the base on which the optimal control algorithm and the smooth transitioning for transition 2 were simulated. The purpose was to compare our real-time control strategy with the fixed held time control strategy for transition 1 and our optimal control strategy with the smooth control strategy for transition 2 on an

efficient and objective basis.

Simulation results presented in Chapters 5 and 6 provide the evidence that the real-time control strategy and the optimal control strategy are superior to the fixed held time control strategy and the smooth control strategy, respectively.

In order to confirm that the real-time control strategy and the optimal control strategy are applicable to a coordinated route as well, they were applied to a four-intersection coordinated route. Simulation results described in Chapter 7 provided evidence of applicability of the real-time control strategy and the optimal control strategy to the coordinated route. Furthermore, simulation results also suggest that an optimized normal operation timing plan could provide a better base for the application of the optimal control strategy.

Theoretical analyses and simulation results indicate the effectiveness and efficiency of the real-time control strategy and the optimal control strategy that are proposed in this research.

Commonly used traffic signal control strategies for transition I give green signals to EV for a fixed period of time. This kind of strategy is not suitable to deal with the fluctuation in traffic conditions. Frequently, the fixed green time is too short to clear the queue ahead of an EV when there is a long queue at the intersection, or it is unnecessarily long when there is a short queue at the intersection. By allocating just enough green time for the queue ahead of the EV to clear the intersection according to

current traffic condition, the real-time control strategy can reduce queues on the cross road to a minimum length, while it ensures that the EV can pass through the intersection at operating speed.

In the single intersection simulation, although a 30-second fixed held time on main road was much shorter than the 59 seconds for real-time control algorithm, it produced a longer average queue of 8.62 vehicles, compared with the average queue of 7.83 vehicles under the real-time control algorithm. Also, it failed seven times in letting EV cross the intersection at operating speed. On the side road, the 20-second fixed held time was slightly less than the 23 seconds for the real-time control. Although it succeeded in EVSP operation, it produced a longer average queue of 9.43 vehicles, compared with the average queue of 7.76 vehicles under the real-time control strategy.

Furthermore, the real-time control strategy is expected to provide the most benefit under situations where fluctuation in vehicle arrivals are frequently encountered on both crossing roads with heavy traffic at the intersection.

For recovery period, transition 2, the optimal control strategy proposed in this research handled conflicting demands from main road and side road very well. Based on the optimal control model and the stepwise search algorithm established in this research, the optimal control strategy outperformed the well recognized smooth transitioning algorithm in both recovery time and average queue caused.

Also in the single intersection simulation, on the main road the optimal control

strategy with average recovery time of 79.83 seconds reduced average queue by 22.2% at the intersection to 9.62 vehicles, compared with 12.37 vehicles under the smooth control algorithm with average recovery time of 162 seconds. On the side road the optimal control strategy got back to normal operation in average 69.17 seconds and reduced average queue by 15% at the intersection to 8.1 vehicles from 9.53 vehicles under the smooth control algorithm with average recovery time of 180 seconds.

The success of the optimal control strategy results from the randomness of vehicle arrivals at the intersection. The fluctuation in traffic makes it possible for the optimal control strategy to allocate green time to each road dynamically only when the most benefit can be obtained by doing so. Furthermore, the methodology and techniques proposed in this research make it feasible.

In the coordinated route simulation, the real-time control strategy wasted very little time with only 1.32 seconds of spare green time, while the 50-second fixed held time control strategy had 40.95 seconds of spare time, during which neither the EV route nor side roads could make use of the green time resource. The short spare time under the real-time control strategy means that there was more green time for side roads during EVSP. For all the four intersections in the subject route, an average of 109.86 seconds was extracted as green time for side roads during each EVSP occurrence.

The effect of extracted green time for side roads was indicated by the queue lengths after transition 1. The real-time control strategy increased control efficiency by 26.2%

compared with the 50-second fixed held time control strategy. While it produced an average queue of 5.53 vehicles, corresponding queue under the fixed held time control algorithm was 7.49 vehicles.

Also in the coordinated route simulation, for transition 2 the optimal control algorithm had a better performance over the smooth transitioning algorithm. Although it used only 36.2% of recovery time under the smooth control algorithm (68.5 seconds vs. 189 seconds), the optimal control strategy produced shorter average queues at every intersection in the subject route. By reducing the average queue on each approach to 5.31 vehicles from 7.87 vehicles incurred by the smooth control algorithm, the optimal control strategy increased control efficiency by 32.5%.

11.3 Summary of further simulation results

After the effectiveness and efficiency of the real-time control strategy and the optimal control strategy had been proven by the simulation results presented in Chapters 5 to 7, further research was conducted to establish general guidelines for the estimation of the impact of preemption and for the selection of the signal preemption control strategies and related parameters.

Further investigations were performed in three aspects: effect of different traffic volumes on the real-time control strategy and the optimal control strategy; effect of

different weighting matrices on the optimal control strategy, and control strategies for multiple EVSP occurrences.

In Chapter 8, the real-time control strategy and the optimal control strategy were used sequentially during the whole EVSP process. Simulations were conducted at different saturation levels, and volume/capacity ratios ranged from 0.27 on approach 2 (on the cross road) to 1.26 on approach 1 (on the EV route). EV entered the system from approach 1 at four different instants and during six different cycles for each entry instant. Simulation results were then averaged over the four EV entry times and the six cycles for each traffic saturation levels.

In the simulation, the real-time control strategy failed in only two cases. Considering there were sixteen traffic saturation scenarios and twenty-four cases (four EV occurrences and six cycles for each EV occurrence) for each traffic scenario, the real-time control strategy can be regarded as reliable.

As expected, queue lengths following the real-time control increased with increasing traffic saturation levels. When traffic volume/capacity ratio reached 1.08 on the EV approach, queue lengths on the EV approach were controlled below 108.2% of corresponding value under normal operation. Due to the diversion of green time to the cross road during the real-time control process, queue lengths on the cross road were controlled below 101% of corresponding value under normal operation for traffic volume/capacity ratios of up to 1.08 on the cross road.

When volume/capacity ratios reached 1.26 on the EV approach, the normal operation timing plan could not control queues below a reasonable length. If such a queue length were used to determine the distance of EV detection, the control results at the end of transition 1 would be unpredictable because of uncertain factors during a long transition period, such as variation of general traffic speed, and influence of mid-block friction. Under these circumstances, the normal operation signal timing plan should be adjusted or other methods should be used so that volume/capacity can be kept below 1.0.

After the EV crossed the intersection, transition 2 under the optimal control began to get the signal back to normal operation. On the cross road, average queue lengths were controlled below corresponding values of normal operation. The longest relative queue of 63.15% appeared in scenario 16, in which oversaturated traffic (volume/capacity ratios 1.08) existed on both the EV route and the cross road. At the end of the optimal control process, slightly longer average queues than corresponding values under normal operation were observed only on the EV approach in scenarios with oversaturated traffic (volume/capacity ratios 1.08) on the approach. When volume/capacity ratios reached 1.08 on both the EV approach and the cross road, the longest relative queue of 110.8% was produced on the EV approach.

In Chapter 9, simulations were conducted to assess the impact of weighting matrices on the optimal control algorithm. Twenty-seven combinations of weighting matrices

were selected for simulation. Then the optimal control strategy with each set of weighting matrix combination was applied to four different traffic conditions. For each set of weighting matrix combination and each traffic condition, simulation results were averaged over four EVSP scenarios and six cycles for each EVSP scenario.

When traffic volumes were light on both the EV road and the cross road (traffic scenario 1), weighting matrix R for intermediate states was more important than the weighting matrix P' for free final state in reducing the average queue length, q_{mean} , over all the approaches to the intersection. Short final queues caused the result that a greater ratio of R to P' benefited the optimal control algorithm.

When volume/capacity ratios rose to 0.72 and 0.54 on the EV route and the cross road, respectively (traffic scenario 2), a large R made q_{mean} short and stable. Simulation results also indicated that R was more important in reducing q_{mean} than the weighting matrix S for control variables.

For traffic scenario 3 with volume/capacity ratios of 0.9 and 0.81 on the EV route and the cross road, respectively, final queue lengths also rose accordingly. Consequently, P' should not be too small relative to R to benefit the optimal control strategy. Simulation results also suggested that large P' and R make q_{mean} short and stable and S should not be greater than R .

For oversaturated traffic condition (scenario 4) with volume/capacity ratio of 1.08 on both the EV route and the cross road, weighting matrix R for intermediate states and

weighting matrix S for control variables had the same order in affecting the efficiency of the optimal control algorithm.

A further simulation with greater differences in weighting matrices was also performed in Chapter 9. Simulation results indicated that the shortest queues under the optimal control algorithm were not sensitive to weighting matrices and greater differences in P' , R , and S did not cause significant improvement in control efficiency.

Control strategies and control efficiencies for multiple EVSP occurrences were described in Chapter 10. When two successive EVs request signal preemptions at a closely timed interval, the first priority for the EVSP operation is to discharge the queue ahead of each EV and to ensure that each EV can cross the intersection without delay. If the queue can be discharged before the EV reaches the intersection, it is possible that a portion of the transition period can be diverted to the cross road. The optimal control algorithm will be put in place only if it does not affect the process of the real-time control for any EV, and the optimal control process for an EV will be interrupted by the approaching of another EV during the recovery period for the first EV.

Simulations were conducted with three EVs requesting preemptions at closely timed intervals. The time headways between two successive EVs were 15, 30, 60, and 80 seconds, respectively. The first EV among the three EVs in a row entered control system at the beginning of green, half green, beginning of red, and half red for the EV route respectively. For each headway and each EV entry time, six cycles were simulated to

consider the fluctuation in general traffic at the intersection. Simulation results for each headway scenario were averaged over the four EV entry times and the six cycles for each EV entry time.

As for the number of control strategy failures, when time headways were 15, 30, 60, and 80 seconds, the real-time control algorithm could not clear the way for EVs in 1, 7, 15, and 5 different cases for multiple EVSP occurrences, respectively. For cases with time headway of 60 seconds, an EV entered the system at about the end of transition 1 for a preceding EV, and longest queues were encountered between the two EVs among the four multiple EVSP scenarios. The longest queues caused 15 failures and the failure rate for this headway scenario reached 20.8% (i.e., $[15/(3 \times 4 \times 6)] \times 100\% = 20.8\%$). When time headway further rose to 80 seconds (headway scenario 4), there was a recovery period after an EV had passed through the intersection and before another EV entered the system. The recovery period relieved the queues accumulated during the real-time control process, and consequently the number of failures decreased to five. Over all the headway scenarios, the total number of EVSP occurrences was 288 (i.e., $4 \times 3 \times 4 \times 6 = 288$) and the total number of failures was 28 (i.e., $1 + 7 + 15 + 5 = 28$). This amounts to a failure rate of 9.7% (i.e., $(28/288) \times 100\% = 9.7\%$).

Average spare time was closely associated with average green time for the side road. For the four headway scenarios, average green times for side road were 35.85, 25.3, 13.84, and 24.2 seconds, respectively, and corresponding average spare times were 0.42,

1.04, 2.63, and 1.83 seconds, respectively. A longer green time for side road means a shorter queue to be discharged on the EV approach ahead of an EV, and consequently it is easier for the real-time control algorithm to allocate just enough time to clear the way for the EV after some green time has been diverted to side road.

Compared with average queues under normal operation, average relative queues on the EV approach following the real-time control were longer than 100%, ranging from 105 to 120%. These relative queue lengths suggested the applicability of the real-time control algorithm for all the four headway scenarios. On other approaches, the real-time control strategy produced shorter queues than corresponding values under normal operation for all headway scenarios with three exceptions: scenario 3 on approach 2, and scenarios 3 and 4 on approach 4. The longest relative queues were 143% and 131% in scenario 3 on approaches 2 and 4, respectively due to the longest queues formed on approach 1 (the EV approach), which needed the longest green time to discharge. On approach 4, the real-time control algorithm produced a slightly longer queue (i.e., 8%) in scenario 4 than corresponding value under normal operation by 8%. For all the approaches, the relative queue lengths were controlled at 82%, 76%, 100%, and 104% for scenarios 1, 2, 3, and 4, respectively.

Regarding average relative queue lengths following the optimal control process, queues longer than 100% occurred on approach 1 (the EV approach) in scenario 4 (117%) and on approach 4 in scenarios 3 (143%) and 4 (159%). The longest relative

queues were observed in scenario 4, which was the only one with some recovery period between two successive EVs among the four scenarios. This result suggested that extending recovery period beyond some value did not necessarily benefit the optimal control algorithm any more in reducing average relative queue lengths. Compared with approach 4 (the cross road), approach 1 (the EV approach) had shorter relative queues in scenarios 3 and 4. A possible reason for this result was that approach 1 had a preferential position in getting green time during the recovery period because green indication always faced approach 1 (the EV approach) at the beginning of the recovery period.

It was promising that average relative queue lengths over all approaches were 62%, 58%, 91%, and 96% at the end of the optimal control process for scenarios 1, 2, 3, and 4, respectively. These control results lead to a smooth resumption of traffic signal to normal operation from EVSP.

11.4 Conclusions

This research made the following major contributions.

1. A real-time control strategy is suggested for transition 1. In the real-time control strategy, the distance of EV detection is first determined based on statistical values of queue length under normal operation so that the EV can cross the intersection

without delay in almost every possible traffic condition. The whole transition period for an EV is then divided into two parts according to current queue lengths at the intersection. One part is just enough to clear the way for the EV, and another part is diverted to cross road whenever it is possible.

2. An optimal control strategy is proposed for transition 2. Based on current traffic conditions and the coordinated signal timing plan for normal operation, a recovery period is first determined, so that the signal indications can resume to coordinated normal operation mode immediately at the end of transition 2. The recovery process is then formulated as an optimal control problem and a time-invarying dynamic system is established. A two-phase algorithm is adopted to overcome the difficulty in solving the optimal control model, which is caused by the interrelationship between successive signal sequences. An M-step model is also suggested to reduce the computation complexity for practical implementation consideration of the optimal control strategy.
3. Simulation software is developed to compare the real-time control strategy with commonly used "fixed held time" transitioning for transition 1. Simulation results indicate that the real-time control strategy is superior to the "fixed held time" transitioning.
4. Simulation software is developed to compare the optimal control strategy with the well known "smooth" transitioning for transition 2. Simulation results show that the

optimal control strategy is superior to the "smooth" transitioning.

5. In situations where signalized intersections are closely placed in a coordinated route, vehicle arrivals at an intersection are determined by the control strategy at its upstream intersection, and the link length between two successive intersections is probably shorter than the required minimum distance of EV detection. The real-time control algorithm is enhanced for the coordinated route control. Simulations demonstrate the applicability of the real-time control strategy and the optimal control strategy to the coordinated route.
6. Simulations conducted at different traffic flow rates demonstrate that the real-time control strategy and the optimal control strategy are appropriate for different traffic conditions up to and even slightly higher than the saturation level.
7. Assessment is performed for the impact of weighting matrices on efficiency of the optimal control strategy. Simulations result in the following findings.
 - a) Large weighting matrix P' for free final state and large weighting matrix R for intermediate states relative to weighting matrix S for control variables will benefit the stability of the optimal control strategy and will cause a reduction in overall relative queue length at the intersection.
 - b) For very light traffic situations, weighting matrix R for intermediate states should be greater than weighting matrix P' for free final state in terms of control stability and efficiency.

- c) The shortest average queue under the optimal control strategy is not sensitive to weighting matrices P' , R , and S , and great differences in P' , R , and S do not cause significant improvement of control efficiency. As a result, an optimal weighting matrix combination can be obtained by searching a small space of weighting matrix combinations for practical application.
8. The real-time control strategy and the optimal control strategy are applicable to single EVSP occurrence as well as multiple EVSP occurrences. For this purpose, enhancements are made to the control algorithms. Simulations are conducted for situations with three EVs requesting signal preemptions in closely timed intervals. Simulations demonstrate promising results under the real-time control strategy and the optimal control strategy for multiple EVSP occurrences. More than ninety percent of EVSP occurrences could be served successfully and smoothly. Compared with single EVSP occurrence in the same traffic setting, the multiple EVSP occurrences caused a higher failure rate by only 9.7% under the real-time control strategy.

11.5 Recommendations

While the results presented in this research are promising, the following problems have been identified and are recommended for future study.

1. The real-time control strategy assumes that there is ample space beyond the intersection so that vehicles in a queue at the intersection can find space to pull over after they cross the intersection. If there is no space available beyond the intersection, such as in grid lock traffic conditions, the real-time control strategy will not be able to clear the way for the EV by allotting green time to the queue. Other control strategies have to be developed to deal with these situations.
2. The distance of EV detection is determined according to statistical values of queue length under normal operation. If the signal timing plan for normal operation is not suitable for traffic conditions or it is not able to control queues below a reasonable length, especially in highly saturated traffic conditions, long queues may form on the EV approach. Consequently, the distance of EV detection will be far away from the intersection, which will make it difficult to estimate mid-block friction and EV and other vehicle arrival time at the intersection. Under such circumstances, the determination of EV detection distance has to be modified so that reliable vehicular information at the intersection can be obtained.
3. The real-time control strategy performs rather well if an EV can be detected at the required minimum distance of EV detection. However, if an EV cannot be detected at such a distance, for example, an EV suddenly starts from a mid-block and the distance of the EV from an nearby intersection is shorter than the required minimum distance of EV detection, it is possible that the real-time control strategy cannot

discharge the queue at the intersection before the EV reaches the intersection. In order to ensure the EV can cross the intersection without delay, other methods at network level have to be developed to provide the EVSP system with an adequate notification time prior to the approaching of the EV at the intersection.

4. Although multiple EVSP occurrences have been considered in this research, all EVs must enter the control system from the same approach. If two EVs enter the control system from different approaches at almost the same time, the real-time control strategy and the optimal control strategy will not be able to solve the conflict. A simplified method on "first come first served" basis can be adopted if no better alternative is available for the problem. However, advanced control processes deserve further study to achieve optimized results.
5. Detailed formulations and algorithms of the real-time control strategy and the optimal control strategy are developed for the four-leg intersection layout with two-phase signal control system. They should be enhanced to control traffic signal timing transition for emergency vehicle preemption at intersections other than four-leg layout and two-phase signal control system.

References

Abbas, M., D. Bullock, and L. Head (2001). "A real-time offset transitioning algorithm for coordinating traffic signals," Transportation Research Record 1748, Transportation Research Board, Washington, D.C.: 26-39.

Advanced Traffic Management Systems Committee and Advanced Public Transportation Systems Committee (2002). An overview of transit signal priority. Final Draft. ITS America, Washington, D.C.

Anderson, C.V. (Editor) (2002). The Federal Emergency Management Agency (FEMA). Nova Science Publishers, Inc., New York.

Bachelder, A. and C. Foster (2004). System would predictively preempt traffic lights for emergency vehicles. On line at: <http://www.nasatech.com/briefs/oct04/npo30573.html>.

Baker, R.J. et al. (2004). An overview of transit signal priority-revised and updated. ITS America, Washington, D.C.

Banks, James H. (2002). Introduction to transportation engineering. 2nd edition, McGraw-Hill Companies, Inc., New York.

Bellman, R.D. (1957). *Dynamic Programming*. Princeton Univ. Press, Princeton, NJ.

Ran, B., and D. Boyce (1996). *Modeling dynamic transportation networks: An intelligent transportation system oriented approach*. 2nd revised edition. Springer-Verlag Berlin, Germany.

Black, J.R. (2003), "Preemption versus priority service for light rail transit vehicles", *Transportation Research E-Circular*, Transportation Research Board, Washington, D.C.: 166-173

Bretherton, R. D., K. Wood and G. T. Bowen (1998). *SCOOT Version 4*. Proceedings IEE 9th international conference on traffic monitoring and control. London.

Bullock, D., J. Morales, and B. Sanderson (1999), "Evaluation of emergency vehicle signal preemption on the Virginia route 7 corridor," Technical Report, Federal Highway Administration, Publication No. FHWA-RD-99-070

Chang, Jinil, Edward B. Lieberman, and Elena Shenk Prassas (2000). "Queue estimation algorithm for real-time control policy using detector data," Presented at the Transportation Research Board's 79th Annual Meeting, Washington, D.C.

Collura, J., H. Rakha, and J. Gifford (2003). Guidelines for the planning and development of emergency vehicle preemption and transit priority strategies. Draft 1.0. Virginia Tech. Transportation Institute and George Mason University, Virginia.

Conrad, Mark, Francois Dion, and Sam Yagar (1998), "Real-time traffic signal optimization with transit priority: Recent advance in the signal priority procedure for optimization in real-time model," Transportation Research Record 1634, Washington, D.C.: 100-109.

Daniel, J., E. Lieberman and R. Srinivasan (2004), "Assess impacts and benefits of traffic signal priority for buses", FHWA report number: FHWA-NJ-2004-013

Dantzig, George B. and Mukund N. Thapa (1997). Linear programming I: Introduction. Springer-Verlag New York, Inc. New York, NY 10010.

Deeter, D., H.M. Zarean, and D. Register (2001). Rural ITS Toolbox, Section 3: Emergency Services, Subsection 3.1 Emergency Vehicle Traffic Signal Preemption. On line at: http://www.itsdocs.fhwa.dot.gov//JPODOCS/REPTS_TE/13477.html

Dion, F., H. Rakha and Y. Zhang (2004), " Evaluation of potential transit signal priority benefits along a fixed-time signalized arterial", Journal of Transportation Engineering, Vol. 130, No. 3, pp: 294-303

Dion, F. and S. Yagar (1996), "Real-time control of signalized networks-different approaches for different needs," In: Proceedings of the 8th International Conference on Road Traffic Monitoring and control, London: 56-59.

Donati, F., V. Mauro, G. Roncolini and M. Vallauri (1984), "A hierarchical decentralized traffic light control system: the first realization," In: IFAC 9th World Congress, Budapest, Hungary, Vol. II, 11G/A-1.

Fang, F. C. and L. Elefteriadou (2005). Optimal and adaptive signal control of diamond interchanges. Presented at 84th TBR Annual meeting, Jan. 9-13, 2005, Washington, D.C.

Federal Highway Administration (1996). Traffic signal control system handbook, Publication number: FHWA-SA-95-032, Washington, D.C.

Federal Highway Administration (2003). Manual on uniform traffic control devices, Washington, D.C.

Festa, D. C. et al. (2001). Experimental analysis of different simulation models for motorway traffic flow. 2001 IEEE Intelligent Transportation Systems Conference Proceedings. Oakland, CA, USA, August 25-29, 2001, pp: 675-680.

Florian, M. et al. (2001). A hybrid optimization-Mesosopic simulation dynamic traffic

assignment model. 2001 IEEE Intelligent Transportation Systems Conference Proceedings. Oakland, CA, USA, August 25-29, 2001, pp: 118-121.

Fox, K. et al. (1998). Selected vehicle priority in the UTM environment, UTM01 project report 1-part A. Institute for Transport Studies, University of Leeds.

Garber, N.J. and L.A. Hoel (2002). Traffic and highway engineering, Third Edition, Brooks/Cole Publishing Company, pp.301-304.

Gartner, N.H. (1983), "OPAC: A demand responsive strategy for traffic signal control," Transportation Research Record 906, Transportation Research Board: 75-81.

Gartner, N. et al. (1991). "Evaluation of optimized policies for adaptive control strategy," Transportation Research Record 1324, Transportation Research Board: 105-114.

Gartner, N. et al. (2001). Implementation of the OPAC adaptive control strategy in a traffic signal network. 2001 IEEE Intelligent Transportation Systems Conference Proceedings. Oakland, CA, USA, Aug. 25-29, 2001, pp:195-200.

Gartner, Nathan H. et al. (2002). "Optimized policies for adaptive control strategies in real-time traffic adaptive control systems," Transportation Research Record 1811, Transportation Research Board: 148-156.

Grafton, R. B. and G. F. Newell (1967). Optimal policies for the control of an under saturated intersection. Proc. 3rd International Symposium on Theory of Traffic Flow. (L. C. Edie et al., Eds.) American Elsevier, NY.

Hanbali, R.M. & C.J. Fornal (2004). "Emergency vehicle preemption signal timing: constraints and solutions". ITE Journal, Vol. 74, No. 7, pp: 32-40.

Henry, J.J. and J.L. Farges (1989), "PRODNY," In: Proceedings of the 6th IFAC-IFIP-IFORS Symposium on Control, Computers, and Communication on Transportation. Paris, France. Oxford, Pergamon: 253-255.

Hillier, Frederic S. et al. (1967). Introduction to operations research. Holden-Day, Inc.

Hunt, P.B., D.I. Robertson, R.D. Bretherton and R.I. Winton (1981), "SCOOT – a traffic responsive method of coordinating signals," Transportation and Road Research Laboratory, Crowthorn, U.K., Report 1014.

Hunt, P.B., D.I. Robertson, and R.D. Bretherton (1982), "The SCOOT on-line traffic optimization technique," Traffic Engineering and Control, 23:190-192.

Ignizio, James P. and Tom M. Cavalier (1994). Linear Programming. Prentice-Hall, Inc Englewood Cliffs, New Jersey 07632.

ITE (2005). Guideline for the activation, modification, or removal of traffic control signals: An ITE recommended practice.

Jaganathan, R. and Ata. M. Khan (2001), "Methodology for the assessment of traffic adaptive control system," ITE Journal, Vol. 71, No. 6, pp. 28-33.

Janos, Melanie and peter G. Furth (2002), "Bus priority with highly interruptible traffic signal control: simulation of San Juan's Avenida Poncede Leon," Transportation Research Record 1811, Transportation Research Board, Washington, D. C. pp. 157-165.

Khisty, C.J. and B. Kent Lall (1998). Transportation engineering, An introduction, Second Edition, Prentice Hall, pp.271-317.

Kutz. M. (2004). Handbook of transportation engineering. New York : McGraw-Hill

Kwon, E.L. and Sangho Kim (2003). Development of dynamic route clearance strategies for emergency vehicle operations, phase I. Center for Transportation Studies, University of Minnesota.

Lewis, Frank L. and Vssilis L. Syrmos (1995). Optimal control. 2nd edition. John Wiley & Sons, Inc.

Lin, Wei-Hua (2001), "An enhanced 0-1 mixed integer LP formulation for the traffic signal problem," the Proceedings of 2001 IEEE Intelligent Transportation Systems Conference-Oakland (CA):189-194.

Louisell, C; J. Collura, D. Teodorovic, S. Tignor (2004), "Simple worksheet method to evaluate emergency vehicle preemption and its impacts on safety", Transportation Research Record 1867, Transportation Research Board: 151-162

Lowrie, P.R. (1982), "SCATS: The Sydney Co-ordinated Adaptive traffic system," Proc. IEE Int. Conf. on Road Traffic Signaling, London, UK, pp.67-70.

Mauro, V. (1991), "Road network control," Concise Encyclopedia of Traffic & Transportation Systems, Pergamon Press: 361-366.

Mauro, V. and C. Di Taranto (1989), "UTOPIA," In: Proceedings of the 6th IFAC/IFIP/IFORS Symposium on Control, Computers, Communications in Transportation, Paris, France: 245-252.

Miller, A.J. (1963), "A computer control system for traffic networks", Proceedings of the 2nd international symposium on the theory of traffic flow, London: 200-220.

Mussa, R. and M.F. Selekwa (2004), "Development of an optimal transition procedure for coordinated traffic signals", *Journal of Advances in Transportation Studies*, Vol. 3, pp: 57-70

Nelson, E. and D. Bullock (2000), "Impact evaluation of emergency vehicle preemption on signalized corridor operation," *Transportation Research Record 1727*, Transportation Research Board, Washington, D.C.: 1-11.

Nelson, P. (2003). Dynamic route clearance: Saving time, saving lives. On line at <http://www.its.umn.edu/sensor/2003/spring/signals.html>.

Newell, G.F (1998), "The rolling horizon scheme of traffic signal control", *Transportation Research - A*, Vol.32, No. 1: 39-44.

Ozbay, K. and P. Kachroo (1999). Incident management in intelligent transportation systems. Artech House, Inc.

Paral, J.M. (2004). "ITS standards for transit signal priority and emergency vehicle preemption in Cleveland", presented on 2004 Annual Meeting and Exhibit of Institute of Transportation Engineers, Lake Buena Vista, Florida, 08/01/2004 - 08/04/2004

Platinum One Net Inc. (2005). How the MIRT works. On line at: <http://www.skyoptics.com/howworks.htm>

Rakha, H, and Y. Zhang (2004), "Sensitivity analysis of transit signal priority impacts on operation of a signalized intersection", *Journal of Transportation Engineering*, Vol. 130, No. 6, pp. 796-804

Robertson, D. I. and R. D. Brotherhood (1974). Optimum control of an intersection for any known sequence of vehicle arrivals. Presented at 2nd IFAC/IFIP/IFORS Symposium on Traffic Control and Transportation Systems. North-Holland, Amsterdam. pp: 3017

Russell, Stuart J. et al. (1995). *Artificial intelligence: a modern approach*. Prentice Hall, Englewood Cliffs, N.J.

Schalkoff, Robert J. (1990). *Artificial intelligence: an engineering approach*. McGraw-Hill, New York.

Sen, S. (1993). Coordinated optimization of phase (COP). SIE working paper, Univ. of Arizona, Tucson, Arizona 85721.

Sen, S. and K. L. Head (1997). Controlled optimization of phases at an intersection. *Transportation Science*, Vol. 31, No. 1, pp:5-17.

Shibuya, S. et. al. (2000), "Fast emergency vehicle pre-emption systems," *Transportation Research Record* 1739, Transportation Research Board: 44-50.

Siemens (2000). Transportation Research Board (TRB) presentation on SCOOT. TRB workshop on adaptive traffic signal control systems, Washington, D.C.

Smith, David K. (1991). Dynamic Programming: a practical introduction. Ellis Horwood Limited.

Steinbachs, John (2004). "Medics say cash injection needed". The Ottawa Sun, July 20, 2004. P. 7.

Teo, K.L., C.J. Goh, and K.H. Wong (1991). A unified computational approach to optimal control problems. Longman, New York.

The Mathworks Inc. (2003). MATLAB-The language of technical computing. Version 6.5.1.199709 Release 13 (Service pack 1).

The Mathworks, Inc. (2004a). MATLAB: the language of technical computing. Ver. 7.0

The Mathworks, Inc (2004b). Simulink: simulation and model-based design. Ver. 6.2

Thompson, A. and Nicholls, J. (1997) "Signal pre-emption using traffic control networks", Traffic Technology International.

Tomar Electronics Inc. (2005). STROBECOM II optical preemption & priority control systems. On line at: <http://www.tomar.com/pdf/strobecom2005.pdf>.

Toy, C., K. Leung, L. Alvarez, R. Horowitz (2001), "Emergency Vehicle Maneuvers and Control Laws for Automated Highway Systems". California PATH program, Institute of Transportation Studies, University of California, Berkeley. Report number: UCB-ITS-PRR-2001-17

Traffic Engineers, Inc. (1991). Emergency Response Management System Study. On line at: <http://www.benefitcost.its.dot.gov/ITS/benecost.nsf>

Transportation Association of Canada (1998). Manual of uniform traffic control devices for Canada, Ottawa, Canada

Transport Canada, U.S. Department of Transportation , and Secretariat of Transportat and Communications of Mexico (2004). 2004 Emergency response guidebook-A guidebook for first responders during the initial phase of a dangerous goods/hazards materials incident.

TRB (2002). Proceedings of National Conference on Traffic Incident Management. On-line at: <http://trb.org/conferences/TIM/TIMProceedings.pdf>

TTI (Texas Transportation Institute) (2003). 2003 urban mobility report. On line at: <http://mobility.tamu.edu>

Urbanik, T., S. Beaird, D. Gettman, L. Head, D. Bullock, E. Smaglik, R. Campbell and M. Ablett (2003), "Traffic signal state transition logic using enhanced sensor information", Transportation Research Board, NCHRP 3-66 interim report.

USDOT, NHTSA (2002). Fatality analysis reporting system. Washington, D.C.

Vincent, R.A. and C.P. Young (1986), "Self-optimising traffic signal control using microprocessors – The TRRL MOVA strategy for isolated intersections," *Traffic Engineering and Control*, 27(7/8):385-387.

Vincent, Thomas L. and Walter J. Grantham (1997). Nonlinear and optimal control systems. John Wiley & Sons, Inc.

Wagner, Harvey M. (1963). Practical slants on operations research. *Harvard Business Review*, 41, pp:61-71.

Watson, S.M. (2002). Municipal emergency management procedures handbook. The Butterworth Group of Companies.

Wolshon, B. and B.H. Meehan (2003). "Emergency evacuation: ensuring safe and efficient transportation out of endangered areas". TR News, Transportation Research Board, Washington, D.C., No 224, pp: 3-9.

Yagar, S. and B. Han (1994), "A procedure for real-time signal control that considers transit interference and priority," Transportation Research, 28B: 315-331.

Zegeer, C.V. and C.B. Seiderman (2000). Designing for pedestrians. The traffic safety toolbox - A primer on traffic safety. Institute of Transportation Engineers, Washington D.C.

Appendix A Computer Program for Generation of Vehicle Arrivals

```
function [rn,arrival,turn] = vehicle_arrival(volume);
% generate random vehicle arrivals
% rn=random number, 1 for arrival and 0 for no arrival
% arrival=time of instant a vehicle arrives at an intersection
% turn=turning factors corresponding to arrivals with 1.0 for straight-through, 1.4 for
% left-turn, and 1.2 for right-turn
% volume=traffic volume in veh/hr,
% lt_rate,rt_rate=left_truning and right_turning rates (%), respectively
lt_rate = 4
rt_rate = 6
green = [60,40]
yellow = [3,3]
allred = [1,1]
cycle_number = 20
cycle_length = green(1) + green(2) + yellow(1) +yellow(2) + allred(1) +allred(2)
tt = cycle_length * cycle_number % tt=totoal time in second for simulation
q = volume / 3600
interval = 0.5 / q % time interval
tti = round(tt / interval) % total # of time interval
rnn = rand(1,tti)
rn = (rnn >= 0.5) % rn=1 if rnn>0.5 otherwise rn=0
tva = sum(rn) % total vehicle arrivals
arrival = find(rn == 1); % indices of rn where an arrival is produced
arrival = (arrival - 1) * interval
% time of instant a vehicle arrives at an intersection, and the possibly earliest one
arrives
% at 0 sec.
turn = ones(1,tva); % facor for straight-through vehicles
rnn = round(100 * rand(1,tva));
left_turn = find(rnn <= lt_rate)
% indices of arrival where the vehicle is left_truning
left_turn_total = length(left_turn)
turn(left_turn) = 1.4; % facor for left-turn vehicles
right_turn = find((rnn > lt_rate) & (rnn <= lt_rate + rt_rate))
% indices of arrival where the vehicle is right_turning
right_turn_total = length(right_turn)
turn(right_turn) = 1.2; % facor for right-turn vehicles
% save vehicle arrival and turn as delimited ASCII files using:
save arrival_approach#.dat arrival -ascii
save turn_approach#.dat turn -ascii
```


Appendix B Computer Program for Determination of EV detection

distance

```
function [distance] = ev_detector_distance(lq);
% input queue length (lq) to determine minimum EV detection distance
% lq=average queue length (veh) on EV approach under normal operation
% distance=EV-detector distance (m) away from the intersection
% EV operating speed SEV = speed_ev = 15 (m/s or 54 Km/h)
% general vehicle mean Smean = speed_mean = 12 (m/s or 43.2 Km/h)
% discharge speed Sdis = speed_dis = 4.44 (m/s or 16 Km/h)
% maximum greens for main road gmax(1) = 60 sec, and side road gmax(2) = 60 sec
% Start up delay t_startup = 2 sec
% Safety time interval sti = 3 sec
% All red =T red-1 = T red-2 =1 sec
% Yellow intervals =Tyellow = T'yellow = 3 sec
% Average spacing between successive vehicles spacing = 8 metre

% L1 >= [STI+ tstartup + LQ/ Sdis ] * SEV
% L2 >= [STI+ tstartup + LQ/ Sdis + Tred-2 - t'red] * SEV and using t'red=0
% L3 >= [STI+ tstartup + LQ/ Sdis + T'yellow + Tred-2] / * SEV
% L4 >= [STI+ tstartup + LQ/ Sdis + T'yellow - t'yellow + T red-2] / * SEV and
using % t'yellow =0

speed_ev = 15
speed_mean = 12
speed_dis = 4.44
gmin = [7,10]
gmax = [60,60]
t_startup = 2
sti = 3
all_red = [1,1]
yellow = [3,3]
spacing = 8

l(1) = (sti + t_startup + spacing * lq / speed_dis ) * speed_ev
l(2) = l(1) + ( all_red(2) -0 ) * speed_ev
l(3) = l(1) + ( yellow(2) + all_red(2) ) * speed_ev
l(4) = l(1) + ( yellow(2) - 0 + all_red(2) ) * speed_ev
l_max = max(l)
distance = l_max
```

Appendix C Computer Program for Normal Operation

```
function [queue_length,total_produced,total_discharged] =  
queue_length_normal(volume,tt,rn,arrival,turn)  
% generate queue length for normal control scenario  
% use green = [60,40] for main road and green = [40,60] for side road  
% volume=traffic volume in veh/hr, tt=total time in second  
% rn=random vehicle arrivals, 1=arrival and 0=no arrival  
% arrival=indices of rn where an arrival is produced  
% turn=turning factor corresponding to each arrival with straight-through (1.0),  
% left-turn (1.4), or right-turn (1.2)  
  
% green = [60,40]  
% used for main road:approaches 1 and 3  
green = [40,60]  
% used for side road:approaches 2 and 4  
yellow = [3,3]  
all_red = [1,1]  
cycle_length = green(1) + yellow(1) + all_red(1) + green(2) + yellow(2) + all_red(2)  
q = volume / 3600  
interval = 0.5 / q % vehicle arrival interval  
saturation = 2000 / 3600 % in veh/sec  
discharge_rate_green = 1 / saturation  
% for straight-through vehicles during green in sec/veh  
discharge_rate_yellow = 2 * discharge_rate_green  
% for straight-through vehicles during green in sec/veh  
yellow_equival_green = yellow(1) * discharge_rate_green / discharge_rate_yellow  
% transfer yellow interval to equivalent green discharging vehicles  
cycle_number = fix(tt / cycle_length) % # of cycles for simulation  
queue_length = zeros(1,cycle_number)  
queue_last_cycle = 0  
  
total_produced = zeros(1,cycle_number)  
total_discharged = zeros(1,cycle_number)  
  
pointer = 1  
t = (arrival(pointer) - 1) / (2*q)  
% arrival time in sec, and the possibly earliest one arrives at 0 sec  
queue_start=pointer  
  
for cycle = 1:cycle_number  
    if t >= cycle * cycle_length % wait to clear in next cycle
```

```

        continue
    end
    queue_green = queue_last_cycle
    queue_red = 0
    discharged_green = 0
    green_time_available = green(1) + yellow_equival_green

    if queue_green > 0 % processing queue from last cycle
        for k = 1:queue_green
            green_time_available = green_time_available - turn(queue_start) *
            discharge_rate_green
            if green_time_available < 0
                break
            end
            discharged_green = k
        end % for
        total_discharged(cycle) = total_discharged(cycle) + discharged_green
        queue_start = queue_start + discharged_green
        queue_green = queue_green - discharged_green
    end % if queue_green > 0

    while t < cycle * cycle_length % processing arrivals during this cycle
        total_produced(cycle) = total_produced(cycle) + 1
        t_quotient = fix(t / cycle_length)
        t_elapsed = t - cycle_length * t_quotient
        % time elapsed from the beginning of green interval
        if (t_elapsed >= 0) && (t_elapsed < green(1) + yellow_equival_green)
            indicator = 'green or first half yellow'
            if queue_green > 0
                % there already exists queue that cannot be discharged during this green period
                queue_green = queue_green + 1
            elseif queue_green == 0 % deal with the situation that no queue exists
                if (green(1) - t_elapsed) < green_time_available
                    green_time_available = green(1) - t_elapsed
                end
                green_time_available = green_time_available - turn(queue_start) *
                discharge_rate_green
                if green_time_available < 0 % queue produced
                    queue_green = queue_green + 1
                elseif green_time_available >= 0
                    discharged_green = discharged_green + 1
                    total_discharged(cycle) = total_discharged(cycle) + 1
                    queue_start = queue_start + 1
                end
            end
        end % if queue_green > 0
    end

```

```

elseif (t_elapsed >= green(1) + yellow_equivall_green)
    indicator = 'second half yellow or red'
    queue_red = queue_red + 1
end % indicator is 'green' or 'red'
if pointer >= length(arrival)
    break
end
pointer = pointer + 1
t = (arrival(pointer) - 1) / (2*q)
end % while
undischarged(cycle) = queue_green % undischarged during this green interval
queue_length(cycle) = queue_green + queue_red
queue_last_cycle = queue_length(cycle)
end % for

```

Appendix D Computer Program for Fixed-held Time Transitioning

```
% open a file to record results
[fid,message] = fopen('queue_length.dat', 'w')
fprintf(fid,'queue_length_fixed_time \n')
status = fclose(fid)
if status == -1 % =0 if successful closed
    disp('error')
    return
end
% then run one of the following to produce queue on corresponding approach
% queue_length_fixed_time_1 for approach 1
% queue_length_fixed_time_2 for approach 2 or 4
% queue_length_fixed_time_3 for approach 3
for instant_entry=1:6
    t_fixed=50 % or another value as required
    for cycle_entry=5:10
        queue_length_fixed_time_approach#(volume,tt,rn,arrival,turn,t_fixed,
        cycle_entry, instant_entry);
    end
    t_fixed=30
    for cycle_entry=5:10
        queue_length_fixed_time_approach# (volume,tt,rn,arrival,turn,t_fixed,
        cycle_entry, instant_entry);
    end
end

%-----
function [queue_length,successful] =
queue_length_fixed_time_1(volume,tt,rn,arrival,turn,t_fixed,cycle_entry,instant_entry);
% generate queue length on EV approach (approach 1) under fixed time strategy for
% transition 1
% successful='Yes!' if EV can cross the intersection, otherwise successful='No!'
% volume=traffic volume in veh/hr, tt=total time in second
% rn=random vehicle arrivals, 1=arrival and 0=no arrival
% arrival=indices of rn where an arrival is produced
% turn=turning factor corresponding to each arrival with straight-through (1.0),
% left-turn (1.4), or right-turn (1.2)
% t_fixed=fixed time period held for EV
% cycle_entry=the number of cycle in which EV arrives
% instant_entry=the time instant at which EV arrives, 1-6 for at 0,green(1)/2,green(1),
% green(1)+yellow(1),green(1)+yellow(1)+all_red(1),and
```

```

% green(1)+yellow(1)+ all_red(1) +green(2)/2, respectively.
green = [60,40]
yellow = [3,3]
all_red = [1,1]
cycle_length = green(1) + yellow(1) + all_red(1) + green(2) + yellow(2) + all_red(2)
q = volume / 3600
interval = 0.5 / q % vehicle arrival interval
saturation = 2000 / 3600 % in veh/sec
discharge_rate_green = 1 / saturation
% for straight-through vehicles during green in sec/veh
discharge_rate_yellow = 2 * discharge_rate_green
% for straight-through vehicles during green in sec/veh
yellow_equival_green = yellow(1) * discharge_rate_green / discharge_rate_yellow
% transfer yellow interval to equivalent green discharging vehicles
cycle_number = fix(tt / cycle_length) % # of cycles for testing
queue_length = zeros(1,cycle_number)
queue_last_cycle = 0

total_produced = zeros(1,cycle_number)
total_discharged = zeros(1,cycle_number)

pointer = 1
t = (arrival(pointer) - 1) / (2*q)
% arrival time in sec, and the possibly earliest one arrives at 0 sec
queue_start=pointer

for cycle = 1:cycle_number
    if t >= cycle * cycle_length % wait to clear in next cycle
        continue
    end
    queue_green = queue_last_cycle
    queue_red = 0
    discharged_green = 0
    green_time_available = green(1) + yellow_equival_green

    if (cycle==cycle_entry)
        switch instant_entry
            case 1
                entry_point = 0
                % time point away from the begining of green at which EV enters
                green_time_available = 0
            case 2
                entry_point = green(1) / 2
                green_time_available = green_time_available - green(1) / 2
            case 3

```

```

        entry_point = green(1)
    case 4
        entry_point = green(1) + yellow(1)
    case 5
        entry_point = green(1) + yellow(1) + all_red(1) % in red period
    case 6
        entry_point = green(1) + yellow(1) + all_red(1) + green(2)/2 % in red period
    end % switch
    if (instant_entry==1)
        % EV in coming at the begining of green and no need to calculate other cycles
        break
    end
end % if

if queue_green > 0 % processing queue from last cycle
    for k = 1:queue_green
        green_time_available = green_time_available - turn(queue_start) *
            discharge_rate_green
        if green_time_available < 0
            break
        end
        discharged_green = k
    end % for
    total_discharged(cycle) = total_discharged(cycle) + discharged_green
    queue_start = queue_start + discharged_green
    queue_green = queue_green - discharged_green
end % if queue_green > 0

while t < cycle * cycle_length % processing arrivals during this cycle
    t_quotient = fix(t / cycle_length)
    t_elapsed = t - cycle_length * t_quotient
    % time elapsed from the begining of green interval
    if (cycle==cycle_entry) && (t_elapsed>=entry_point)
        % Ev is coming and no need to calculate other cycles
        break
    end
    total_produced(cycle) = total_produced(cycle) + 1
    if (t_elapsed >= 0) && (t_elapsed < green(1) + yellow_equival_green)
        indicator = 'green or first half yellow'
        if queue_green > 0
            % there already exists queue that cannot be discharged during this green period
            queue_green = queue_green + 1
        elseif queue_green == 0 % deal with the situation that no queue exists
            if (green(1) - t_elapsed) < green_time_available
                green_time_available = green(1) - t_elapsed
            end
        end
    end
end

```

```

        end
        green_time_available = green_time_available - turn(queue_start) *
        discharge_rate_green
        if green_time_available < 0 % queue produced
            queue_green = queue_green + 1
        elseif green_time_available >= 0
            discharged_green = discharged_green + 1
            total_discharged(cycle) = total_discharged(cycle) + 1
            queue_start = queue_start + 1
        end
    end % if queue_green > 0
elseif (t_elapsed >= green(1) + yellow_equival_green)
    indicator = 'second half yellow or red'
    queue_red = queue_red + 1
end % indicator is 'green' or 'red'
if pointer >= length(arrival)
    break
end
pointer = pointer + 1
t = (arrival(pointer) - 1) / (2*q)
end % while
undischarged(cycle) = queue_green % undischarged during this green interval
queue_length(cycle) = queue_green + queue_red
queue_last_cycle = queue_length(cycle)
if cycle==cycle_entry % Ev is coming and no need to calculate other cycles
    break
end
end % for

queue = queue_last_cycle % queue from normal operation
discharged_green = 0
green_time_available = t_fixed

if queue > 0 %discharge queue from normal operation
    total_green_time_needed = 0
    for k = 1:queue
        total_green_time_needed = total_green_time_needed + turn(queue_start + k - 1) *
        discharge_rate_green
    end % for
    for k = 1:queue
        green_time_available = green_time_available - turn(queue_start + k - 1) *
        discharge_rate_green
        if green_time_available < 0
            break
        end
    end
end

```



```

        discharged_green = k
    end % for
    discharged_for_ev = discharged_green
    queue_start = queue_start + discharged_green
    queue = queue - discharged_green
end % if queue > 0

if (queue > 0)
    successful = 'No!' % more time is needed
    flag = -1
    time_needed = total_green_time_needed - t_fixed
    % return
elseif (queue == 0)
    successful = 'Yes!'
    flag = 1
    time_spare = green_time_available
    start = (cycle_entry - 1) * cycle_length + entry_point
    while (t >= start) && (t <= start + t_fixed)
        % produce vehicles during held time,t_fixed
        queue = queue + 1
        if pointer >= length(arrival)
            break
        end
        pointer = pointer + 1
        t = (arrival(pointer) - 1) / (2*q)
    end % while
    queue_length(cycle_entry) = queue % produced during t_fixed
end %if

[fid,message] = fopen('queue_length.dat', 'a');
fprintf(fid,'instant_entry = %-3d ',instant_entry);
fprintf(fid,' cycle_entry = %-3d ',cycle_entry);
if flag == -1
    fprintf(fid,' time needed = %-6.2f',time_needed);
    fprintf(fid,' queue undischarged on approach 1 = %-6.2f\n',queue);
elseif flag == 1
    fprintf(fid,' time spare = %-6.2f',time_spare);
    fprintf(fid,' queue formed on approach 1 = %-6.2f\n',queue);
end
status = fclose(fid);
if status == -1 % =0 if successful closed
    disp('error')
    return
end

```

```

%-----
function [queue_length] =
queue_length_fixed_time_2(volume,tt,rn,arrival,turn,t_fixed,cycle_entry,instant_entry);
% generate queue length on cross road under fixed time strategy for transition 1
% Two runs are needed. one for approach 2 and another for approach 4
% volume=traffic volume in veh/hr, tt=total time in second
% rn=random vehicle arrivals, 1=arrival and 0=no arrival
% arrival=indices of rn where an arrival is produced
% turn=turning factor corresponding to each arrival with straight-through (1.0),
% left-turn (1.4), or right-turn (1.2)
% t_fixed=fixed time period held for EV
% cycle_entry=the number of cycle in which EV arrives
% instant_entry=the time instant at which EV arrives, 1-6 for at 0,green(1)/2,green(1),
% green(1)+yellow(1),green(1)+yellow(1)+all_red(1),and
% green(1)+yellow(1)+all_red(1)+green(2)/2, respectively.

green = [40,60] % for cross road
yellow = [3,3]
all_red = [1,1]
cycle_length = green(1) + yellow(1) + all_red(1) + green(2) + yellow(2) + all_red(2)
q = volume / 3600
interval = 0.5 / q % vehicle arrival interval
saturation = 2000 / 3600 % in veh/sec
discharge_rate_green = 1 / saturation
% for straight-through vehicles during green in sec/veh
discharge_rate_yellow = 2 * discharge_rate_green
% for straight-through vehicles during yellow in sec/veh
yellow_equiv_green = yellow(1) * discharge_rate_green / discharge_rate_yellow
% transfer yellow interval to equivalent green discharging vehicles
cycle_number = fix(tt / cycle_length) % # of cycles for testing
queue_length = zeros(1,cycle_number)
queue_last_cycle = 0

total_produced = zeros(1,cycle_number)
total_discharged = zeros(1,cycle_number)

pointer = 1
t = (arrival(pointer) - 1) / (2*q)
% arrival time in sec, and the possibly earliest one arrives at 0 sec
queue_start=pointer

for cycle = 1:cycle_number
    if t >= cycle * cycle_length % wait to clear in next cycle
        continue
    end

```

```

queue_green = queue_last_cycle
queue_red = 0
discharged_green = 0
green_time_available = green(1) + yellow_equival_green

if (cycle==cycle_entry)
    switch instant_entry
        case 1
            entry_point = green(1) + yellow(1) + all_red(1) % in red period
            % time point away from the beginning of green at which EV enters on EV
road
        case 2
            entry_point = green(1) + yellow(1) + all_red(1) + green(2) / 2
        case 3
            entry_point = green(1) + yellow(1) + all_red(1) + green(2)
        case 4
            entry_point = green(1) + yellow(1) + all_red(1) + green(2) + yellow(2)
        case 5
            entry_point = 0
            green_time_available = 0
        case 6
            entry_point = green(1) / 2 % only this case in green period
            green_time_available = green_time_available - green(1) / 2
    end % switch
end % if

if queue_green > 0 % processing queue from last cycle
    for k = 1:queue_green
        green_time_available = green_time_available - turn(queue_start) *
        discharge_rate_green
        if green_time_available < 0
            break
        end
        discharged_green = k
    end % for
    total_discharged(cycle) = total_discharged(cycle) + discharged_green
    queue_start = queue_start + discharged_green
    queue_green = queue_green - discharged_green
end % if queue_green > 0

while t < cycle * cycle_length % processing arrivals during this cycle
    t_quotient = fix(t / cycle_length)
    t_elapsed = t - cycle_length * t_quotient
    % time elapsed from the beginning of green interval
    if (cycle==cycle_entry) && (t_elapsed >= entry_point)

```

```

    % Ev is coming and no need to calculate other cycles
    break
end
total_produced(cycle) = total_produced(cycle) + 1
if (t_elapsed >= 0) && (t_elapsed < green(1) + yellow_equival_green)
    indicator = 'green or first half yellow'
    if queue_green > 0
        % there already exists queue that cannot be discharged during this green period
        queue_green = queue_green + 1
    elseif queue_green == 0 % deal with the situation that no queue exists
        if (green(1) - t_elapsed) < green_time_available
            green_time_available = green(1) - t_elapsed
        end
        green_time_available = green_time_available - turn(queue_start) *
        discharge_rate_green
        if green_time_available < 0 % queue produced
            queue_green = queue_green + 1
        elseif green_time_available >= 0
            discharged_green = discharged_green + 1
            total_discharged(cycle) = total_discharged(cycle) + 1
            queue_start = queue_start + 1
        end
    end % if queue_green > 0
elseif (t_elapsed >= green(1) + yellow_equival_green)
    indicator = 'second half yellow or red'
    queue_red = queue_red + 1
end % indicator is 'green' or 'red'
if pointer >= length(arrival)
    break
end
pointer = pointer + 1
t = (arrival(pointer) - 1) / (2*q)
end % while
undischarged(cycle) = queue_green % undischarged during this green interval
queue_length(cycle) = queue_green + queue_red
queue_last_cycle = queue_length(cycle)
if cycle == cycle_entry % EV is coming and no need to calculate other cycles
    break
end
end % for

queue = queue_last_cycle % queue from normal operation
red_time_imposed = t_fixed

start = (cycle_entry - 1) * cycle_length + entry_point

```

```

while (t >= start) && (t <= start + red_time_imposed)
    % produce vehicles during held time,t_fixed
    queue = queue + 1
    if pointer >= length(arrival)
        break
    end
    pointer = pointer + 1
    t = (arrival(pointer) - 1) / (2*q)
end % while
queue_length(cycle_entry) = queue % produced during t_fixed

[fid,message] = fopen('queue_length.dat', 'a');
fprintf(fid,'instant_entry = %-3d ',instant_entry);
fprintf(fid,' cycle_entry = %-3d ',cycle_entry);
fprintf(fid,' queue on approach 2/4 = %-6.2g \n',queue_length(cycle_entry));
status = fclose(fid);
if status == -1 % =0 if successful closed
    disp('error')
    return
end

%-----
function [queue_length] =
queue_length_fixed_time_3(volume,tt,rn,arrival,turn,t_fixed,cycle_entry,instant_entry);
% generate queue length on approach 3(opposite to EV approach) under fixed time
% strategy for transition 1
% volume=traffic volume in veh/hr, tt=total time in second
% rn=random vehicle arrivals, 1=arrival and 0=no arrival
% arrival-indices of rn where an arrival is produced
% turn=turning factor corresponding to each arrival with straight-through (1.0),
% left-turn (1.4), or right-turn (1.2)
% t_fixed=fixed time period held for EV
% cycle_entry=the number of cycle in which EV arrives
% instant_entry=the time instant at which EV arrives, 1-6 for at 0,green(1)/2,green(1),
% green(1)+yellow(1),green(1)+yellow(1)+all_red(1),and
% green(1)+yellow(1)+all_red(1)+green(2)/2, respectively.

green = [60,40]
yellow = [3,3]
all_red = [1,1]
cycle_length = green(1) + yellow(1) + all_red(1) + green(2) + yellow(2) + all_red(2)
q = volume / 3600
interval = 0.5 / q % vehicle arrival interval
saturation = 2000 / 3600 % in veh/sec
discharge_rate_green = 1 / saturation

```

```

% for straight-through vehicles during green in sec/veh
discharge_rate_yellow = 2 * discharge_rate_green
% for straight-through vehicles during green in sec/veh
yellow_equiv_green = yellow(1) * discharge_rate_green / discharge_rate_yellow
% transfer yellow interval to equivalent green discharging vehicles
cycle_number = fix(tt / cycle_length) % # of cycles for testing
queue_length = zeros(1,cycle_number)
queue_last_cycle = 0

total_produced = zeros(1,cycle_number)
total_discharged = zeros(1,cycle_number)

pointer = 1
t = (arrival(pointer) - 1) / (2*q)
% arrival time in sec, and the possibly earliest one arrives at 0 sec
queue_start=pointer

for cycle = 1:cycle_number
    if t >= cycle * cycle_length % wait to clear in next cycle
        continue
    end
    queue_green = queue_last_cycle
    queue_red = 0
    discharged_green = 0
    green_time_available = green(1) + yellow_equiv_green

    if (cycle==cycle_entry)
        switch instant_entry
            case 1
                entry_point = 0
                % time point away from the begining of green at which EV enters
                green_time_available = 0
            case 2
                entry_point = green(1) / 2
                green_time_available = green_time_available - green(1) / 2
            case 3
                entry_point = green(1)
            case 4
                entry_point = green(1) + yellow(1)
            case 5
                entry_point = green(1) + yellow(1) + all_red(1) % in red period
            case 6
                entry_point = green(1) + yellow(1) + all_red(1) + green(2)/2 % in red period
        end % switch
        if (instant_entry==1)

```

```

        % EV is coming at the begining of green and no need to calculate other cycles
        break
    end
end % if

if queue_green > 0 % processing queue from last cycle
    for k = 1:queue_green
        green_time_available = green_time_available - turn(queue_start) *
            discharge_rate_green
        if green_time_available < 0
            break
        end
        discharged_green = k
    end % for
    total_discharged(cycle) = total_discharged(cycle) + discharged_green
    queue_start = queue_start + discharged_green
    queue_green = queue_green - discharged_green
end % if queue_green > 0

while t < cycle * cycle_length % processing arrivals during this cycle
    t_quotient = fix(t / cycle_length)
    t_elapsed = t - cycle_length * t_quotient
    % time elapsed from the begining of green interval
    if (cycle == cycle_entry) && (t_elapsed >= entry_point)
        % EV is coming and no need to calculate other cycles
        break
    end
    total_produced(cycle) = total_produced(cycle) + 1
    if (t_elapsed >= 0) && (t_elapsed < green(1) + yellow_equival_green)
        indicator = 'green or first half yellow'
        if queue_green > 0
            % there already exists queue that cannot be discharged during this green period
            queue_green = queue_green + 1
        elseif queue_green == 0 % deal with the situation that no queue exists
            if (green(1) - t_elapsed) < green_time_available
                green_time_available = green(1) - t_elapsed
            end
            green_time_available = green_time_available - turn(queue_start) *
                discharge_rate_green
            if green_time_available < 0 % queue produced
                queue_green = queue_green + 1
            elseif green_time_available >= 0
                discharged_green = discharged_green + 1
                total_discharged(cycle) = total_discharged(cycle) + 1
                queue_start = queue_start + 1
            end
        end
    end
end

```

```

        end
        end % if queue_green > 0
    elseif (t_elapsed >= green(1) + yellow_equival_green)
        indicator = 'second half yellow or red'
        queue_red = queue_red + 1
    end % indicator is 'green' or 'red'
    if pointer >= length(arrival)
        break
    end
    pointer = pointer + 1
    t = (arrival(pointer) - 1) / (2*q)
end % while
undischarged(cycle) = queue_green % undischarged during this green interval
queue_length(cycle) = queue_green + queue_red
queue_last_cycle = queue_length(cycle)
if cycle==cycle_entry % EV is coming and no need to calculate other cycles
    break
end
end % for

queue = queue_last_cycle % queue from normal operation
discharged_green = 0
green_time_available = t_fixed

if queue > 0 % processing queue from normal operation
    total_green_time_needed = 0
    for k = 1:queue
        total_green_time_needed = total_green_time_needed + turn(queue_start + k - 1) *
            discharge_rate_green
    end % for
    for k = 1:queue
        green_time_available = green_time_available - turn(queue_start + k - 1) *
            discharge_rate_green
        if green_time_available < 0
            break
        end
        discharged_green = k
    end % for
    discharged_for_ev = discharged_green
    queue_start = queue_start + discharged_green
    queue = queue - discharged_green
end % if queue > 0

start = (cycle_entry - 1) * cycle_length + entry_point
while (t >= start) && (t <= start + t_fixed) % produce vehicles during held time,t_fixed

```



```

if queue > 0
    % there already exists queue that cannot be discharged during this green period
    queue = queue + 1
elseif queue == 0 % deal with the situation that no queue exists
    if (start + t_fixed - t) < green_time_available
        green_time_available = start + t_fixed - t
    end
    green_time_available = green_time_available - turn(queue_start) *
    discharge_rate_green
    if green_time_available < 0 % queue produced
        queue = queue + 1
    elseif green_time_available >= 0
        discharged_green = discharged_green + 1
        queue_start = queue_start + 1
    end
end % if queue > 0
if pointer >= length(arrival)
    break
end
pointer = pointer + 1
t = (arrival(pointer) - 1) / (2*q)
end % while
queue_length(cycle_entry) = queue % produced at the end of t_fixed period

[fid,message] = fopen('queue_length.dat', 'a');
fprintf(fid,'instant_entry = %-3d ',instant_entry);
fprintf(fid,' cycle_entry = %-3d ',cycle_entry);
fprintf(fid,' queue on approach 3 = %-6.2g \n',queue_length(cycle_entry));
status = fclose(fid);
if status == -1 % =0 if successful closed
    disp('error')
    return
end

```

Appendix E Computer Program for Smooth Transitioning

```
% open a file to record results
[fid,message] = fopen('queue_length.dat', 'w')
fprintf(fid,'queue_length_smooth \n')
status = fclose(fid)
if status == -1 % =0 if successful closed
    disp('error')
    return
end
% then run one of the following to produce queue on corresponding approach
% queue_length_smooth_1 for approach 1
% queue_length_smooth_2 for approach 2 or 4
% queue_length_smooth_3 for approach 3
for instant_entry=1:6
    t_fixed=50 % or another value as required
    for cycle_entry=5:10
        queue_length_smooth_Approach#(volume,tt,rn,arrival,turn,t_fixed,
        cycle_entry, instant_entry);
    end
end

%-----
function [queue_length,successful] =
queue_length_smooth_1(volume,tt,rn,arrival,turn,t_fixed,cycle_entry,instant_entry);
% generate queue length on EV road (approach 1) under smooth strategy after 50 sec.
% fixed held time for transition 1
% successful='Yes!' if EV can cross the intersection, otherwise successful='No!'
% volume=traffic volume in veh/hr, tt=total time in second
% rn=random vehicle arrivals, 1=arrival and 0=no arrival
% arrival=indices of rn where an arrival is produced
% turn=turning factor corresponding to each arrival with straight-through (1.0),
% left-turn (1.4), or right-turn (1.2)
% t_fixed=fixed time period held for EV
% cycle_entry=the number of cycle in which EV arrives
% instant_entry=the time instant at which EV arrives, 1-6 for at 0,green(1)/2,green(1),
% green(1)+yellow(1),green(1)+yellow(1)+all_red(1),and
% green(1)+yellow(1)+all_red(1)+green(2)/2, respectively.

green = [60,40]
yellow = [3,3]
all_red = [1,1]
```

```

cycle_length = green(1) + yellow(1) + all_red(1) + green(2) + yellow(2) + all_red(2)
q = volume / 3600
interval = 0.5 / q % vehicle arrival interval
saturation = 2000 / 3600 % in veh/sec
discharge_rate_green = 1 / saturation
% for straight-through vehicles during green in sec/veh
discharge_rate_yellow = 2 * discharge_rate_green
% for straight-through vehicles during green in sec/veh
yellow_equival_green = yellow(1) * discharge_rate_green / discharge_rate_yellow
% transfer yellow interval to equivalent green discharging vehicles
cycle_number = fix(tt / cycle_length) % # of cycles for testing
queue_length = zeros(1,cycle_number)
queue_last_cycle = 0
gmin = [7,10]
gmax = [100,100]

total_produced = zeros(1,cycle_number)
total_discharged = zeros(1,cycle_number)

pointer = 1
t = (arrival(pointer) - 1) / (2*q)
% arrival time in sec, and the possibly earliest one arrives at 0 sec
queue_start=pointer

for cycle = 1:cycle_number
    if t >= cycle * cycle_length % wait to clear in next cycle
        continue
    end
    queue_green = queue_last_cycle
    queue_red = 0
    discharged_green = 0
    green_time_available = green(1) + yellow_equival_green

    if (cycle==cycle_entry)
        switch instant_entry
            case 1
                entry_point = 0
                % time point away from the beginning of green at which EV enters
                green_time_available = 0
            case 2
                entry_point = green(1) / 2
                green_time_available = green_time_available - green(1) / 2
            case 3
                entry_point = green(1)
            case 4

```

```

        entry_point = green(1) + yellow(1)
    case 5
        entry_point = green(1) + yellow(1) + all_red(1) % in red period
    case 6
        entry_point = green(1) + yellow(1) + all_red(1) + green(2)/2 % in red period
    end % switch
    if (instant_entry==1)
        % EV is coming at the beginning of green and no need to calculate other cycles
        break
    end
end % if

if queue_green > 0 % processing queue from last cycle
    for k = 1:queue_green
        green_time_available = green_time_available - turn(queue_start) *
            discharge_rate_green
        if green_time_available < 0
            break
        end
        discharged_green = k
    end % for
    total_discharged(cycle) = total_discharged(cycle) + discharged_green
    queue_start = queue_start + discharged_green
    queue_green = queue_green - discharged_green
end % if queue_green > 0

while t < cycle * cycle_length % processing arrivals during this cycle
    t_quotient = fix(t / cycle_length)
    t_elapsed = t-cycle_length * t_quotient
    % time elapsed from the beginning of green interval
    if (cycle==cycle_entry) && (t_elapsed>=entry_point)
        % EV is coming and no need to calculate other cycles
        break
    end
    total_produced(cycle) = total_produced(cycle) + 1
    if (t_elapsed >= 0) && (t_elapsed < green(1) + yellow_equival_green)
        indicator = 'green or first half yellow'
        if queue_green > 0
            % there already exists queue that cannot be discharged during this green
            period
                queue_green = queue_green + 1
            elseif queue_green == 0 % deal with the situation that no queue exists
                if (green(1) - t_elapsed) < green_time_available
                    green_time_available = green(1) - t_elapsed
                end
            end
        end
    end
end

```

```

    green_time_available = green_time_available - turn(queue_start) *
    discharge_rate_green
    if green_time_available < 0 % queue produced
        queue_green = queue_green + 1
    elseif green_time_available >= 0
        discharged_green = discharged_green + 1
        total_discharged(cycle) = total_discharged(cycle) + 1
        queue_start = queue_start + 1
    end
end % if queue_green > 0
elseif (t_elapsed >= green(1) + yellow_equival_green)
    indicator = 'second half yellow or red'
    queue_red = queue_red + 1
end % indicator is 'green' or 'red'
if pointer >= length(arrival)
    break
end
pointer = pointer + 1
t = (arrival(pointer) - 1) / (2*q)
end % while
undischarged(cycle) = queue_green % undischarged during this green interval
queue_length(cycle) = queue_green + queue_red
queue_last_cycle = queue_length(cycle)
if cycle==cycle_entry % EV is coming and no need to calculate other cycles
    break
end
end % for

% begin fixed held time period, transition 1
queue = queue_last_cycle % queue from normal operation
discharged_green = 0
green_time_available = t_fixed

if queue > 0 %discharge queue from normal operation
    total_green_time_needed = 0
    for k = 1:queue
        total_green_time_needed = total_green_time_needed + turn(queue_start + k - 1) *
        discharge_rate_green
    end % for
    for k = 1:queue
        green_time_available = green_time_available - turn(queue_start + k - 1) *
        discharge_rate_green
        if green_time_available < 0
            break
        end
    end
end

```

```

        discharged_green = k
    end % for
    discharged_for_ev = discharged_green
    queue_start = queue_start + discharged_green
    queue = queue - discharged_green
end % if queue > 0

if (queue > 0)
    successful = 'No!' % more time is needed
    time_needed = total_green_time_needed - t_fixed
elseif (queue == 0)
    successful = 'Yes!'
    time_spare = green_time_available
end %if
    start = (cycle_entry - 1) * cycle_length + entry_point
while (t >= start) && (t <= start + t_fixed)
    % produce vehicles during held time, t_fixed
    queue = queue + 1
    if pointer >= length(arrival)
        break
    end
    pointer = pointer + 1
    t = (arrival(pointer) - 1) / (2*q)
end % while
queue_length(cycle_entry) = queue % produced during t_fixed

% begin smooth transition 2. After this period, signal control goes back to normal
% operation.
queue_last_cycle = queue % queue from t_fixed
t_fin = entry_point + t_fixed % finishing point of EV operation, t_fin.
t_fin_quotient = fix(t_fin / cycle_length)
t_fin_elapsed = t_fin - cycle_length * t_fin_quotient % away from the beginning of
green
t_dif = green(1) - t_fin_elapsed
% time difference between the finishing point and the end of green interval of
% background cycle
if abs(t_dif) < 0.1
    indicator = 'no need to adjust, get back to normal operation immediately.'
    cycle_needed = 0
elseif t_dif > 0
    indicator = 'in green interval.'
    n_cycle_plus = ceil(t_dif / ( 0.2*green(1) + 0.2* green(2) ) ) % addition option
    t_dif_remained = t_dif
    green_plus(1) = 0
    t_trans_1 = 0

```

```

for k = 1:n_cycle_plus
    if t_dif_remained <= 0.2*green(2)
        red_plus(k) = t_dif_remained
    else
        red_plus(k) = 0.2*green(2)
    end
    if red_plus(k) + green(2) > gmax(2)
        red_plus(k) = gmax(2) - green(2)
    end
    cycle_needed_plus = k
    t_trans_1 = t_trans_1 + red_plus(k)
    t_dif_remained = t_dif_remained - red_plus(k)
    if abs(t_dif_remained) < 0.1
        break
    end
    if t_dif_remained <= 0.2*green(1)
        green_plus(k+1) = t_dif_remained
        red_plus(k+1) = 0
    else
        green_plus(k+1) = 0.2*green(1)
    end
    if green_plus(k+1) + green(1) > gmax(1)
        green_plus(k+1) = gmax(1) - green(1)
    end
    cycle_needed_plus = k+1
    t_trans_1 = t_trans_1 + green_plus(k+1)
    t_dif_remained = t_dif_remained - green_plus(k+1)
    if abs(t_dif_remained) < 0.1
        break
    end
end % for addition option
t_trans_2 = 0 % subtraction option
n_cycle_sub = ceil( (green(1) - t_dif) / (0.17*green(1) + 0.17* green(2) ) )
t_dif_remained = green(1) - t_dif
for k = 1:n_cycle_sub
    if t_dif_remained <= 0.17*green(1)
        green_sub(k) = t_dif_remained
    else
        green_sub(k) = 0.17*green(1)
    end
    if green(1) - green_sub(k) < gmin(1)
        green_sub(k) = green(1) - gmin(1)
    end
    cycle_needed_sub = k
    t_trans_2 = t_trans_2 + green_sub(k)
end

```

```

t_dif_remained = t_dif_remained - green_sub(k)
if abs(t_dif_remained) < 0.1
    red_sub(k) = 0
    break
end
if t_dif_remained <= 0.17*green(2)
    red_sub(k) = t_dif_remained
else
    red_sub(k) = 0.17*green(2)
end
if green(2)- red_sub(k) < gmin(2)
    red_sub(k) = green(2) - gmin(2)
end
t_trans_2 = t_trans_2 + red_sub(k)
t_dif_remained = t_dif_remained - red_sub(k)
if abs(t_dif_remained) < 0.1
    break
end
end % for subtraction option
else % t_dif < 0
    indicator = 'in red interval.'
    minus_t_dif = -t_dif
    n_cycle_plus = ceil( (green(2) - minus_t_dif) / (0.2*green(1) + 0.2* green(2) ) )
    t_dif_remained = green(2) - minus_t_dif
    green_plus(1) = 0
    red_plus(1) = 0
    t_trans_1 = 0
    for k = 1:n_cycle_plus % addition option
        if t_dif_remained <= 0.2*green(1)
            green_plus(k+1) = t_dif_remained
        else
            green_plus(k+1) = 0.2*green(1)
        end
        if green_plus(k+1) + green(1) > gmax(1)
            green_plus(k+1) = gmax(1) - green(1)
        end
        cycle_needed_plus = k+1
        t_trans_1 = t_trans_1 + green_plus(k+1)
        t_dif_remained = t_dif_remained - green_plus(k+1)
        if abs(t_dif_remained) < 0.1
            red_plus(k+1) = 0
            break
        end
        if t_dif_remained <= 0.2*green(2)
            red_plus(k+1) = t_dif_remained

```



```

else
    red_plus(k+1) = 0.2*green(2)
end
if red_plus(k+1) + green(2) > gmax(2)
    red_plus(k+1) = gmax(2) - green(2)
end
t_trans_1 = t_trans_1 + red_plus(k+1)
t_dif_remained = t_dif_remained - red_plus(k+1)
if abs(t_dif_remained) < 0.1
    break
end
end % for addition option
t_trans_2 = 0 % subtraction option
n_cycle_sub = ceil( minus_t_dif / (0.17*green(1) + 0.17* green(2) ) )
t_dif_remained = minus_t_dif
green_sub(1) = 0
for k = 1:n_cycle_sub
    if t_dif_remained <= 0.17*green(2)
        red_sub(k) = t_dif_remained
    else
        red_sub(k) = 0.17*green(2)
    end
    if green(2) - red_sub(k) < gmin(2)
        red_sub(k) = green(2) - gmin(2)
    end
    cycle_needed_sub = k
    t_trans_2 = t_trans_2 + red_sub(k)
    t_dif_remained = t_dif_remained - red_sub(k)
    if abs(t_dif_remained) < 0.1
        break
    end
    if t_dif_remained <= 0.17*green(1)
        green_sub(k+1) = t_dif_remained
    else
        green_sub(k+1) = 0.17*green(1)
    end
    if green(1) - green_sub(k+1) < gmin(1)
        green_sub(k+1) = green(1) - gmin(1)
    end
    cycle_needed_sub = k+1
    t_trans_2 = t_trans_2 + green_sub(k+1)
    t_dif_remained = t_dif_remained - green_sub(k+1)
    if abs(t_dif_remained) < 0.1
        red_sub(k+1) = 0
        break
    end
end

```

```

        end
    end % for subtraction option
end % if abs(t_dif) < 0.1
if abs(t_dif) > 0.1
    if (cycle_needed_plus < cycle_needed_sub) || ((cycle_needed_plus ==
        cycle_needed_sub) && (t_trans_1 < t_trans_2))
        % choose addition option
        t_transition = t_trans_1
        cycle_needed = cycle_needed_plus
        for k = 1:cycle_needed
            green_adj(k) = green_plus(k)
            red_adj(k) = red_plus(k)
        end
    elseif (cycle_needed_plus > cycle_needed_sub) || ((cycle_needed_plus ==
        cycle_needed_sub) && (t_trans_1 >= t_trans_2))
        % choose subtraction option
        t_transition = t_trans_2
        cycle_needed = cycle_needed_sub
        for k = 1:cycle_needed
            green_adj(k) = -green_sub(k)
            red_adj(k) = -red_sub(k)
        end
    else
        indicator = 'situation impossible happened.'
        return
    end
end
end
if (cycle_entry == 5) && (instant_entry == 1)
    fid = fopen('trans_1.dat','w') % 1 for standard, 2 for error
elseif (cycle_entry == 5) && (instant_entry > 1)
    fid = fopen('trans_1.dat','a') % append data to the end of the file
end
if cycle_entry == 5
    fprintf(fid,'%d \n', cycle_needed)
    if cycle_needed ~= 0
        fprintf(fid,'%6.2f \n', green_adj)
        fprintf(fid,'%6.2f \n', red_adj)
    end
    status = fclose(fid)
    if status == -1 % =0 if successful closed
        disp('error')
    end
    return
end
end
end

```

```

for k = 1: cycle_needed
    if green_adj(k) == 0
        green_time(k) = 0
    else
        green_time(k) = green(1) + green_adj(k)
    end
    if red_adj(k) == 0
        red_time(k) = 0
    else
        red_time(k) = green(2) + red_adj(k)
    end
end
if (cycle_entry == 5) && (instant_entry == 1)
    fid = fopen('trans_1_timing.dat','w') % 1 for standard, 2 for error
elseif (cycle_entry == 5) && (instant_entry > 1)
    fid = fopen('trans_1_timing.dat','a') % append data to the end of the file
end
if cycle_entry == 5
    fprintf(fid,'%d \n', cycle_needed)
    if cycle_needed > 0
        fprintf(fid,'%6.2f\n',green_time)
        fprintf(fid,'%6.2f\n',red_time)
    end
    status = fclose(fid)
    if status == -1 % =0 if successful closed
        disp('error')
        return
    end
end
if cycle_needed == 0
    queue_formed = queue_last_cycle % queue from transition 1
    [fid,message] = fopen('queue_length.dat', 'a');
    fprintf(fid,'instant_entry = %-3d ',instant_entry);
    fprintf(fid,' cycle_entry = %-3d ',cycle_entry);
    fprintf(fid,' no need to adjust and queue from t_fixed on approach 1 = %-6.2f\n',queue_formed);
    status = fclose(fid);
    if status == -1 % =0 if successful closed
        disp('error')
        return
    end
    return
end

start_point = (cycle_entry - 1) * cycle_length + entry_point + t_fixed

```

```

for cycle = 1:cycle_needed
    queue_green = queue_last_cycle % queue from transition 1
    if green_time(cycle)==0 && (red_adj(cycle) < 0 || red_adj(cycle) == 0)
        % falls in red interval
        % red_adj(cycle) < 0: in first half red; red_adj(cycle) == 0: in second half red
        % red_adj(cycle) > 0: in second half green
        amber(1) = 0
        green_time_available = 0
    else % falls in green interval
        amber(1) = yellow(1) + all_red(1)
        green_time_available = green_time(cycle) + yellow_equival_green
    end
    if (red_time(cycle) == 0) && (cycle == cycle_needed)
        % in the last cycle and no adjustment to red
        red_time_imposed = green(2) + yellow(2) + all_red(2)
    else
        red_time_imposed = red_time(cycle) + yellow(2) + all_red(2)
    end
    % produce queue after red
    queue_red = 0
    discharged_green = 0
    total_discharged = 0
    if (queue_green > 0) && (green_time_available > 0)
        % processing queue from transition 1
        for k = 1:queue_green
            green_time_available = green_time_available - turn(queue_start) * \
            discharge_rate_green
            if green_time_available < 0
                break
            end
            discharged_green = k
        end % for
        total_discharged = total_discharged + discharged_green
        queue_start = queue_start + discharged_green
        queue_green = queue_green - discharged_green
    end % if queue_green > 0
    while (t >= start_point) &&
    (t < start_point + green_time(cycle) + amber(1) + red_time_imposed)
        if t < start_point + green_time(cycle) + amber(1)
            indicator = 'green or first half yellow'
        end
        if queue_green > 0
            % there already exists queue that cannot be discharged during this
            green
            % period
            queue_green = queue_green + 1
        end
    end
end

```

```

elseif queue_green == 0 % deal with the situation that no queue exists
    if (start_point + green_time(cycle) - t) < green_time_available
        green_time_available = start_point + green_time(cycle) - t
    end
    green_time_available = green_time_available - turn(queue_start) *
discharge_rate_green
    if green_time_available < 0 % queue produced
        queue_green = queue_green + 1
    elseif green_time_available >= 0
        discharged_green = discharged_green + 1
        total_discharged = total_discharged + 1
        queue_start = queue_start + 1
    end
end % if queue_green > 0
elseif t >= start_point + green_time(cycle) + amber(1)
    indicator = 'second half yellow or red'
    queue_red = queue_red + 1
end % indicator is 'green' or 'red'
if pointer >= length(arrival)
    break
end
pointer = pointer + 1
t = (arrival(pointer) - 1) / (2*q)
end % while
queue_formed = queue_green + queue_red % queue formed in this cycle
queue_last_cycle = queue_formed
start_point = start_point + green_time(cycle) + amber(1) + red_time_imposed
end % for cycle_needed

[fid,message] = fopen('queue_length.dat', 'a');
fprintf(fid,'instant_entry = %-3d ',instant_entry);
fprintf(fid,' cycle_entry = %-3d ',cycle_entry);
fprintf(fid,' recovery cycles needed = %-6.2f ',cycle_needed);
fprintf(fid,' queue on approach 1 = %-6.2f \n',queue_formed);
status = fclose(fid);
if status == -1 % =0 if successful closed
    disp('error')
    return
end

%-----
function [queue_length] =
queue_length_smooth_2(volume,tt,rn,arrival,turn,t_fixed,cycle_entry,instant_entry);
% generate queue length on cross road under smooth strategy after 50 sec. fixed held
time

```

```

% for transition 1
% Two runs are needed. one for approach 2 and another for approach 4
% volume=traffic volume in veh/hr, tt=total time in second
% rn=random vehicle arrivals, 1=arrival and 0=no arrival
% arrival=indices of rn where an arrival is produced
% turn=turning factor corresponding to each arrival with straight-through (1.0),
% left-turn (1.4), or right-turn (1.2)
% t_fixed=fixed time period held for EV
% cycle_entry=the number of cycle in which EV arrives
% instant_entry=the time instant at which EV arrives, 1-6 for at 0,green(1)/2,green(1),
% green(1)+yellow(1),green(1)+yellow(1)+all_red(1),and
% green(1)+yellow(1)+all_red(1)+green(2)/2, respectively.

green = [40,60] % for side road
yellow = [3,3]
all_red = [1,1]
cycle_length = green(1) + yellow(1) + all_red(1) + green(2) + yellow(2) + all_red(2)
q = volume / 3600
interval = 0.5 / q % vehicle arrival interval
saturation = 2000 / 3600 % in veh/sec
discharge_rate_green = 1 / saturation
% for straight-through vehicles during green in sec/veh
discharge_rate_yellow = 2 * discharge_rate_green
% for straight-through vehicles during green in sec/veh
yellow_equival_green = yellow(1) * discharge_rate_green / discharge_rate_yellow
% transfer yellow interval to equivalent green discharging vehicles
cycle_number = fix(tt / cycle_length) % # of cycles for testing
queue_length = zeros(1,cycle_number)
queue_last_cycle = 0

total_produced = zeros(1,cycle_number)
total_discharged = zeros(1,cycle_number)

pointer = 1
t = (arrival(pointer) - 1) / (2*q)
% arrival time in sec, and the possibly earliest one arrives at 0 sec
queue_start=pointer

for cycle = 1:cycle_number
    if t >= cycle * cycle_length % wait to clear in next cycle
        continue
    end
    queue_green = queue_last_cycle
    queue_red = 0
    discharged_green = 0

```

```

green_time_available = green(1) + yellow_equiv_green

if (cycle==cycle_entry)
    switch instant_entry
        case 1
            entry_point = green(1) + yellow(1) + all_red(1) % in red period
            % time point away from the beginning of green at which EV enters on EV
road
        case 2
            entry_point = green(1) + yellow(1) + all_red(1) + green(2) / 2
        case 3
            entry_point = green(1) + yellow(1) + all_red(1) + green(2)
        case 4
            entry_point = green(1) + yellow(1) + all_red(1) + green(2) + yellow(2)
        case 5
            entry_point = 0 % beginning of green
            green_time_available = 0
        case 6
            entry_point = green(1) / 2 % in green period
            green_time_available = green_time_available - green(1) / 2
    end % switch
end % if

if queue_green > 0 % processing queue from last cycle
    for k = 1:queue_green
        green_time_available = green_time_available - turn(queue_start) *
        discharge_rate_green
        if green_time_available < 0
            break
        end
        discharged_green = k
    end % for
    total_discharged(cycle) = total_discharged(cycle) + discharged_green
    queue_start = queue_start + discharged_green
    queue_green = queue_green - discharged_green
end % if queue_green > 0

while t < cycle * cycle_length % processing arrivals during this cycle
    t_quotient = fix(t / cycle_length)
    t_elapsed = t - cycle_length * t_quotient
    % time elapsed from the beginning of green interval
    if (cycle==cycle_entry) && (t_elapsed >= entry_point)
        % Ev is coming and no need to calculate other cycles
        break
    end
end

```

```

total_produced(cycle) = total_produced(cycle) + 1
if (t_elapsed >= 0) && (t_elapsed < green(1) + yellow_equival_green)
    indicator = 'green or first half yellow'
    if queue_green > 0
        % there already exists queue that cannot be discharged during this green
period
        queue_green = queue_green + 1
    elseif queue_green == 0 % deal with the situation that no queue exists
        if (green(1) - t_elapsed) < green_time_available
            green_time_available = green(1) - t_elapsed
        end
        green_time_available = green_time_available - turn(queue_start) *
discharge_rate_green
        if green_time_available < 0 % queue produced
            queue_green = queue_green + 1
        elseif green_time_available >= 0
            discharged_green = discharged_green + 1
            total_discharged(cycle) = total_discharged(cycle) + 1
            queue_start = queue_start + 1
        end
    end % if queue_green > 0
elseif (t_elapsed >= green(1) + yellow_equival_green)
    indicator = 'second half yellow or red'
    queue_red = queue_red + 1
end % indicator is 'green' or 'red'
if pointer >= length(arrival)
    break
end
pointer = pointer + 1
t = (arrival(pointer) - 1) / (2*q)
end % while
undischarged(cycle) = queue_green % undischarged during this green interval
queue_length(cycle) = queue_green + queue_red
queue_last_cycle = queue_length(cycle)
if cycle==cycle_entry % EV is coming and no need to calculate other cycles
    break
end
end % for

% begin fixed held time for transition 1
queue = queue_last_cycle % queue from normal operation
red_time_imposed = t_fixed

start = (cycle_entry - 1) * cycle_length + entry_point
while (t >= start) && (t <= start + red_time_imposed)

```



```

% produce vehicles during held time,t_fixed
queue = queue + 1
if pointer >= length(arrival)
    break
end
pointer = pointer + 1
t = (arrival(pointer) - 1) / (2*q)
end % while
queue_length(cycle_entry) = queue % produced during t_fixed

% begin smooth transition 2. After this period, signal control goes back to normal
% operation
% get phase timing for transiting from approach 1
fid = fopen('trans_1_timing.dat','r')
if fid == -1 % no such file
    disp(message)
    return
end
for instant = 1:instant_entry
    cycle_needed = fscanf(fid,'%d', 1)
    if cycle_needed == 0
        continue
    end
    red_time = fscanf(fid,'%f',cycle_needed)
    % green for EV road is red for cross road
    green_time = fscanf(fid,'%f',cycle_needed)
    %red for EV road is green for cross road
end
status = fclose(fid)
if status == -1 % =0 if successful closed
    disp('error')
    return
end
if cycle_needed == 0
    queue_form = queue % queue from t_fixed
    [fid,message] = fopen('queue_length.dat', 'a');
    fprintf(fid,'instant_entry = %-3d ',instant_entry);
    fprintf(fid,' cycle_entry = %-3d ',cycle_entry);
    fprintf(fid,' no need to adjust and queue from t_fixed on approach 2/4 = %-6.2g\n',queue_formed);
    status = fclose(fid);
    if status == -1 % =0 if successful closed
        disp('error')
        return
    end
end

```

```

    return
end

queue_last_cycle = queue % queue from t_fixed
t_fin = entry_point + t_fixed % finishing point of EV operation, t_fin.
t_fin_quotient = fix(t_fin / cycle_length)
t_fin_elapsed = t_fin - cycle_length * t_fin_quotient % away from the beginning of
green
t_dif = green(1) + yellow(1) + all_red(1) + green(2) - t_fin_elapsed
% time difference between the finishing point and the end of green(1) interval of
% background cycle
if abs(t_dif) < 0.1
    indicator = 'no need to adjust, get back to normal operation immediately.'
    return
end

start_point = (cycle_entry - 1) * cycle_length + entry_point + t_fixed
for cycle = 1:cycle_needed
    queue_red = queue_last_cycle % queue from transition 1
    red_time_imposed = red_time(cycle) + yellow(2) + all_red(2)
    while (t >= start_point) && (t < start_point + red_time_imposed)
        indicator = 'in red'
        queue_red = queue_red + 1
        if pointer >= length(arrival)
            break
        end
        pointer = pointer + 1
        t = (arrival(pointer) - 1) / (2*q)
    end % while
    start_point = start_point + red_time_imposed
    if green_time(cycle) == 0 && cycle == cycle_needed
        green_time_imposed = green(1)
    else
        green_time_imposed = green_time(cycle)
    end
    green_time_available = green_time_imposed + yellow_equival_green
    amber(1) = yellow(1) + all_red(1)
    queue_green = queue_red
    queue_red = 0
    discharged_green = 0
    total_discharged = 0
    if (queue_green > 0) && (green_time_available > 0)
        % processing queue from red interval
        for k = 1:queue_green
            green_time_available = green_time_available - turn(queue_start) *

```

```

    discharge_rate_green
    if green_time_available < 0
        break
    end
    discharged_green = k
end % for
total_discharged = total_discharged + discharged_green
queue_start = queue_start + discharged_green
queue_green = queue_green - discharged_green
end % if queue_green > 0
while (t >= start_point) && (t < start_point + green_time_imposed + amber(1))
    if t < start_point + green_time_imposed + yellow(1)/2
        indicator = 'green or first half yellow'
        if queue_green > 0
            % there already exists queue that cannot be discharged during this green
            % period
            queue_green = queue_green + 1
        elseif queue_green == 0 % deal with the situation that no queue exists
            if (start_point + green_time_imposed - t) < green_time_available
                green_time_available = start_point + green_time_imposed - t
            end
            green_time_available = green_time_available - turn(queue_start) *
            discharge_rate_green
            if green_time_available < 0 % queue produced
                queue_green = queue_green + 1
            elseif green_time_available >= 0
                discharged_green = discharged_green + 1
                total_discharged = total_discharged + 1
                queue_start = queue_start + 1
            end
        end % if queue_green > 0
    elseif t >= start_point + green_time_imposed + yellow(1)/2
        indicator = 'second half yellow or red'
        queue_red = queue_red + 1
    end % indicator is 'green' or 'red'
    if pointer >= length(arrival)
        break
    end
    pointer = pointer + 1
    t = (arrival(pointer) - 1) / (2*q)
end % while
queue_formed = queue_green + queue_red % queue formed in this cycle
queue_last_cycle = queue_formed
start_point = start_point + green_time_imposed + amber(1)
end % for cycle_needed

```

```

[fid,message] = fopen('queue_length.dat', 'a');
fprintf(fid,'instant_entry = %-3d ',instant_entry);
fprintf(fid,' cycle_entry = %-3d ',cycle_entry);
fprintf(fid,' queue on approach 2/4 = %-6.2f\n',queue_formed);
status = fclose(fid);
if status == -1 % =0 if successful closed
    disp('error')
    return
end

%-----
function [queue_length] =
queue_length_smooth_3(volume,tt,rn,arrival,turn,t_fixed,cycle_entry,instant_entry);
% generate queue length on approach 3 (opposite to EV approach) under smooth
strategy
% after 50 sec. fixed held time for transition 1
% volume=traffic volume in veh/hr, tt=total time in second
% rn=random vehicle arrivals, 1=arrival and 0=no arrival
% arrival=indices of rn where an arrival is produced
% turn=turning factor corresponding to each arrival with straight-through (1.0),
% left-turn (1.4), or right-turn (1.2)
% t_fixed=fixed time period held for EV
% cycle_entry=the number of cycle in which EV arrives
% instant_entry=the time instant at which EV arrives, 1-6 for at 0,green(1)/2,green(1),
% green(1)+yellow(1),green(1)+yellow(1)+all_red(1),and
% green(1)+yellow(1)+all_red(1)+green(2)/2, respectively.

green = [60,40]
yellow = [3,3]
all_red = [1,1]
cycle_length = green(1) + yellow(1) + all_red(1) + green(2) + yellow(2) + all_red(2)
q = volume / 3600
interval = 0.5 / q % vehicle arrival interval
saturation = 2000 / 3600 % in veh/sec
discharge_rate_green = 1 / saturation
% for straight-through vehicles during green in sec/veh
discharge_rate_yellow = 2 * discharge_rate_green
% for straight-through vehicles during yellow in sec/veh
yellow_equival_green = yellow(1) * discharge_rate_green / discharge_rate_yellow
% transfer yellow interval to equivalent green discharging vehicles
cycle_number = fix(tt / cycle_length) % # of cycles for testing
queue_length = zeros(1,cycle_number)
queue_last_cycle = 0

```

```

total_produced = zeros(1,cycle_number)
total_discharged = zeros(1,cycle_number)

pointer = 1
t = (arrival(pointer) - 1) / (2*q)
% arrival time in sec, and the possibly earliest one arrives at 0 sec
queue_start=pointer

for cycle = 1:cycle_number
    if t >= cycle * cycle_length % wait to clear in next cycle
        continue
    end
    queue_green = queue_last_cycle
    queue_red = 0
    discharged_green = 0
    green_time_available = green(1) + yellow_equivall_green

    if (cycle==cycle_entry)
        switch instant_entry
            case 1
                entry_point = 0
                % time point away from the beginning of green at which EV enters
                green_time_available = 0
            case 2
                entry_point = green(1) / 2
                green_time_available = green_time_available - green(1) / 2
            case 3
                entry_point = green(1)
            case 4
                entry_point = green(1) + yellow(1)
            case 5
                entry_point = green(1) + yellow(1) + all_red(1) % in red period
            case 6
                entry_point = green(1) + yellow(1) + all_red(1) + green(2)/2 % in red period
        end % switch
        if (instant_entry==1)
            % EV is coming at the beginning of green and no need to calculate other cycles
            break
        end
    end % if

    if queue_green > 0 % processing queue from last cycle
        for k = 1:queue_green
            green_time_available = green_time_available - turn(queue_start) *
            discharge_rate_green
        end
    end
end

```

```

        if green_time_available < 0
            break
        end
        discharged_green = k
    end % for
    total_discharged(cycle) = total_discharged(cycle) + discharged_green
    queue_start = queue_start + discharged_green
    queue_green = queue_green - discharged_green
end % if queue_green > 0

while t < cycle * cycle_length % processing arrivals during this cycle
    t_quotient = fix(t / cycle_length)
    t_elapsed = t - cycle_length * t_quotient
    % time elapsed from the beginning of green interval
    if (cycle == cycle_entry) && (t_elapsed >= entry_point)
        % Ev is coming and no need to calculate other cycles
        break
    end
    total_produced(cycle) = total_produced(cycle) + 1
    if (t_elapsed >= 0) && (t_elapsed < green(1) + yellow_equival_green)
        indicator = 'green or first half yellow'
        if queue_green > 0
            % there already exists queue that cannot be discharged during this green
period
            queue_green = queue_green + 1
        elseif queue_green == 0 % deal with the situation that no queue exists
            if (green(1) - t_elapsed) < green_time_available
                green_time_available = green(1) - t_elapsed
            end
            green_time_available = green_time_available - turn(queue_start) *
discharge_rate_green
            if green_time_available < 0 % queue produced
                queue_green = queue_green + 1
            elseif green_time_available >= 0
                discharged_green = discharged_green + 1
                total_discharged(cycle) = total_discharged(cycle) + 1
                queue_start = queue_start + 1
            end
        end % if queue_green > 0
    elseif (t_elapsed >= green(1) + yellow_equival_green)
        indicator = 'second half yellow or red'
        queue_red = queue_red + 1
    end % indicator is 'green' or 'red'
    if pointer >= length(arrival)
        break
    end
end

```

```

    end
    pointer = pointer + 1
    t = (arrival(pointer) - 1) / (2*q)
end % while
undischarged(cycle) = queue_green % undischarged during this green interval
queue_length(cycle) = queue_green + queue_red
queue_last_cycle = queue_length(cycle)
if cycle==cycle_entry % EV is coming and no need to calculate other cycles
    break
end
end % for

% begin fixed held time period
queue = queue_last_cycle % queue from normal operation
discharged_green = 0
green_time_available = t_fixed

if queue > 0 % processing queue from normal operation
    total_green_time_needed = 0
    for k = 1:queue
        total_green_time_needed = total_green_time_needed + turn(queue_start + k - 1) *
            discharge_rate_green
    end % for
    for k = 1:queue
        green_time_available = green_time_available - turn(queue_start + k - 1) *
            discharge_rate_green
        if green_time_available < 0
            break
        end
        discharged_green = k
    end % for
    discharged_for_ev = discharged_green
    queue_start = queue_start + discharged_green
    queue = queue - discharged_green
end % if queue > 0

start = (cycle_entry - 1) * cycle_length + entry_point
while (t >= start) && (t <= start + t_fixed) % produce vehicles during held time,t_fixed
    if queue > 0
        % there already exists queue that cannot be discharged during this green period
        queue = queue + 1
    elseif queue == 0 % deal with the situation that no queue exists
        if (start + t_fixed - t) < green_time_available
            green_time_available = start + t_fixed - t
        end
    end
end

```

```

green_time_available = green_time_available - turn(queue_start) *
discharge_rate_green
if green_time_available < 0 % queue produced
    queue = queue + 1
elseif green_time_available >= 0
    discharged_green = discharged_green + 1
    queue_start = queue_start + 1
end
end % if queue > 0
if pointer >= length(arrival)
    break
end
pointer = pointer + 1
t = (arrival(pointer) - 1) / (2*q)
end % while
queue_formed = queue
queue_length(cycle_entry) = queue % produced at the end of t_fixed period

% begin smooth transition 2. After this period, signal control goes back to normal
% operation
% get phase timing for transiting from approach 1
fid = fopen('trans_1.dat','r')
if fid == -1 % no such file
    disp(message)
    return
end
for instant = 1:instant_entry
    cycle_needed = fscanf(fid,'%d', 1)
    if cycle_needed == 0
        continue
    end
    green_adj = fscanf(fid,'%f',cycle_needed)
    red_adj = fscanf(fid,'%f',cycle_needed)
end
status = fclose(fid)
if status == -1 % =0 if successful closed
    disp('error')
    return
end
if cycle_needed == 0
    queue_formed = queue % queue from t_fixed
    [fid,message] = fopen('queue_length.dat', 'a');
    fprintf(fid,'instant_entry = %-3d ',instant_entry);
    fprintf(fid,' cycle_entry = %-3d ',cycle_entry);
    fprintf(fid,' no need to adjust and queue from t_fixed on approach 3 = %-6.2g

```



```

        \n',queue_formed);
        status = fclose(fid);
        if status == -1 % =0 if successful closed
            disp('error')
            return
        end
        return
    end

    queue_last_cycle = queue % queue from t_fixed
    t_fin = entry_point + t_fixed % finishing point of EV operation, t_fin.
    t_fin_quotient = fix(t_fin / cycle_length)
    t_fin_elapsed = t_fin - cycle_length * t_fin_quotient % away from the beginning of
    green
    t_dif = green(1) - t_fin_elapsed
    % time difference between the finishing point and the end of green interval of
    % background cycle
    if abs(t_dif) < 0.1
        indicator = 'no need to adjust, get back to normal operation immediately.'
        return
    end

    for k = 1: cycle_needed
        if green_adj(k) == 0
            green_time(k) = 0
        else
            green_time(k) = green(1) + green_adj(k)
        end
        if red_adj(k) == 0
            red_time(k) = 0
        else
            red_time(k) = green(2) + red_adj(k)
        end
    end

    start_point = (cycle_entry - 1) * cycle_length + entry_point + t_fixed
    for cycle = 1:cycle_needed
        queue_green = queue_last_cycle % queue from transition 1
        if green_time(cycle) == 0 && (red_adj(cycle) < 0 || red_adj(cycle) == 0)
            % falls in red interval
            % red_adj(cycle) < 0: in first half red; red_adj(cycle) == 0: in second half red
            % red_adj(cycle) > 0: in second half green
            amber(1) = 0
            green_time_available = 0
        else % falls in green interval
            amber(1) = yellow(1) + all_red(1)
        end
    end

```

```

        green_time_available = green_time(cycle) + yellow_equival_green
    end
    if (red_time(cycle) == 0) && (cycle == cycle_needed)
        % in the last cycle and no adjustment to red
        red_time_imposed = green(2) + yellow(2) + all_red(2)
    else
        red_time_imposed = red_time(cycle) + yellow(2) + all_red(2)
    end
    queue_red = 0
    discharged_green = 0
    total_discharged = 0
    if (queue_green > 0) && (green_time_available > 0)
        % processing queue from transition 1
        for k = 1:queue_green
            green_time_available = green_time_available - turn(queue_start) *
            discharge_rate_green
            if green_time_available < 0
                break
            end
            discharged_green = k
        end % for
        total_discharged = total_discharged + discharged_green
        queue_start = queue_start + discharged_green
        queue_green = queue_green - discharged_green
    end % if queue_green > 0
    while (t >= start_point) &&
    (t < start_point + green_time(cycle) + amber(1) + red_time_imposed)
        if t < start_point + green_time(cycle) + amber(1)
            indicator = 'green or first half yellow'
            if queue_green > 0 % there already exists queue that cannot be
discharged
                % during this green period
                queue_green = queue_green + 1
            elseif queue_green == 0 % deal with the situation that no queue exists
                if (start_point + green_time(cycle) - t) < green_time_available
                    green_time_available = start_point + green_time(cycle) - t
                end
                green_time_available = green_time_available - turn(queue_start) *
                discharge_rate_green
                if green_time_available < 0 % queue produced
                    queue_green = queue_green + 1
                elseif green_time_available >= 0
                    discharged_green = discharged_green + 1
                    total_discharged = total_discharged + 1
                    queue_start = queue_start + 1
                end
            end
        end
    end

```

```

        end
        end % if queue_green > 0
        elseif t >= start_point + green_time(cycle) + amber(1)
            indicator = 'second half yellow or red'
            queue_red = queue_red + 1
        end % indicator is 'green' or 'red'
        if pointer >= length(arrival)
            break
        end
        pointer = pointer + 1
        t = (arrival(pointer) - 1) / (2*q)
    end % while
    queue_formed = queue_green + queue_red % queue formed in this cycle
    queue_last_cycle = queue_formed
    start_point = start_point + green_time(cycle) + amber(1) + red_time_imposed
end % for cycle

[fid,message] = fopen('queue_length.dat', 'a');
fprintf(fid,'instant_entry = %-3d ',instant_entry);
fprintf(fid,' cycle_entry = %-3d ',cycle_entry);
fprintf(fid,' queue on approach 3 = %-6.2f\n',queue_formed);
status = fclose(fid);
if status == -1 % =0 if successful closed
    disp('error')
    return
end

```

Appendix F Computer Program for Real-time Control

```
% open a file to record simulation results
[fid,message] = fopen('queue_length.dat', 'w')
fprintf(fid,'queue_length_real_time_approach# \n')
status = fclose(fid)
if status == -1 % =0 if successful closed
    disp('error')
    return
end
% then run one of the following to produce queue on corresponding approach
% queue_length_real_time_1 for approach 1
% queue_length_real_time_2 for approach 2 or 4
% queue_length_real_time_3 for approach 3
% input simulation period tt = 2160 seconds for 20 cycles
tt = 2160
% input traffic volume as required and then generate vehicle arrivals
volume = 1000
[rn,arrival,turn] = vehicle_arrival(volume)
for instant_entry=1:6
    for cycle_entry=5:10
        queue_length_real_time_approach#(volume,tt,rn,arrival,turn,
            cycle_entry,instant_entry);
    end
end

end

%-----
function [queue_length,successful] =
queue_length_real_time_1(volume,tt,rn,arrival,turn,cycle_entry,instant_entry);
% generate queue length on EV approach under real time strategy for transition 1
% For comparison with fixed time control, STI and startup delay will not be included in
% the calculation, i.e., STI=0 and t_startup=0
% Using detector distance L=960m
% EV operating speed SEV = speed_ev = 15 (m/s or 54 Km/h),
% General vehicle mean Smean = speed_mean = 12 (m/s or 43.2 Km/h)
% discharge speed Sdis = speed_dis = 4.44 (m/s or 16 Km/h)=2000*8/3600
% Average spacing between successive vehicles spacing = 8 metre
% Minimum greens for main road gmin(1) = 7 sec, and side road gmin(2) = 10 sec
% Maximum greens for main road gmax(1) = 100 sec, and side road gmax(2) = 100 sec

% successful='Yes!' if EV can cross the intersection, otherwise successful='No!'
% volume=traffic volume in veh/hr, tt=total time in second
```

```
% rn=random vehicle arrivals, 1=arrival and 0=no arrival
% arrival=indices of rn where an arrival is produced
% turn=turning factor corresponding to each arrival with straight-through (1.0),
% left-turn (1.4), or right-turn (1.2)
% cycle_entry=the number of cycle in which EV arrives
% instant_entry=the time instant at which EV arrives, 1-6 for at 0,green(1)/2,green(1),
% green(1)+yellow(1), green(1)+yellow(1)+all_red(1),and
% green(1)+yellow(1)+all_red(1)+green(2)/2, respectively.
```

```
green = [60,40]
yellow = [3,3]
all_red = [1,1]
cycle_length = green(1) + yellow(1) + all_red(1) + green(2) + yellow(2) + all_red(2)
q = volume / 3600
interval = 0.5 / q % vehicle arrival interval
saturation = 2000 / 3600 % in veh/sec
discharge_rate_green = 1 / saturation
% for straight-through vehicles during green in sec/veh
discharge_rate_yellow = 2 * discharge_rate_green
% for straight-through vehicles during yellow in sec/veh
yellow_equival_green = yellow(1) * discharge_rate_green / discharge_rate_yellow
% transfer yellow interval to equivalent green
cycle_number = fix(tt / cycle_length) % # of cycles for testing
queue_length = zeros(1,cycle_number)
queue_last_cycle = 0
```

```
ll=960 % detector distance
speed_ev = 15 % EV operating speed SEV(m/s or 54 Km/h)
t_ev = ll / speed_ev % take EV to traverse l
sti = 0
t_startup = 0
t_ev = t_ev - 5
% minus STI=3 and startup delay=2 for comparison with fixed held time control
speed_mean = 12 % General vehicle mean Smean (m/s or 43.2 Km/h)
spacing = 8 % Average spacing between successive vehicles spacing
speed_dis = saturation * spacing
% discharge speed Sdis=4.44 (m/s or 16 Km/h)=2000*8/3600
gmin = [7,10]
gmax = [100,100]
total_produced = zeros(1,cycle_number);
total_discharged = zeros(1,cycle_number);
```

```
pointer = 1
t = (arrival(pointer) - 1) / (2*q)
% arrival time in sec, and the possibly earliest one arrives at 0 sec
```

```

queue_start=pointer

for cycle = 1:cycle_number
    if t >= cycle * cycle_length % wait to clear in next cycle
        continue
    end
    queue_green = queue_last_cycle
    % queue_yellow = 0
    queue_red = 0
    discharged_green = 0
    % discharged_yellow = 0
    green_time_available = green(1) + yellow_equival_green + 0.01
    % plus 0.01 for counting calculation error

    if (cycle==cycle_entry)
        switch instant_entry
            case 1
                entry_point = 0
                % time point away from the beginning of green at which EV enters
                green_time_available = 0
            case 2
                entry_point = green(1) / 2
                green_time_available = green_time_available - green(1) / 2
            case 3
                entry_point = green(1)
            case 4
                entry_point = green(1) + yellow(1)
            case 5
                entry_point = green(1) + yellow(1) + all_red(1) % in red period
            case 6
                entry_point = green(1) + yellow(1) + all_red(1) + green(2)/2 % in red period
        end % switch
        if (instant_entry==1)
            % EV in coming at the beginning of green and no need to calculate other cycles
            break
        end
    end % if

    if queue_green > 0 % processing queue from last cycle
        for k = 1:queue_green
            green_time_available = green_time_available - turn(queue_start) *
            discharge_rate_green
            if green_time_available < 0
                break
            end
        end
    end
end

```

```

        discharged_green = k
    end % for
    total_discharged(cycle) = total_discharged(cycle) + discharged_green
    queue_start = queue_start + discharged_green
    queue_green = queue_green - discharged_green
end % if queue_green > 0

while t < cycle * cycle_length % processing arrivals during this cycle
    t_quotient = fix(t / cycle_length)
    t_elapsed = t - cycle_length * t_quotient
    % time elapsed from the beginning of green interval
    if (cycle == cycle_entry) && (t_elapsed >= entry_point)
        % EV is coming and no need to calculate other cycles
        break
    end
    total_produced(cycle) = total_produced(cycle) + 1
    if (t_elapsed >= 0) && (t_elapsed < green(1) + yellow_equival_green)
        indicator = 'green or first half yellow'
        if queue_green > 0
            % there already exists queue that cannot be discharged during this green period
            queue_green = queue_green + 1
        elseif queue_green == 0 % deal with the situation that no queue exists
            if (green(1) - t_elapsed) < green_time_available
                green_time_available = green(1) - t_elapsed
            end
            green_time_available = green_time_available - turn(queue_start) *
            discharge_rate_green
            if green_time_available < 0 % queue produced
                queue_green = queue_green + 1
            elseif green_time_available >= 0
                discharged_green = discharged_green + 1
                total_discharged(cycle) = total_discharged(cycle) + 1
                queue_start = queue_start + 1
            end
        end
        end % if queue_green > 0
    elseif (t_elapsed >= green(1) + yellow_equival_green)
        indicator = 'second half yellow or red'
        queue_red = queue_red + 1
    end % indicator is 'green' or 'red'
    if pointer >= length(arrival)
        break
    end
    pointer = pointer + 1
    t = (arrival(pointer) - 1) / (2*q)
end % while

```

```

undischarged(cycle) = queue_green % undischarged during this green interval
queue_length(cycle) = queue_green + queue_red
queue_last_cycle = queue_length(cycle)
if cycle==cycle_entry % EV is coming and no need to calculate other cycles
    break
end
end % for

queue = queue_last_cycle % queue from normal operation
discharged_green = 0
queue_equiv_length = 0
if queue == 0
    t_clear = 0
else % queue > 0
    for k = 1:queue
        queue_equiv_length = queue_equiv_length + turn(queue_start + k - 1)
    end

    t_clear_1 = queue_equiv_length * spacing / speed_mean
    % take general traffic to traverse queue length
    t_clear_2 = queue_equiv_length / saturation
    if t_clear_1 > t_clear_2
        t_clear = t_clear_1 + t_startup
    else
        t_clear = t_clear_2 + t_startup
    end
    if t_clear > gmax(1)
        t_clear = gmax(1)
        successful = 'no, because detector distance not long enough.'
        return
    elseif t_clear < gmin(1)
        t_clear = gmin(1)
    end
end
end
switch instant_entry % calculate amber time prior to giving green to side road
case 1
    t_amber_1 = 0
case {2,3}
    t_amber_1 = yellow(1) + all_red(1)
case 4
    t_amber_1 = all_red(1) % begining of red
case 5
    t_amber_1 = 0
case 6
    t_amber_1 = 0 % in red period

```



```

end % switch
t_amber_2 = yellow(2) + all_red(2) + sti
% calculate amber time after giving green to side road
if t_clear < 0.1 % == 0 for calculation error
    t_green_side = t_ev - t_amber_1 - t_amber_2
else % t_clear > 0
    t_green_side = t_ev - t_amber_1 - t_amber_2 - t_clear
end
if t_green_side > gmax(2)
    t_green_side = gmax(2)
elseif t_green_side < gmin(2)
    t_green_side = 0
end
switch instant_entry % calculate yellow_equival_green to EV route
    case {2,3}
        yellow_equival_green = ( t_amber_1 - all_red(1) ) * discharge_rate_green /
            discharge_rate_yellow
    case {1,4,5,6}
        yellow_equival_green = 0
end % switch
if t_green_side == 0 % all t_ev is green for main road and red for side road.
    t_green_main = t_ev - t_amber_1
else % t_green_side > 0
    t_green_main = t_ev - t_amber_1 - t_amber_2 - t_green_side
end % if
green_time_available = t_green_main + yellow_equival_green + 0.01

if queue > 0 % processing queue from normal operation
    total_green_time_needed = 0
    for k = 1:queue
        total_green_time_needed = total_green_time_needed + turn(queue_start + k - 1) *
            discharge_rate_green
    end % for
    for k = 1:queue
        green_time_available = green_time_available - turn(queue_start + k - 1) *
            discharge_rate_green
        if green_time_available < 0
            break
        end
        discharged_green = k
    end % for
    discharged_for_ev = discharged_green
    queue_start = queue_start + discharged_green
    queue = queue - discharged_green
end % if queue > 0

```

```

if (queue > 0)
    successful = 'No!' % more time is needed
    flag = -1
    time_needed = total_green_time_needed - t_green_main - yellow_equival_green
elseif (queue == 0)
    successful = 'Yes!'
    flag = 1
    time_spare = green_time_available - yellow_equival_green
    % exclude yellow_equival_green
    time_spare = round(time_spare * 10) / 10 % exclude yellow_equival_green
    start = (cycle_entry - 1) * cycle_length + entry_point
    while (t >= start) && (t <= start + t_ev)
        % produce affected vehicles during EV operation: t_ev
        queue = queue + 1
        if pointer >= length(arrival)
            break
        end
        pointer = pointer + 1
        t = (arrival(pointer) - 1) / (2*q)
    end % while
    queue_formed = queue % produced during t_ev
end %if

[fid,message] = fopen('queue_length.dat', 'a');
fprintf(fid,'instant_entry = %-3d ',instant_entry);
fprintf(fid,' cycle_entry = %-3d \n',cycle_entry);
if flag == -1
    fprintf(fid,' green time for side road = %-6.2f',t_green_side);
    fprintf(fid,' time_needed on approach 1 = %-6.2f',time_needed);
    fprintf(fid,' queue undischarged = %-6.2g \n',queue);
elseif flag == 1
    fprintf(fid,' green time for main road = %-6.2f',t_green_main);
    fprintf(fid,' green time for side road = %-6.2f',t_green_side);
    fprintf(fid,' spare time = %-6.2f',time_spare);
    fprintf(fid,' queue formed = %-6.2g \n',queue);
end
status = fclose(fid);
if status == -1 % =0 if successful closed
    disp('error')
    return
end

%-----

```

```

function [queue_length,successful] = queue_length_real_time_2(volume,tt,rn,arrival,
turn,t_green_side,cycle_entry,instant_entry);
% generate queue length on cross road (approach 2,4) under real time strategy for
% transition 1
% two runs are needed for approaches 2 and 4 respectively
% For comparison with fixed time control, STI and startup delay will not be included
in
% the calculation, i.e., STI=0 and t_startup=0
% Using detector distance L=960m
% EV operating speed SEV = speed_ev = 15 (m/s or 54 Km/h),
% General vehicle mean Smean = speed_mean = 12 (m/s or 43.2 Km/h)
% discharge speed Sdis = speed_dis = 4.44 (m/s or 16 Km/h)=2000*8/3600
% Average spacing between successive vehicles spacing = 8 metre
% Minimum greens for main road gmin(1) = 7 sec, and side road gmin(2) = 10 sec
% Maximum greens for main road gmax(1) = 100 sec, and side road gmax(2) = 100 sec

% successful='Yes!' if EV can cross the intersection, otherwise successful='No!'
% volume=traffic volume in veh/hr, tt=total time in second
% rn=random vehicle arrivals, 1=arrival and 0=no arrival
% arrival=indices of rn where an arrival is produced
% turn=turning factor corresponding to each arrival with straight-through (1.0),
% left-turn (1.4), or right-turn (1.2)
% t_green_side=green time available for side road, then give green to main road to clear
% the way for EV
% cycle_entry=the number of cycle in which EV arrives
% instant_entry=the time instant at which EV arrives, 1-6 for at 0,green(1)/2,green(1),
% green(1)+yellow(1), green(1)+yellow(1)+all_red(1),and
% green(1)+yellow(1)+all_red(1)+green(2)/2, respectively.

green = [40,60] % for side road
yellow = [3,3]
all_red = [1,1]
cycle_length = green(1) + yellow(1) + all_red(1) + green(2) + yellow(2) + all_red(2)
q = volume / 3600
interval = 0.5 / q % vehicle arrival interval
saturation = 2000 / 3600 % in veh/sec
discharge_rate_green = 1 / saturation
% for straight-through vehicles during green in sec/veh
discharge_rate_yellow = 2 * discharge_rate_green
% for straight-through vehicles during green in sec/veh
yellow_equival_green = yellow(1) * discharge_rate_green / discharge_rate_yellow
% transfer yellow interval to equivalent green discharging vehicles
cycle_number = fix(tt / cycle_length) % # of cycles for testing
queue_length = zeros(1,cycle_number)
queue_last_cycle = 0

```

```

ll=960 % detector distance
speed_ev = 15 % EV operating speed SEV(m/s or 54 Km/h)
t_ev = ll / speed_ev % take EV to traverse l
sti = 0
t_startup = 0
t_ev = t_ev - 5
% minus STI=3 and startup delay=2 for comparison with fixed held time control
speed_mean = 12 % General vehicle mean Smean (m/s or 43.2 Km/h)
spacing = 8 % Average spacing between successive vehicles spacing
speed_dis = saturation * spacing
% discharge speed Sdis=4.44 (m/s or 16 Km/h)=2000*8/3600
gmin = [7,10] % 7 for main road
gmax = [100,100]
total_produced = zeros(1,cycle_number);
total_discharged = zeros(1,cycle_number);

pointer = 1
t = (arrival(pointer) - 1) / (2*q)
% arrival time in sec, and the possibly earliest one arrives at 0 sec
queue_start=pointer

for cycle = 1:cycle_number
    if t >= cycle * cycle_length % wait to clear in next cycle
        continue
    end
    queue_green = queue_last_cycle
    queue_red = 0
    discharged_green = 0
    green_time_available = green(1) + yellow_equival_green + 0.01
    % for counting calculation error

    if (cycle==cycle_entry)
        switch instant_entry
            case 1
                entry_point = green(1) + yellow(1) + all_red(1) % in red period
                % time point away from the beginning of green at which EV enters on
                % main road
            case 2
                entry_point = green(1) + yellow(1) + all_red(1) + green(2) / 2
            case 3
                entry_point = green(1) + yellow(1) + all_red(1) + green(2)
            case 4
                entry_point = green(1) + yellow(1) + all_red(1) + green(2) + yellow(2)
            case 5

```

```

        entry_point = 0
        green_time_available = 0
    case 6
        entry_point = green(1) / 2 % only this case in green period
        green_time_available = green_time_available - green(1) / 2
    end % switch
end % if

if queue_green > 0 % processing queue from last cycle
    for k = 1:queue_green
        green_time_available = green_time_available - turn(queue_start) *
            discharge_rate_green
        if green_time_available < 0
            break
        end
        discharged_green = k
    end % for
    total_discharged(cycle) = total_discharged(cycle) + discharged_green
    queue_start = queue_start + discharged_green
    queue_green = queue_green - discharged_green
end % if queue_green > 0

while t < cycle * cycle_length % processing arrivals during this cycle
    t_quotient = fix(t / cycle_length)
    t_elapsed = t - cycle_length * t_quotient
    % time elapsed from the beginning of green interval
    if (cycle == cycle_entry) && (t_elapsed >= entry_point)
        % EV is coming and no need to calculate other cycles
        break
    end
    total_produced(cycle) = total_produced(cycle) + 1
    if (t_elapsed >= 0) && (t_elapsed < green(1) + yellow_equival_green)
        indicator = 'green or first half yellow'
        if queue_green > 0
            % there already exists queue that cannot be discharged during this green period
            queue_green = queue_green + 1
        elseif queue_green == 0 % deal with the situation that no queue exists
            if (green(1) - t_elapsed) < green_time_available
                green_time_available = green(1) - t_elapsed
            end
            green_time_available = green_time_available - turn(queue_start) *
                discharge_rate_green
            if green_time_available < 0 % queue produced
                queue_green = queue_green + 1
            elseif green_time_available >= 0

```

```

        discharged_green = discharged_green + 1
        total_discharged(cycle) = total_discharged(cycle) + 1
        queue_start = queue_start + 1
    end
    end % if queue_green > 0
elseif (t_elapsed >= green(1) + yellow_equival_green)
    indicator = 'second half yellow or red'
    queue_red = queue_red + 1
end % indicator is 'green' or 'red'
if pointer >= length(arrival)
    break
end
pointer = pointer + 1
t = (arrival(pointer) - 1) / (2*q)
end % while
undischarged(cycle) = queue_green % undischarged during this green interval
queue_length(cycle) = queue_green + queue_red
queue_last_cycle = queue_length(cycle)
if cycle==cycle_entry % EV is coming and no need to calculate other cycles
    break
end
end % for

queue = queue_last_cycle % queue from normal operation
switch instant_entry % calculate amber time prior to giving green to side road
case 1
    t_amber_1 = 0
case {2,3}
    t_amber_1 = yellow(2) + all_red(2)
case 4
    t_amber_1 = all_red(2) % beginning of red
case 5
    t_amber_1 = 0
case 6
    t_amber_1 = 0 % in red period
end % switch
t_amber_2 = yellow(1) + all_red(1) + sti
% calculate amber time after giving green to side road
if t_green_side == 0
    % all t_ev is green for main road and red for side road. t_green_side:input parameter
    t_red_imposed_1 = 0 % red before getting side green
    t_red_imposed_2 = t_ev % red after getting side green
    yellow_equival_green = 0 % no yellow for side road
else % t_green_side > 0
    t_red_imposed_1 = t_amber_1 % red before getting side green

```

```

    t_red_imposed_2 = t_ev - t_amber_1 - t_green_side % red after getting side green
end % if
green_time_available = t_green_side + yellow_equival_green + 0.01

if t_red_imposed_1 > 0 % produce queue prior to side green
    start = (cycle_entry - 1) * cycle_length + entry_point
    while (t >= start) && (t < start + t_red_imposed_1)
        queue = queue + 1
        if pointer >= length(arrival)
            break
        end
        pointer = pointer + 1
        t = (arrival(pointer) - 1) / (2*q)
    end % while
end

% processing queue from t_red_imposed_1
discharged_green = 0
if queue > 0
    for k = 1:queue
        green_time_available = green_time_available - turn(queue_start + k - 1) *
            discharge_rate_green
        if green_time_available < 0
            break
        end
        discharged_green = k
    end % for
    discharged_for_ev = discharged_green
    queue_start = queue_start + discharged_green
    queue = queue - discharged_green
end % if queue > 0

% process queue and arrivals after t_red_imposed_1
discharged_green = 0
queue_red = 0
start = (cycle_entry - 1) * cycle_length + entry_point + t_red_imposed_1
while (t >= start) && (t < start + t_green_side + t_red_imposed_2)
    if (t < start + t_green_side + yellow_equival_green)
        indicator = 'green or first half yellow'
        if queue > 0
            % there already exists queue that cannot be discharged during t_green_side
            queue = queue + 1
        elseif queue == 0 % deal with the situation that no queue exists
            if (start + t_green_side - t) < green_time_available
                green_time_available = start + t_green_side - t
            end
        end
    end
end

```

```

end
green_time_available = green_time_available - turn(queue_start) *
discharge_rate_green
if green_time_available < 0 % queue produced
    queue = queue + 1
elseif green_time_available >= 0
    discharged_green = discharged_green + 1
    queue_start = queue_start + 1
end
end % if queue_green > 0
elseif (t >= start + t_green_side + yellow_equival_green)
    indicator = 'second half yellow or red'
    queue_red = queue_red + 1
end % indicator is 'green' or 'red'
if pointer >= length(arrival)
    break
end
pointer = pointer + 1
t = (arrival(pointer) - 1) / (2*q)
end % while
queue_formed = queue + queue_red

[fid,message] = fopen('queue_length.dat', 'a');
fprintf(fid,'instant_entry = %-3d ',instant_entry);
fprintf(fid,' cycle_entry = %-3d ',cycle_entry);
fprintf(fid,' queue on approach 2/4 = %-6.2g \n',queue_formed);
status = fclose(fid);
if status == -1 % =0 if successful closed
    disp('error')
    return
end

%-----
function [queue_length,successful]=queue_length_real_time_3(volume,tt,rn,arrival,
turn,t_green_side,cycle_entry,instant_entry);
% generate queue length on opposite approach (approach 3) under real time strategy for
% transition 1
% For comparison with fixed time control, STI and startup delay will not be included
in
% the calculation, i.e., STI=0 and t_startup=0
% Using detector distance L=960m
% EV operating speed SEV = speed_ev = 15 (m/s or 54 Km/h),
% General vehicle mean Smean = speed_mean = 12 (m/s or 43.2 Km/h)
% discharge speed Sdis = speed_dis = 4.44 (m/s or 16 Km/h)=2000*8/3600
% Average spacing between successive vehicles spacing = 8 metre

```



```

% Minimum greens for main road gmin(1) = 7 sec, and side road gmin(2) = 10 sec
% Maximum greens for main road gmax(1) = 100 sec, and side road gmax(2) = 100 sec

% successful='Yes!' if EV can cross the intersection, otherwise successful='No!'
% volume=traffic volume in veh/hr, tt=totoal time in second
% rn=random vehicle arrivals, 1=arrival and 0=no arrival
% arrival=indices of rn where an arrival is produced
% turn=turning factor corresponding to each arrival with straight-through (1.0),
% left-turn (1.4), or right-turn (1.2)
% t_green_side=green time available for side road, then give green to main road to clear
% the way for EV
% cycle_entry=the number of cycle in which EV arrives
% instant_entry=the time instant at which EV arrives, 1-6 for at 0,green(1)/2,green(1),
% green(1)+yellow(1), green(1)+yellow(1)+all_red(1),and
% green(1)+yellow(1)+all_red(1)+green(2)/2, respectively.

green = [60,40]
yellow = [3,3]
all_red = [1,1]
cycle_length = green(1) + yellow(1) + all_red(1) + green(2) + yellow(2) + all_red(2)
q = volume / 3600
interval = 0.5 / q % vehicle arrival interval
saturation = 2000 / 3600 % in veh/sec
discharge_rate_green = 1 / saturation
% for straight-through vehicles during green in sec/veh
discharge_rate_yellow = 2 * discharge_rate_green
% for straight-through vehicles during green in sec/veh
yellow_equival_green = yellow(1) * discharge_rate_green / discharge_rate_yellow
% transfer yellow interval to equivalent green discharging vehicles
cycle_number = fix(tt / cycle_length) % # of cycles for testing
queue_length = zeros(1,cycle_number)
queue_last_cycle = 0

ll=960 % detector distance
speed_ev = 15 % EV operating speed SEV(m/s or 54 Km/h)
t_ev = ll / speed_ev % take EV to traverse l
sti = 0
t_startup = 0
t_ev = t_ev - 5
% minus STI=3 and startup delay=2 for comparison with fixed held time control
speed_mean = 12 % General vehicle mean Smean (m/s or 43.2 Km/h)
spacing = 8 % Average spacing between successive vehicles spacing
speed_dis = saturation * spacing
% discharge speed Sdis=4.44 (m/s or 16 Km/h)=2000*8/3600
gmin = [7,10]

```

```

gmax = [100,100]
total_produced = zeros(1,cycle_number);
total_discharged = zeros(1,cycle_number);

pointer = 1
t = (arrival(pointer) - 1) / (2*q)
% arrival time in sec, and the possibly earliest one arrives at 0 sec
queue_start=pointer

for cycle = 1:cycle_number
    if t >= cycle * cycle_length % wait to clear in next cycle
        continue
    end
    queue_green = queue_last_cycle
    queue_red = 0
    discharged_green = 0
    green_time_available = green(1) + yellow_equival_green + 0.01
    % for counting calculation error

    if (cycle==cycle_entry)
        switch instant_entry
            case 1
                entry_point = 0
                % time point away from the beginning of green at which EV enters
                green_time_available = 0
            case 2
                entry_point = green(1) / 2
                green_time_available = green_time_available - green(1) / 2
            case 3
                entry_point = green(1)
            case 4
                entry_point = green(1) + yellow(1)
            case 5
                entry_point = green(1) + yellow(1) + all_red(1)
            case 6
                entry_point = green(1) + yellow(1) + all_red(1) + green(2)/2
                % only this case in red period
        end % switch
        if (instant_entry==1)
            % EV in coming at the beginning of green and no need to calculate other cycles
            break
        end
    end % if

    if queue_green > 0 % processing queue from last cycle

```

```

for k = 1:queue_green
    green_time_available = green_time_available - turn(queue_start) *
    discharge_rate_green
    if green_time_available < 0
        break
    end
    discharged_green = k
end % for
total_discharged(cycle) = total_discharged(cycle) + discharged_green
queue_start = queue_start + discharged_green
queue_green = queue_green - discharged_green
end % if queue_green > 0

while t < cycle * cycle_length % processing arrivals during this cycle
    t_quotient = fix(t / cycle_length)
    t_elapsed = t - cycle_length * t_quotient
    % time elapsed from the beginning of green interval
    if (cycle == cycle_entry) && (t_elapsed >= entry_point)
        % EV is coming and no need to calculate other cycles
        break
    end
    total_produced(cycle) = total_produced(cycle) + 1
    if (t_elapsed >= 0) && (t_elapsed < green(1) + yellow_equival_green)
        indicator = 'green or first half yellow'
        if queue_green > 0
            % there already exists queue that cannot be discharged during this green period
            queue_green = queue_green + 1
            % continue
        elseif queue_green == 0 % deal with the situation that no queue exists
            if (green(1) - t_elapsed) < green_time_available
                green_time_available = green(1) - t_elapsed
            end
            green_time_available = green_time_available - turn(queue_start) *
            discharge_rate_green
            if green_time_available < 0 % queue produced
                queue_green = queue_green + 1
            elseif green_time_available >= 0
                discharged_green = discharged_green + 1
                total_discharged(cycle) = total_discharged(cycle) + 1
                queue_start = queue_start + 1
            end
        end % if queue_green > 0
    elseif (t_elapsed >= green(1) + yellow_equival_green)
        indicator = 'second half yellow or red'
        queue_red = queue_red + 1
    end
end

```

```

    end % indicator is 'green' or 'red'
    if pointer >= length(arrival)
        break
    end
    pointer = pointer + 1
    t = (arrival(pointer) - 1) / (2*q)
end % while
undischarged(cycle) = queue_green % undischarged during this green interval
queue_length(cycle) = queue_green + queue_red
queue_last_cycle = queue_length(cycle)
if cycle==cycle_entry % EV is coming and no need to calculate other cycles
    break
end
end % for

queue = queue_last_cycle % queue from normal operation
discharged_green = 0
switch instant_entry % calculate amber time prior to giving green to side road
    case 1
        t_amber_1 = 0
    case {2,3}
        t_amber_1 = yellow(1) + all_red(1)
    case 4
        t_amber_1 = all_red(1) % beginning of red
    case 5
        t_amber_1 = 0
    case 6
        t_amber_1 = 0 % in red period
end % switch
t_amber_2 = yellow(2) + all_red(2) + sti
% calculate amber time after giving green to side road
if t_green_side == 0
    % all t_ev is green for main road and red for side road. t_green_side:input parameter
    t_red_imposed = 0
    t_green_main = t_ev
    yellow_equival_green = 0
else % t_green_side > 0
    t_red_imposed = t_amber_1 + t_amber_2 + t_green_side
    t_green_main = t_ev - t_red_imposed
    switch instant_entry % calculate yellow_equival_green to main road
        case {2,3}
            yellow_equival_green = ( t_amber_1 - all_red(1) ) *
                discharge_rate_green / discharge_rate_yellow
        case {1,4,5,6}
            yellow_equival_green = 0
    end
end

```

```

    end % switch
end % if
green_time_available = t_green_main + yellow_equival_green + 0.01

if t_red_imposed > 0 % produce queue during t_red_imposed
    start = (cycle_entry - 1) * cycle_length + entry_point
    while (t >= start) && (t <= start + t_red_imposed)
        queue = queue + 1
        if pointer >= length(arrival)
            break
        end
        pointer = pointer + 1
        t = (arrival(pointer) - 1) / (2*q)
    end % while
end

% processing queue from t_red_imposed
if queue > 0
    for k = 1:queue
        green_time_available = green_time_available - turn(queue_start + k - 1) *
            discharge_rate_green
        if green_time_available < 0
            break
        end
        discharged_green = k
    end % for
    discharged_for_ev = discharged_green
    queue_start = queue_start + discharged_green
    queue = queue - discharged_green
end % if queue > 0
queue_formed = queue

[fid,message] = fopen('queue_length.dat', 'a');
fprintf(fid,'instant_entry = %-3d ',instant_entry);
fprintf(fid,' cycle_entry = %-3d ',cycle_entry);
fprintf(fid,' queue on approach 3 = %-6.2g \n',queue_formed);
status = fclose(fid);
if status == -1 % =0 if successful closed
    disp('error')
    return
end

```

Appendix G Computer Program for Optimal Control

```

% open a file to record results
[fid,message] = fopen('queue_length.dat', 'w')
fprintf(fid,'queue_length_optimal \n')
status = fclose(fid)
if status == -1 % =0 if successful closed
    disp('error')
    return
end
for instant_entry=1:6
    for cycle_entry=5:10
        queue_length_optimal(cycle_entry,instant_entry);
    end
end

function [queue_length] = queue_length_optimal(cycle_entry,instant_entry);
% generate queue length on approach 1-4 under optimal control for transition 2
% cycle_entry=the number of cycle in which EV arrives
% instant_entry=the time instant at which EV arrives, 1-6 for at 0,green(1)/2,green(1),
%             green(1)+yellow(1), green(1)+yellow(1)+all_red(1),
%             and green(1)+yellow(1)+all_red(1)+green(2)/2, respectively.
% queue_in=queue length from transition 1, imported from queue.dat see below
% using level_1,level_2,level_3,level_4 one-dimensional array as the data structure for
% control sequence
global level_1 level_2 level_3 level_4 green yellow all_red a_matrix b_vector unknown
global green_max green_min
global pp_matrix wx_matrix wuu_matrix r_matrix s_matrix sg_matrix sy_matrix
global volume a_1 a_2 a_3 a_4
volume=[1000,400,800,650]
% volume=traffic volume in veh/hr
% importing vehicle arrivals:
arrival_1 = dlmread('arrival_1.dat');
arrival_2 = dlmread('arrival_2.dat');
arrival_3 = dlmread('arrival_3.dat');
arrival_4 = dlmread('arrival_4.dat');
% arrival=indices of rn where an arrival is produced
% importing turn factors:
turn_1 = dlmread('turn_1.dat');
turn_2 = dlmread('turn_2.dat');
turn_3 = dlmread('turn_3.dat');
turn_4 = dlmread('turn_4.dat');

```

```

% turn=turning factor corresponding to each arrival with straight-through (1.0),
% left-turn (1.4), or right-turn (1.2)
% import queue data from transition 1
% format for each instant_entry (1-6): cycle_entry(5-10),queue on approach 1-4
queue_data = dlmread('queue.dat'); % from fixed held time control
queue_in = queue_get(queue_data,cycle_entry,instant_entry);
% get initial queue on each approach corresponding to input cycle_entry and
% instant_entry

% 1 Select  $P=P'=R=I(4 \times 4 \text{ identity})$  and  $S=I(2 \times 2 \text{ identity})$ 
p_matrix = eye(4);
pp_matrix = eye(4); % pp_matrix stands for p' matrix
r_matrix = eye(4);
s_matrix = eye(2);
% 2 Average green intervals (sec) and queue lengths (veh) from normal operation
queue_mean = [18.8,7.25,11.8,13.55];
green = [60,40];
yellow = [3,3];
all_red = [1,1];
cycle_length = green(1) + yellow(1) + all_red(1) + green(2) + yellow(2) + all_red(2);
green_max = [100,100];
green_min = [7,10];
q = volume / 3600;
for k=1:4
    interval(k) = 0.5 / q(k); % vehicle arrival intervals
end
saturation(1:4) = 2000 / 3600; % in veh/sec for four approaches
for k=1:4
    discharge_rate_green(k) = 1 / saturation(k);
    % for straight-through vehicles during green in sec/veh
end
discharge_rate_yellow = 2 * discharge_rate_green;
% for straight-through vehicles during yellow in sec/veh
yellow_equiv_green = yellow(1)/2;
% transfer yellow interval to equivalent green one
tt = 2160; % total simulation duration in sec.
cycle_number = fix(tt / cycle_length);
% # of cycles
wu_matrix = green;
wu_matrix = [green(1),0, 0,green(2); green(1),0, 0,green(2)];
% wu_matrix stands for mu' matrix
wx_matrix = eye(4);
for k=1:4
    wx_matrix(k,k) = wx_matrix(k,k)/queue_mean(k);
end

```

```

sg_matrix = eye(4);
for k=1:4
    sg_matrix(k,k) = saturation(k);
end
sy_matrix = sg_matrix/2;
% 4 Decide time step duration,t_step, and the total number of time step, step_needed
switch instant_entry
    case 1
        rnc = 58 % time point away from the beginning of next green
        step_needed = 2 % # of time steps needed to recover
        t_step = 29 % in second
    case 2
        rnc = 28
        step_needed = 1
        t_step = 28
    case 3
        rnc = 106
        step_needed = 4
        t_step = 26.5
    case 4
        rnc = 103
        step_needed = 4
        t_step = 25.75
    case 5
        rnc = 102
        step_needed = 4
        t_step = 25.5
    case 6
        rnc = 82
        step_needed = 3
        t_step = 27.3
end % switch

% If queue lengths are still too long to get back to normal operation,
% another normal cycle length will be used for recovery.
% This procedure can proceed until recovery is completed. In our example

% 5 Generate equivalent through vehicle arrivals, A_1/2/3/4(i), for each time step
start_point = cycle_entry * cycle_length - rnc;
a_1(step_needed) = 0; % arrivals on approach 1 during each time step
a_2(step_needed) = 0;
a_3(step_needed) = 0;
a_4(step_needed) = 0;
a_1 = arrival_produce(arrival_1,turn_1,start_point,step_needed,t_step,interval(1));
a_2 = arrival_produce(arrival_2,turn_2,start_point,step_needed,t_step,interval(2));

```



```

a_3 = arrival_produce(arrival_3,turn_3,start_point,step_needed,t_step,interval(3));
a_4 = arrival_produce(arrival_4,turn_4,start_point,step_needed,t_step,interval(4));

% 6 Produce combination of delta1(i) and delta2(i) for current M-step sequences.
% immediatly after transition 1, indication is in green on main road, so
option_1 = [1,2,3,4,9,10,11];
tn_1 = 0;
tn_2 = 0;
tn_3 = 0;
tn_4 = 0;
for k = 1:step_needed
    level_1 = [];
    level_2 = [];
    level_3 = [];
    level_4 = [];
    tn_1 = length(option_1);
    if k == step_needed
        % only level_1 applicable, construct data structure,level_1, for it
        tn_2 = data_structure_1(option_1);
    elseif k + 1 == step_needed
        tn_2 = data_structure_1(option_1);
        tn_3 = data_structure_2(tn_2);
    elseif k + 2 == step_needed
        tn_2 = data_structure_1(option_1);
        tn_3 = data_structure_2(tn_2);
        tn_4 = data_structure_3(tn_3);
    else % k + 2 < step_needed , set other data: level_4 = [delta_1(4),delta_2(4),
        % g_1(4),g_2(4)]
        n_2 = data_structure_1(option_1);
        tn_3 = data_structure_2(tn_2);
        tn_4 = data_structure_3(tn_3);
        data_structure_4(t_step);
        level_4(1,1) = step_needed;
    end
    list_option(k,step_needed,tn_1,tn_2,tn_3,tn_4);
    if k==1
        level_1(1:tn_1,13) = 0; % gc_1(k)
        level_1(1:tn_1,14) = 0; % gc_2(k)
    elseif k>1
        level_1(1:tn_1,12) = operation(k-1,12) + operation(k-1,2);
        % gc_1(k)= gc_1(k-1)+g1(k-1)
        level_1(1:tn_1,13) = operation(k-1,13) + operation(k-1,3);
        % gc_2(k)= gc_2(k-1)+g2(k-1)
    end
    pi_optimal =1000;

```

```

% unknown
vector=(g_1(i),g_2(i),q_1(i),q_2(i),q_3(i),q_4(i),q_1(i+1),q_2(i+1),q_3(i+1),q_4(i+1),
% lambda_1(i),lambda_2(i),lambda_3(i),lambda_4(i),lambda_1(i+1), lambda_2(i+1),
% lambda_3(i+1),lambda_4(i+1),sigma(i))'
% first construct coefficient matrix, a_matrix,
% and right-hand side column vector, b_vector
% a_matrix(equation#,variable#), b_vector(equation#)
if k == 1
    queue_initial = queue_in;
end
% different at each instant_entry and cycle_entry, queue formed at the end of
% transition 1
queue_desired = queue_mean; % =[18.8,7.25,11.8,13.55]
equation_tn = 0;
a_matrix = [];
b_vector = [];
equation_tn = eq_initial(equation_tn,queue_initial);
% construct eqs for initial queues (eq 12)
list_equation_coefficient;
step_remained = step_needed - k + 1;
equation_tn = eq_final(equation_tn,queue_desired,step_remained);
% construct eqs for final desired queues (eq 13)
list_equation_coefficient;
eq_fixed_0 = equation_tn % used for reset a_matrix,b_vector after each iteration
for k_1 = 1:tn_1
    [g_1(1),g_2(1),delta_1(1),delta_2(1)] = get_data(1,k_1);
    % indicator=1 for level_1
    equation_tn = equation_add(1,eq_fixed_0,g_1(1),g_2(1));
    % indicator=1 for level_1
    % construct eqs for q_1(1)=0 or g_2(1)=0
    list_equation_coefficient;
    step_n = k;
    equation_tn = eq_level(1,equation_tn,t_step,step_needed,step_n,
    delta_1(1),delta_2(1),a_1,a_2,a_3,a_4);
    % construct eqs for level_1; (indicator=1)
    list_equation_coefficient;
    eq_fixed_1 = equation_tn
    gc_1(step_n) = level_1(k_1, 13);
    gc_2(step_n) = level_1(k_1, 14);
    seq(1) = level_1(k_1,1) % sequence# for this option
    if k == step_needed
        unknown = a_matrix \ b_vector';
        unknown = unknown'
        if unknown(1)<-0.1
            % set g_1 = green_min(1) while g_1 is (applicable & <0)

```

```

a_matrix(equation_tn+1,1:19)=0;
a_matrix(equation_tn+1,1)=1;
b_vector(equation_tn+1)=green_min(1);
equation_tn = equation_tn +1;
unknown = a_matrix \ b_vector';
unknown = unknown'
end
if unknown(2)<-0.1
    % set g_2 = green_min(2) while g_2 is (applicable & <0)
    a_matrix(equation_tn+1,1:19)=0;
    a_matrix(equation_tn+1,2)=1;
    b_vector(equation_tn+1)=green_min(2);
    equation_tn = equation_tn +1;
    unknown = a_matrix \ b_vector';
    unknown = unknown'
end
[gc_1,gc_2] = constraint_ver(1,seq,gc_1,gc_2,t_step,step_n);
% verification of constraints
option_entry = [];
option_entry = k_1
fill_back(1,option_entry);
% fill back data structure for level 1(indicator=1)
pi = pi_calculate(1,queue_mean);
% calculate PI for only one level (indicator=1)
if pi < pi_optimal % select optimal option
    pi_optimal = pi;
    level_1(k_1,10) = pi; % fill back pi into data structure
    optimal_list(1,1:21)=level_1(k_1,1:21); % for level 1
end
initialize(eq_fixed_0);
continue % goto next k_1
end
tn_2 = level_1(k_1,11)
for k_2 = 1:tn_2
    pointer_2 = level_1(k_1,21) + k_2 -1;
    % pointer to sub_option position in level_2
    [g_1(2),g_2(2),delta_1(2),delta_2(2)] = get_data(2,pointer_2);
    equation_tn = equation_add(2,eq_fixed_1,g_1(2),g_2(2));
    equation_tn = eq_level(2,equation_tn,t_step,step_needed,step_n,
    delta_1(2),delta_2(2),a_1,a_2,a_3,a_4);
    % construct eqs for level_2 (indicator=2)
    list_equation_coefficient
    eq_fixed_2 = equation_tn
    seq(2) = level_2(pointer_2,1) % sequence# for this option in level_2
    if k + 1 == step_needed

```

```

unknown = a_matrix \ b_vector';
unknown = unknown'
if unknown(1)<-0.1 % set g_1 = green_min(1) while g_1 is (applicable & <0)
    a_matrix(equation_tn+1,1:19) =0;
    a_matrix(equation_tn+1,1) =1;
    b_vector(equation_tn+1) =green_min(1);
    equation_tn = equation_tn +1;
    unknown = a_matrix \ b_vector';
    unknown = unknown'
end
if unknown(2)<-0.1 % set g_2 = green_min(2) while g_2 is (applicable & <0)
    a_matrix(equation_tn+1,1:19) =0;
    a_matrix(equation_tn+1,2) =1;
    b_vector(equation_tn+1) =green_min(2);
    equation_tn = equation_tn +1;
    unknown = a_matrix \ b_vector';
    unknown = unknown'
end
if unknown(19+1)<-0.1
    % set g_1 = green_min(1) while g_1 is (applicable & <0)
    a_matrix(equation_tn+1,1:19+11) =0;
    a_matrix(equation_tn+1,19+1) =1;
    b_vector(equation_tn+1) =green_min(1);
    equation_tn = equation_tn +1;
    unknown = a_matrix \ b_vector';
    unknown = unknown'
end
if unknown(19+2)<-0.1
    % set g_2 = green_min(2) while g_2 is (applicable & <0)
    a_matrix(equation_tn+1,1:19+11) =0;
    a_matrix(equation_tn+1,19+2) =1;
    b_vector(equation_tn+1) =green_min(2);
    equation_tn = equation_tn +1;
    unknown = a_matrix \ b_vector';
    unknown = unknown'
end
[gc_1,gc_2] = constraint_ver(2,seq,gc_1,gc_2,t_step,step_n);
% verification of constraints
option_entry = [];
option_entry = [k_1,pointer_2];
fill_back(2,option_entry);
% fill back data structure for level 1&2
pi = pi_calculate(2,queue_mean);
% calculate PI for only two levels (indicator=2)
if pi < pi_optimal % select optimal option

```

```

    pi_optimal = pi;
    level_1(k_1,10) = pi; % fill back pi into data structure
    optimal_list(1,1:21) = level_1(k_1,1:21); % for level 1
    optimal_list(2,1:21) = level_2(pointer_2,1:21); % for level 2
end
initialize(eq_fixed_1);
continue % goto next k_2
end
tn_3 = level_2(k_2,11);
for k_3 = 1:tn_3
    pointer_3 = level_2(k_2,21) + k_3 - 1;
    % pointer to sub-option position in level_3
    [g_1(3),g_2(3),delta_1(3),delta_2(3)] = get_data(3,pointer_3);
    equation_tn = equation_add(3,eq_fixed_2,g_1(3),g_2(3));
    equation_tn = eq_level(3,equation_tn,t_step,step_needed,step_n,
    delta_1(3),delta_2(3),a_1,a_2,a_3,a_4);
    % construct eqs for level_3 (indicator=3)
    eq_fixed_3 = equation_tn
    if k + 2 < step_needed
        g_1(4)=level_4(1,2);
        g_2(4)=level_4(1,3);
        delta_1(4) = 1;
        delta_2(4) = 1;
        for k_4 = (k + 3):step_needed
            % construct eqs for fixed g(1) and g(2)
            a_matrix(equation_tn,1) = 1;
            b_vector(equation_tn) = g_1(4);
            equation_tn = equation_tn + 1;
            a_matrix(equation_tn,2) = 1;
            b_vector(equation_tn) = g_2(4);
            equation_tn = equation_tn + 1;
            %construct eqs for level k_4
            equation_tn = eq_level(4,equation_tn,t_step,step_needed,step_n,
            delta_1(4),delta_2(4),a_1,a_2,a_3,a_4);
        end % for k_4 (indicator=4)
    end
    unknown = a_matrix \ b_vector';
    unknown = unknown'
    if unknown(1) < -0.1 % set g_1 = green_min(1) while g_1 is (applicable & < 0)
        a_matrix(equation_tn+1,1:19) = 0;
        a_matrix(equation_tn+1,1) = 1;
        b_vector(equation_tn+1) = green_min(1);
        equation_tn = equation_tn + 1;
        unknown = a_matrix \ b_vector';
        unknown = unknown'
    end
end

```

```

end
if unknown(2)<-0.1 % set g_2 = green_min(2) while g_2 is (applicable & <0)
    a_matrix(equation_tn+1,1:19)=0;
    a_matrix(equation_tn+1,2)=1;
    b_vector(equation_tn+1)=green_min(2);
    equation_tn = equation_tn +1;
    unknown = a_matrix \ b_vector';
    unknown = unknown'
end
if unknown(19+1)<-0.1
    % set g_1 = green_min(1) while g_1 is (applicable & <0)
    a_matrix(equation_tn+1,1:19+11)=0;
    a_matrix(equation_tn+1,19+1)=1;
    b_vector(equation_tn+1)=green_min(1);
    equation_tn = equation_tn +1;
    unknown = a_matrix \ b_vector';
    unknown = unknown'
end
if unknown(19+2)<-0.1
    % set g_2 = green_min(2) while g_2 is (applicable & <0)
    a_matrix(equation_tn+1,1:19+11)=0;
    a_matrix(equation_tn+1,19+2)=1;
    b_vector(equation_tn+1)=green_min(2);
    equation_tn = equation_tn +1;
    unknown = a_matrix \ b_vector';
    unknown = unknown'
end
if unknown(19+11+1)<-0.1
    % set g_1 = green_min(1) while g_1 is (applicable & <0)
    a_matrix(equation_tn+1,1:19+11+11)=0;
    a_matrix(equation_tn+1,19+11+1)=1;
    b_vector(equation_tn+1)=green_min(1);
    equation_tn = equation_tn +1;
    unknown = a_matrix \ b_vector';
    unknown = unknown'
end
if unknown(19+11+2)<-0.1
    % set g_2 = green_min(2) while g_2 is (applicable & <0)
    a_matrix(equation_tn+1,1:19+11+11)=0;
    a_matrix(equation_tn+1,19+11+2)=1;
    b_vector(equation_tn+1)=green_min(2);
    equation_tn = equation_tn +1;
    unknown = a_matrix \ b_vector';
    unknown = unknown'
end
end

```

```

seq(3) = level_3(pointer_3,1)
[gc_1,gc_2] = constraint_ver(3,seq,gc_1,gc_2,t_step,step_n);
% verification of constraints
option_entry = [];
option_entry = [k_1,pointer_2,pointer_3];
fill_back(3,option_entry);
% fill back data structure for level 3
pi = pi_calculate(3,queue_mean);
% calculate PI for 3 levels (indicator=3)
if pi < pi_optimal % select optimal option
    pi_optimal = pi;
    level_1(k_1,10) = pi; % fill back pi into data structure
    optimal_list(1,1:21) = level_1(k_1,1:21); % for level 1
    optimal_list(2,1:21) = level_2(k_2,1:21); % for level 2
    optimal_list(3,1:21) = level_3(k_3,1:21); % for level 3
end
initialize(eq_fixed_2);
end % for k_3
initialize(eq_fixed_1);
end % for k_2
initialize(eq_fixed_0);
end % for k_1
% change queue_initial
queue_initial(1:4) = [];
queue_initial(1:4) = optimal_list(1,4:7)
% change option_1
option_1 = [];
sub_tn = optimal_list(1,11);
pointer = optimal_list(1,21);
for kk = 1:sub_tn
    option_1(kk) = level_2(pointer,1);
    pointer = pointer + 1;
end
operation(k,1:21) = optimal_list(1,1:21) % the final operation adopted for step k
end % for k = 1:step_needed
queue_length = 0
for k = 1:4
    queue_length(k) = operation(step_needed, 3+k) % queue at the end of transition 2
end
list_queue(instant_entry,cycle_entry,queue_length)
[fid,message] = fopen('queue_length.dat','a'); % 1 for standard, 2 for error
fprintf(fid,'instant_entry = %-3d ',instant_entry);
fprintf(fid,' cycle_entry = %-3d \n',cycle_entry);
fprintf(fid,' transition period = %-3d \n',rnc);
fprintf(fid,' step_needed = %-3d \n',step_needed);

```

```

fprintf(fid,' t_step = %-3d \n',t_step);
fprintf(fid,' initial queue length on:\n');
for kk = 1:4
    fprintf(fid,' approach %-3d',kk);
    fprintf(fid,'= %-6.2f',queue_in(kk));
end
fprintf(fid,' \n');
fprintf(fid,' final queue length on:\n');
for kk = 1:4
    fprintf(fid,' approach %-3d',kk);
    fprintf(fid,'= %-6.2f',queue_length(kk));
end
fprintf(fid,' \n');
fprintf(fid,' vehicle arrivals during each time_step on:\n');
fprintf(fid,' approach 1 = ');
for kk = 1:step_needed
    fprintf(fid,'%6.2g ',a_1(kk));
end
fprintf(fid,' \n');
fprintf(fid,' approach 2 = ');
for kk = 1:step_needed
    fprintf(fid,'%6.2g ',a_2(kk));
end
fprintf(fid,' \n');
fprintf(fid,' approach 3 = ');
for kk = 1:step_needed
    fprintf(fid,'%6.2g ',a_3(kk));
end
fprintf(fid,' \n');
fprintf(fid,' approach 4 = ');
for kk = 1:step_needed
    fprintf(fid,'%6.2g ',a_4(kk));
end

fprintf(fid,' \n');
fprintf(fid,' signal timing for each time step: \n');
fprintf(fid,' green time for main road= ');
for kk = 1:step_needed
    fprintf(fid,'%6.2f',operation(kk,2));
end
fprintf(fid,' \n');
fprintf(fid,' green time for side road= ');
for kk = 1:step_needed
    fprintf(fid,'%6.2f',operation(kk,3));
end

```



```

fprintf(fid,'\n');

status = fclose(fid);
if status == -1 % =0 if successful closed
    disp('error')
    return
end

% end of main program
% -----
% -----
% Following functions used in above main program

function a = arrival_produce(arrival,turn,start_point,step_needed,t_step,interval)
% produce arrivals on approach corresponding to "arrival" during recovery
% and transfer vehicle arrivals to equivalent through vehicle numbers
a(step_needed) = 0;
pointer = 1;
t = (arrival(pointer) - 1) * interval;
% arrival time in sec, and the possibly earliest one arrives at 0 sec
for k=1:step_needed
    start = start_point + t_step *(k-1);
    while (t < start + t_step)
        if (t < start)
            pointer = pointer + 1;
            t = (arrival(pointer) - 1) * interval;
            continue
        end
        if (t >= start + t_step)
            break
        end
        a(k) = a(k) + 1*turn(pointer);
        if pointer >= length(arrival)
            break
        end
        pointer = pointer + 1;
        t = (arrival(pointer) - 1) * interval;
    end
end

function [sub_option,parameter] = option_generate(option_in)
% generate sub-options following option_in
% output sub-option list,parameter=[g1,g2,delta1,delta2] for option_in
switch option_in
    case {1,7,11,12}

```

```

        sub_option = [1,2,3,4,9,10,11];
    case {3,5,9,14}
        sub_option = [5,6,7,8,12,13,14];
    case {2,8,13}
        sub_option = [5,6,7,8];
    case {4,6,10}
        sub_option = [1,2,3,4];
end
% output parameter=[g1,g2,delta1,delta2] for option_in
%=-9 (if not applicable), =999 (if applicable), or = actual value (otherwise)
switch option_in
    case 1
        parameter = [999,-9,-9,-9];
    case 2
        parameter = [999,-9,1,-9];
    case 3
        parameter = [999,999,1,-9];
    case 4
        parameter = [999,999,1,1];
    case 5
        parameter = [-9,999,-9,-9];
    case 6
        parameter = [-9,999,-9,1];
    case 7
        parameter = [999,999,-9,1];
    case 8
        parameter = [999,999,1,1];
    case 9
        parameter = [-9,999,1,-9];
    case 10
        parameter = [-9,999,1,1];
    case 11
        parameter = [999,999,1,1];
    case 12
        parameter = [999,-9,-9,1];
    case 13
        parameter = [999,-9,1,1];
    case 14
        parameter = [999,999,1,1];
end % switch

```

```

function tn = data_structure_1(option)
% construct data structure for level 1, and fill in info except pi
global level_1 level_2 level_3 level_4

```

```

tn_1 = length(option); % total options for level 1
pointer_2 = 1;
tn_2 = 0;
for k=1:tn_1
    level_1(k,1) = option(k); % option #
    level_1(k,21) = pointer_2; % pointer to option_1(k)'s first sub-optoin in level 2
    [sub_option,parameter] = option_generate(option(k));
    % fill in g1,g2,delta1,delta2
    level_1(k,2:3) = parameter(1:2);
    level_1(k,8:9) = parameter(3:4);
    tn = length(sub_option);
    level_1(k,11) = tn; % fill in sub-tn
    for h=1:tn % fill in sub-option # in level 2
        level_2(tn_2+h,1) = sub_option(h); % sub-option #
    end
    pointer_2 = pointer_2 + tn;
    tn_2 = tn_2 + tn; % total options for level 2
end
tn = tn_2;

```

```

function tn = data_structure_2(tn_in)
% construct data structure for level 2, and fill in info except pi
global level_1 level_2 level_3 level_4
tn_2 = tn_in;
pointer_3 = 1;
tn_3 = 0;
for k=1:tn_2
    level_2(k,21) = pointer_3; % pointer to option(k)'s first sub-optoin in level 3
    option_in = level_2(k,1);
    [sub_option,parameter] = option_generate(option_in);
    % fill in g1,g2,delta1,delta2
    level_2(k,2:3) = parameter(1:2);
    level_2(k,8:9) = parameter(3:4);
    tn = length(sub_option);
    level_2(k,11) = tn; % fill in sub-tn
    for h=1:tn % fill in sub-option # in level 3
        level_3(tn_3+h,1) = sub_option(h); % sub-option #
    end
    pointer_3 = pointer_3 + tn;
    tn_3 = tn_3 + tn; % total options for level 3
end
tn = tn_3;

```

```

function tn = data_structure_3(tn_in)
% construct data structure for level 3, and fill in info except pi

```

```

global level_1 level_2 level_3 level_4
tn_3 = tn_in;
pointer_4 = 1;
tn_4 = 0;
for k=1:tn_3
    level_3(k,21) = pointer_4; % pointer to option(k)'s first sub-optoin in level 4
    option_in = level_3(k,1);
    [sub_option,parameter] = option_generate(option_in);
    % fill in u1,u2,delta1,delta2
    level_3(k,2:3) = parameter(1:2);
    level_3(k,8:9) = parameter(3:4);
    tn = length(sub_option);
    level_3(k,11) = tn; % fill in sub-tn
    pointer_4 = pointer_4 + tn;
    tn_4 = tn_4 + tn; % total options for level 4
end
tn = tn_4;

function data_structure_4(t_step)
% 7 Set other delta1(i) = delta2(i) =1 and
% g1(i)=g2(i)=(T - y(1)-y(2)-AR(1)-AR(2))/(g1 + g1 )
global green yellow all_red level_4
tep = (t_step - yellow(1) - yellow(2) - all_red(1) - all_red(2))/(green(1) + green(2));
g_1 = green(1) * tep;
g_2 = green(2) * tep;
% u_1 = g_1 / green(1)
% u_2 = g_2 / green(2)
level_4(1,2) = g_1;
level_4(1,3) = g_2;
delta_1 = 1;
delta_2 = 1;
level_4(1,8) = delta_1;
level_4(1,9) = delta_2;

function list_option(step_start,step_needed,tn_1,tn_2,tn_3,tn_4)
global level_1 level_2 level_3 level_4
[fid,message] = fopen('option_list.dat', 'w'); % 1 for stdandard, 2 for error
fprintf(fid,'options for levels 1-3 \n');
fprintf(fid,'level_1 record \n');
for record=1:tn_1
    fprintf(fid,'record= %3d',record);
    for position=1:13
        fprintf(fid,'%6d',level_1(record,position));
    end
    fprintf(fid,'%6d \n',level_1(record,21));

```

```

end
if step_start == step_needed
    status = fclose(fid);
    if status == -1 % =0 if successful closed
        disp('error');
        return
    end
    return
end
fprintf(fid,'level_2 record \n');
for record=1:tn_2
    fprintf(fid,'record= %3d',record);
    for position=1:13
        fprintf(fid,'%6d',level_2(record,position));
    end
    fprintf(fid,'%6d \n',level_2(record,21));
end
if step_start +1 == step_needed
    status = fclose(fid);
    if status == -1 % =0 if successful closed
        disp('error');
        return
    end
    return
end
fprintf(fid,'level_3 record \n');
for record=1:tn_3
    fprintf(fid,'record= %3d',record);
    for position=1:13
        fprintf(fid,'%6d',level_3(record,position));
    end
    fprintf(fid,'%6d \n',level_3(record,21));
end
if step_start +2 == step_needed
    status = fclose(fid);
    if status == -1 % =0 if successful closed
        disp('error');
        return
    end
    return
end
fprintf(fid,'level_4 record \n');
record = 1
for position=1:9
    fprintf(fid,'%6d',level_4(record,position));

```

```

end
status = fclose(fid);
if status == -1 % =0 if successful closed
    disp('error');
    return
end

function list_equation_coefficient(equation_tn)
global a_matrix b_vector
[fid,message] = fopen('equation_coefficient.dat', 'w'); % 1 for stdandard, 2 for error
fprintf(fid,'coefficient matrix \n');
[row,column] = size(a_matrix)
fprintf(fid,'Total equations, total unknowns= %-6d %-6d\n',row,column);
for kk = 1:row
    fprintf(fid,'Equation # = %-3d \n',kk);
    for jj = 1:column
        fprintf(fid,'%4.1d',a_matrix(kk,jj));
        nn = fix(jj / 10);
        mm = jj - nn * 10;
        if mm == 0
            fprintf(fid,'(%-3d)',10*nn);
            % fprintf(fid,')')
        end
    end
    fprintf(fid,' b= %-4.1d \n',b_vector(kk));
end
status = fclose(fid);
if status == -1 % =0 if successful closed
    disp('error');
    return
end

function [g_1,g_2,delta_1,delta_2] = get_data(indicator,k)
global level_1 level_2 level_3
if indicator == 1
    level = level_1;
elseif indicator == 2
    level = level_2;
elseif indicator == 3
    level = level_3;
end
g_1 = level(k,2);
g_2 = level(k,3);
delta_1 = level(k,8);
delta_2 = level(k,9);

```

```

if g_1 == -9 % not applicable
    g_1 = 0;
end
if g_2 == -9 % not applicable
    g_2 = 0;
end
if delta_1 == -9 % not applicable
    delta_1 = 0;
end
if delta_2 == -9 % not applicable
    delta_2 = 0;
end

function equation_tn_out = eq_initial(equation_tn,queue_initial)
global a_matrix b_vector
% construct eqs for initial queues (eq 12)
for k=1:4
    a_matrix(equation_tn +k,1:19) = 0; % inititalize a
    a_matrix(equation_tn +k,2+k) = 1;
end
for k=1:4
    b_vector(equation_tn +k) = queue_initial(k);
end
equation_tn_out = equation_tn + 4;

function equation_tn_out = eq_final(equation_tn,queue_desired,step_remaining)
global a_matrix b_vector pp_matrix wx_matrix
% construct eqs for final desired queues (eq 13)
tep = pp_matrix * wx_matrix;
if step_remaining == 1
    unknown_level = 19;
    unknown_total = 19;
    start = 0;
    offset_q = 6; % offset for q(N+1)
    offset_lambda = 14; % offset for lamda(N+1)
elseif step_remaining == 2
    unknown_level = 11; % total unknowns for level 2
    unknown_total = 19 + 11 *(step_remaining -1); % total unknowns for all levels
    start = unknown_total -11; % pointer at the begining for level 2
    offset_q = 2; % offset for q(N+1)
    offset_lambda = 6; % offset for lamda(N+1)
elseif step_remaining > 2
    unknown_level = 11; % total unknowns for level 3
    unknown_total = 19 + 11 *(3 -1); % total unknowns for all levels
    start = unknown_total -11; % pointer at the begining for level 3

```

```

    offset_q = 2; % offset for q(N+1)
    offset_lambda = 6; % offset for lambda(N+1)
end
for k=1:4
    a_matrix(equation_tn +k,1:unknown_total) = 0; % initialize
    a_matrix(equation_tn +k,start+offset_q+k) = tep(k,k); % q1(N+1)~q4(N+1)
end
for k=1:4
    a_matrix(equation_tn +k,start+offset_lambda+k) = -1; % lambda1(N+1)~lambda4(N+1)
end
for k=1:4
    b_vector(equation_tn +k) = queue_desired(k) *tep(k,k);
end
equation_tn_out = equation_tn + 4;

function equation_tn_out = equation_add(indicator,equation_tn,g_1,g_2)
% indicator=1~3 for level 1~3
global a_matrix b_vector
% construct eqs for q(1)=0 or g(2)=0
if indicator == 1
    unknown_level = 19;
    start = 0;
else
    unknown_level = 11;
    start = 19 + 11 * (indicator - 2);
end
if g_1 == 0
    a_matrix(equation_tn +1,start+1:start+unknown_level) = 0;
    a_matrix(equation_tn +1,start+1) = 1;
    b_vector(equation_tn +1) = 0;
    equation_tn = equation_tn + 1;
end
if g_2 == 0
    a_matrix(equation_tn +1,start+1:start+unknown_level) = 0;
    a_matrix(equation_tn +1,start+2) = 1;
    b_vector(equation_tn +1) = 0;
    equation_tn = equation_tn + 1;
end
equation_tn_out = equation_tn;

function equation_tn_out = eq_level(indicator,equation_tn,t_step,step_needed,step_n,
delta_1,delta_2,a_1,a_2,a_3,a_4)
% construct eqs for level_1,2,3,4, respectively, indicated by indicator=1/2/3/4
% a_# = vehicle arrivals
% step_n = 1~step_needed for a_#(step_n +indicator -1)

```



```

global green yellow all_red a_matrix b_vector
global r_matrix s_matrix wx_matrix wuu_matrix sg_matrix sy_matrix
global volume
if indicator == 1
    unknown_level = 19;
    start = 0;
else
    unknown_level = 11;
    start = 19 + 11 * (indicator - 2);
end
for k = 1:11 % 11 quations for each more level
    a_matrix(equation_tn + k, start + 1 : start + unknown_level) = 0;
end
% for equation (1b)
a_matrix(equation_tn + 1, start + 1 : start + 2) = 1; % g1(i) and g2(i)
b_vector(equation_tn + 1) = t_step - delta_1 * (yellow(1) + all_red(1)) -
delta_2 * (yellow(2) + all_red(2));
% for equation (2b)
a_matrix(equation_tn + 2, start + 1) = sg_matrix(1,1); % g1(i),g2(i)
a_matrix(equation_tn + 3, start + 2) = sg_matrix(2,2);
a_matrix(equation_tn + 4, start + 1) = sg_matrix(3,3);
a_matrix(equation_tn + 5, start + 2) = sg_matrix(4,4);
if indicator == 1
    offset = 2; % pointer at the beginging of q(i)
elseif indicator == 2
    offset = -19 + 6;
else
    offset = -11 + 2;
end
for k = 1:4
    a_matrix(equation_tn + 1 + k, start + offset + k) = -1; % q(i)
end
if indicator == 1
    offset = 6; % pointer at the beginging of q(i+1)
else
    offset = 2;
end
for k = 1:4
    a_matrix(equation_tn + 1 + k, start + offset + k) = 1; % q(i+1)
end
if (step_n + indicator - 1) > step_needed
    saturation = volume / 3600; % in veh/sec
    arrival_average = saturation * t_step;
    a_1(step_n + indicator - 1) = arrival_average(1);
    a_2(step_n + indicator - 1) = arrival_average(2);
end

```

```

    a_3(step_n + indicator - 1) = arrival_average(3);
    a_4(step_n + indicator - 1) = arrival_average(4);
end
b_vector(equation_tn + 2) = a_1(step_n + indicator - 1) - sy_matrix(1,1) * delta_1 *
yellow(1);
b_vector(equation_tn + 3) = a_2(step_n + indicator - 1) - sy_matrix(2,2) * delta_2 *
yellow(2);
b_vector(equation_tn + 4) = a_3(step_n + indicator - 1) - sy_matrix(3,3) * delta_1 *
yellow(1);
b_vector(equation_tn + 5) = a_4(step_n + indicator - 1) - sy_matrix(4,4) * delta_2 *
yellow(2);
% for equation (10)
tep = r_matrix * wx_matrix;
if indicator == 1
    offset = 2; % pointer at the beginging of q(i)
elseif indicator == 2
    offset = -19 + 6;
else
    offset = -11 + 2;
end
for k=1:4
    a_matrix(equation_tn + 5 + k, start + offset + k) = tep(k,k); % q(i)
end
if indicator == 1
    offset = 10; % pointer at the beginging of lamda(i)
elseif indicator == 2
    offset = -19 + 14;
else
    offset = -11 + 6;
end
for k=1:4
    a_matrix(equation_tn + 5 + k, start + offset + k) = -1; % lamda(i)
end
if indicator == 1
    offset = 14; % pointer at the beginging of lamda(i+1)
else
    offset = 6;
end
for k=1:4
    a_matrix(equation_tn + 5 + k, start + offset + k) = 1; % lamda(i+1)
end
b_vector(equation_tn + 6:equation_tn + 9) = 0;
% for equation (11)
tep = [1/green(1), 1/green(2)];
tep = s_matrix * tep;

```

```

a_matrix(equation_tn +10,start+1) = tep(1,1); % g1(i),g2(i)
a_matrix(equation_tn +11,start+2) = tep(2,1);
tep = -(wuu_matrix)' * sg_matrix * wx_matrix; % 2X4 matrix
if indicator == 1
    offset = 10; % pointer at the beginging of lamda(i)
elseif indicator == 2
    offset = -19 +14;
else
    offset = -11 +6;
end
a_matrix(equation_tn +10,start+offset+1:start+offset+4) = tep(1,1:4); % lamda(i)
a_matrix(equation_tn +11,start+offset+1:start+offset+4) = tep(2,1:4);
if indicator == 1
    position = start +19; % position of sigma(i)
else
    position = start +11;
end
a_matrix(equation_tn +10,position) = green(1); % sigma(i)
a_matrix(equation_tn +11,position) = green(2);
b_vector(equation_tn +10:equation_tn +11) = 0;

equation_tn_out = equation_tn +11;

function [gc_1,gc_2] = constraint_ver(indicator,seq,gc_1,gc_2,t_step,step_n)
% verification of constraints and fill back data structure
% indicator =1/2/3 stands for total # of levels included in the unknown
% seq(1:3)=sequence # for options in level_1/2/3
global green_max green_min sg_matrix unknown
global a_1 a_2 a_3 a_4
for k =1:indicator
    if k == 1
        start = 0; % pointer at the beginging of unknown for this level
        offset = 6; % starting position of q(i+1) for level 1
    else
        start = 19 +11* (k -2);
        offset = 2; % starting position of q(i) for level 2&3
    end
    ksai(5) = 0;
    ksai(6) = 0;
    gc_1(step_n +k) = (gc_1(step_n +k -1) + unknown(start +1) ) *(1 -ksai(5));
    % seq(1,7,12)
    gc_2(step_n +k) = (gc_2(step_n +k -1) + unknown(start +2) ) *(1 -ksai(6));
    % seq(5,3,9)
    if unknown(start +1) < 0.1 % g_1 ==0 considering calculation error
        gc_1(step_n +k) = 0; % seq(5,6,9,10); ksai(5) = 1
    end
end

```

```

elseif unknown(start + 1) > 0.1 % g_1 > 0 considering calculation error
    if seq(k) == 2 || seq(k) == 3 || seq(k) == 4 || seq(k) == 8 || seq(k) == 13 || seq(k) == 14
        % g_1 <> 0 followed by delta_1 <> 0
        gc_1(step_n + k) = 0;
    elseif seq(k) == 11
        % g_1 <> 0 proceeded by delta <> 0
        gc_1(step_n + k) = unknown(start + 1);
    end
end
end
if unknown(start + 1) > 0.1 % g_1 > 0: begin to check g1_max
    ksai(2) = 0;
    tep = green_max(1) - gc_1(step_n + k - 1);
    if unknown(start + 1) > tep
        dif = unknown(start + 1) - tep;
        unknown(start + 1) = tep;
        unknown(start + 2) = unknown(start + 2) + dif;
        min_q1 = min_choose(a_1(step_n + k - 1), dif * sg_matrix(1, 1));
        min_q3 = min_choose(a_3(step_n + k - 1), dif * sg_matrix(3, 3));
        min_q2 = min_choose(a_2(step_n + k - 1), dif * sg_matrix(2, 2));
        min_q4 = min_choose(a_4(step_n + k - 1), dif * sg_matrix(4, 4));
        unknown(start + offset + 1) = unknown(start + offset + 1) + min_q1; % q1(i+1)
        unknown(start + offset + 3) = unknown(start + offset + 3) + min_q3; % q3(i+1)
        unknown(start + offset + 2) = unknown(start + offset + 2) - min_q2; % q2(i+1)
        unknown(start + offset + 4) = unknown(start + offset + 4) - min_q4; % q4(i+1)
    end
end
if ((unknown(start + 1) > 0.1 && gc_1(step_n + k) == 0) ||
    (unknown(start + 1) <= 0.1 && gc_1(step_n + k - 1) > 0))
    % g_1 <> 0 & gc_1(i) == 0 or g_1 == 0 & gc_1(i-1) <> 0
    % begin to check g1_min
    ksai(1) = 0;
    tep = green_min(1) - gc_1(step_n + k - 1);
    if unknown(start + 1) < tep % g_1
        dif = tep - unknown(start + 1);
        unknown(start + 1) = tep;
        unknown(start + 2) = unknown(start + 2) - dif;
        min_q1 = min_choose(a_1(step_n + k - 1), dif * sg_matrix(1, 1));
        min_q3 = min_choose(a_3(step_n + k - 1), dif * sg_matrix(3, 3));
        min_q2 = min_choose(a_2(step_n + k - 1), dif * sg_matrix(2, 2));
        min_q4 = min_choose(a_4(step_n + k - 1), dif * sg_matrix(4, 4));
        unknown(start + offset + 1) = unknown(start + offset + 1) - min_q1; % q1(i+1)
        unknown(start + offset + 3) = unknown(start + offset + 3) - min_q3; % q3(i+1)
        unknown(start + offset + 2) = unknown(start + offset + 2) + min_q2; % q2(i+1)
        unknown(start + offset + 4) = unknown(start + offset + 4) + min_q4; % q4(i+1)
    end
end

```

```

end
if unknown(start + 2) < 0.1 % g_2 == 0 considering calculation error
    gc_2(step_n + k) = 0; % seq(1,2,12,13): ksai(6) = 1
elseif unknown(start + 2) > 0.1 % g_2 > 0 considering calculation error
    if seq(k) == 6 || seq(k) == 7 || seq(k) == 8 || seq(k) == 10 || seq(k) == 11 || seq(k) == 4
        % g_2 <> 0 followed by delta_2 <> 0
        gc_2(step_n + k) = 0;
    elseif seq(k) == 14
        % g_2 <> 0 proceeded by delta_2 <> 0
        gc_2(step_n + k) = unknown(start + 2);
    end
end
end
if unknown(start + 2) > 0.1 % g_2 > 0: begin to check g2_max
    ksai(4) = 0;
    tep = green_max(2) - gc_2(step_n + k - 1);
    if unknown(start + 2) > tep
        dif = unknown(start + 2) - tep;
        unknown(start + 2) = tep;
        unknown(start + 1) = unknown(start + 2) + dif;
        min_q1 = min_choose(a_1(step_n + k - 1), dif * sg_matrix(1,1));
        min_q3 = min_choose(a_3(step_n + k - 1), dif * sg_matrix(3,3));
        min_q2 = min_choose(a_2(step_n + k - 1), dif * sg_matrix(2,2));
        min_q4 = min_choose(a_4(step_n + k - 1), dif * sg_matrix(4,4));
        unknown(start + offset + 1) = unknown(start + offset + 1) + min_q1; % q1(i+1)
        unknown(start + offset + 3) = unknown(start + offset + 3) + min_q3; % q3(i+1)
        unknown(start + offset + 2) = unknown(start + offset + 2) - min_q2; % q2(i+1)
        unknown(start + offset + 4) = unknown(start + offset + 4) - min_q4; % q4(i+1)
    end
end
if (unknown(start + 2) > 0.1 && gc_2(step_n + k) == 0) ||
(unknown(start + 2) <= 0.1 && gc_2(step_n + k - 1) > 0)
    % g_2 <> 0 & gc_2(i) == 0 or g_2 == 0 & gc_2(i-1) <> 0
    % begin to check g2_min
    ksai(3) = 0;
    tep = green_min(2) - gc_2(step_n + k - 1);
    if unknown(start + 2) < tep % g_2
        dif = tep - unknown(start + 2);
        unknown(start + 2) = tep;
        unknown(start + 1) = unknown(start + 1) - dif;
        min_q1 = min_choose(a_1(step_n + k - 1), dif * sg_matrix(1,1));
        min_q3 = min_choose(a_3(step_n + k - 1), dif * sg_matrix(3,3));
        min_q2 = min_choose(a_2(step_n + k - 1), dif * sg_matrix(2,2));
        min_q4 = min_choose(a_4(step_n + k - 1), dif * sg_matrix(4,4));
        unknown(start + offset + 1) = unknown(start + offset + 1) - min_q1; % q1(i+1)
        unknown(start + offset + 3) = unknown(start + offset + 3) - min_q3; % q3(i+1)
    end
end

```

```

        unknown(start + offset + 2) = unknown(start + offset + 2) + min_q2; % q2(i+1)
        unknown(start + offset + 4) = unknown(start + offset + 4) + min_q4; % q4(i+1)
    end
end
for kk = 1:4
    if unknown(start + offset + kk) < 0.1
        unknown(start + offset + kk) = 0;
    end
end
end % for k

```

```

function fill_back(indicator, option_entry)
% fill back data structure
global level_1 level_2 level_3 unknown
if indicator >= 1 % level 1
    option = option_entry(1); % option #
    level_1(option, 2:3) = unknown(1:2); % g1(i) and g2(i)
    level_1(option, 4:7) = unknown(7:10); % q1(i+1)~q4(i+1)
end
if indicator >= 2 % level 2
    option = option_entry(2);
    start = 19;
    level_2(option, 2:3) = unknown(start + 1:start + 2); % g1(i) and g2(i)
    level_2(option, 4:7) = unknown(start + 3:start + 6); % q1(i+1)~q4(i+1)
end
if indicator >= 3 % level 3
    option = option_entry(3);
    start = 19 + 11;
    level_3(option, 2:3) = unknown(start + 1:start + 2); % g1(i) and g2(i)
    level_3(option, 4:7) = unknown(start + 3:start + 6); % q1(i+1)~q4(i+1)
end
end

```

```

function pi = pi_calculate(indicator, queue_mean)
% calculate PI from level 1 to level indicator
global green r_matrix s_matrix unknown
pi = 0;
for k = 1:indicator
    if k == 1 % for level 1
        start = 0; % starting position of unknowns for this level
        offset = 6; % pointer at q(i+1)_1 away from the beginning
    else % for other levels
        start = 19 + 11 * (k - 2); % starting position of unknowns for this level
        offset = 2; % pointer at q(i+1)_1 away from the beginning
    end
    for kk = 1:4

```

```

        x(kk) = unknown(start + offset + kk) / queue_mean(kk); % q(i+1)_1 to _4
    end
    for kk = 1:2
        u(kk) = unknown(start + kk) / green(kk); % g1(i) and g2(i), offset same for all
    levels
    end
    pi = pi + 0.5 * (x * r_matrix * x' + u * s_matrix * u');
end

```

```

function initialize(eq_n)
% initialize a_matrix, b_vector, unknown and keep the first eq_n equations unchanged
global a_matrix b_vector unknown
[total_eq, total_unknown] = size(a_matrix);
temp_a(1:eq_n, 1:total_unknown) = a_matrix(1:eq_n, 1:total_unknown);
temp_b = b_vector(1:eq_n);
a_matrix = [];
b_vector = [];
unknown = [];
a_matrix = temp_a;
b_vector = temp_b;

```

```

function queue_initial = queue_get(queue_data, cycle_entry, instant_entry)
% get initial queue on each approach corresponding to input cycle_entry and
% instant_entry
pointer = (instant_entry - 1) * 6 + (cycle_entry - 5); % cycle_entry begins from 5 to 10
cycle_no = queue_data(pointer + 1, 1);
queue_initial = queue_data(pointer + 1, 2:5);

```

```

function min = min_choose(a, b)
if a <= b
    min = a;
else
    min = b;
end

```

Appendix H Computer Program for Coordinated Route Control

H.1 Normal operation

```
offset_1 = 40 % between intersection 1 and 2
offset_2 = 25 % between intersection 3 and 3
offset_3 = 30 % between intersection 3 and 4
length_link_1 = 500 % meter, between intersection 1 and 2
length_link_2 = 300 % meter, between intersection 2 and 3
length_link_3 = 400 % meter, between intersection 4 and 4

% control for intersection 1 eastbound
offset_in = 0
length_link_in = length_link_1
arrival_1 = dlmread('arrival_1e.dat')
% through arrivals at an intersection from eastbound approach to the route
arrival_2 = dlmread('arrival_1n.dat') % right turn to the route
arrival_3 = dlmread('arrival_1s.dat') % left turn to the route
turn_1 = dlmread('turn_1e.dat') % turn factor for through vehicles at an intersection
turn_2 = dlmread('turn_1n.dat') % right turn
turn_3 = dlmread('turn_1s.dat') % left turn
[queue_length,arrival_2e,turn_2e] = normal_intersection(offset_in, length_link_in,
arrival_1,turn_1, arrival_2,turn_2, arrival_3,turn_3)
% queue_length=queue length at intersection 1
% arrival_2e, turn_2e = vehicle arrivals at intersection 2 from eastbound approach
% and corresponding turn factors

% control for intersection 2 eastbound
offset_in = offset_1
length_link_in = length_link_2
arrival_1 = arrival_2e % through arrivals
arrival_2 = dlmread('arrival_2n.dat') % right turn to the route
arrival_3 = dlmread('arrival_2s.dat') % left turn to the route
turn_1 = turn_2e % turn factor for through vehicles
turn_2 = dlmread('turn_2n.dat') % right turn
turn_3 = dlmread('turn_2s.dat') % left turn
[queue_length,arrival_3e,turn_3e] = normal_intersection(offset_in, length_link_in,
arrival_1,turn_1, arrival_2,turn_2, arrival_3,turn_3)

% control for intersection 3 eastbound
offset_in = offset_1+offset_2
length_link_in = length_link_3
```



```

arrival_1 = arrival_3e % through arrivals
arrival_2 = dlmread('arrival_3n.dat') % right turn to the route
arrival_3 = dlmread('arrival_3s.dat') % left turn to the route
turn_1 = turn_3e % turn factor for through vehicles
turn_2 = dlmread('turn_3n.dat') % right turn
turn_3 = dlmread('turn_3s.dat') % left turn
[queue_length,arrival_4e,turn_4e] = normal_intersection(offset_in, length_link_in,
arrival_1,turn_1, arrival_2,turn_2, arrival_3,turn_3)

% control for intersection 4 eastbound
offset_in = offset_1+offset_2+offset_3
length_link_in = 0 % no more intersection after intersection 4
arrival_1 = arrival_4e % through arrivals
arrival_2 = dlmread('arrival_4n.dat') % right turn to the route
arrival_3 = dlmread('arrival_4s.dat') % left turn to the route
turn_1 = turn_4e % turn factor for through vehicles
turn_2 = dlmread('turn_4n.dat') % right turn
turn_3 = dlmread('turn_4s.dat') % left turn
[queue_length,arrival_5e,turn_5e] = normal_intersection(offset_in, length_link_in,
arrival_1,turn_1, arrival_2,turn_2, arrival_3,turn_3)

% control for intersection 4 westbound
offset_in = 0
length_link_in = length_link_3
arrival_1 = dlmread('arrival_4w.dat')
% through arrivals at an intersection from westbound approach to the route
arrival_2 = dlmread('arrival_4s.dat') % right turn to the route
arrival_3 = dlmread('arrival_4n.dat') % left turn to the route
turn_1 = dlmread('turn_4w.dat') % turn factor for through vehicles
turn_2 = dlmread('turn_4s.dat') % right turn
turn_3 = dlmread('turn_4n.dat') % left turn
[queue_length,arrival_3w,turn_3w] = normal_intersection(offset_in, length_link_in,
arrival_1,turn_1, arrival_2,turn_2, arrival_3,turn_3)
% queue_length=queue length at intersection 4
% arrival_3w, turn_3w = vehicle arrivals at intersection 3 from weststbound approach
% and corresponding turn factors

% control for intersection 3 westbound
offset_in = offset_3
length_link_in = length_link_2
arrival_1 = arrival_3w
% through arrivals at an intersection from westbound approach to the route
arrival_2 = dlmread('arrival_3s.dat') % right turn to the route
arrival_3 = dlmread('arrival_3n.dat') % left turn to the route
turn_1 = turn_3w % turn factor for through vehicles

```

```

turn_2 = dlmread('turn_3s.dat') % right turn
turn_3 = dlmread('turn_3n.dat') % left turn
[queue_length,arrival_2w,turn_2w] = normal_intersection(offset_in, length_link_in,
arrival_1,turn_1, arrival_2,turn_2, arrival_3,turn_3)

% control for intersection 2 westbound
offset_in = offset_2+offset_3
length_link_in = length_link_1
arrival_1 = arrival_2w
% through arrivals at an intersection from westbound approach to the route
arrival_2 = dlmread('arrival_2s.dat') % right turn to the route
arrival_3 = dlmread('arrival_2n.dat') % left turn to the route
turn_1 = turn_2w % turn factor for through vehicles
turn_2 = dlmread('turn_2s.dat') % right turn
turn_3 = dlmread('turn_2n.dat') % left turn
[queue_length,arrival_1w,turn_1w] = normal_intersection(offset_in, length_link_in,
arrival_1,turn_1, arrival_2,turn_2, arrival_3,turn_3)

% control for intersection 1 westbound
offset_in = offset_1+offset_2+offset_3
length_link_in = 0 % no more intersection after intersection 1
arrival_1 = arrival_1w
% through arrivals at an intersection from westbound approach to the route
arrival_2 = dlmread('arrival_1s.dat') % right turn to the route
arrival_3 = dlmread('arrival_1n.dat') % left turn to the route
turn_1 = turn_1w % turn factor for through vehicles
turn_2 = dlmread('turn_1s.dat') % right turn
turn_3 = dlmread('turn_1n.dat') % left turn
[queue_length,arrival_0e,turn_0e] = normal_intersection(offset_in, length_link_in,
arrival_1,turn_1, arrival_2,turn_2, arrival_3,turn_3)

% -----end of main program-----

function [queue_length,vehicle_remained,turn] =
normal_intersection(offset_in,length_link,arrival_1,turn_1, arrival_2,turn_2,
arrival_3,turn_3)
% generate queue length at an intersection and traffic profile on the next link for
% normal control scenario
% queue_length[1:3,cycle_number]= queue formed on approach 1 to 3
% vehicle_remained, turn= vehicles remained on the route after the intersection, which
% are arriving times at the next intersection, and corresponding turn factors
% offset_in= offset relative to the first intersection
% length_link= link length between this intersection and the next one
green = [60,40]
yellow = [3,3]

```

```

allred = [1,1]
cycle_number = 20
s_mean = 12 % m/s, general traffic mean speed
t_link = length_link / s_mean

turn_flag = 1 % through
offset = offset_in
[queue_length_1,vehicle_remained_1] =
normal_approach(turn_flag,arrival_1,turn_1,green,yellow,allred,cycle_number,offset)
turn_flag = 2 % right turn
offset = offset + 64 %green(1)+yellow(1)+allred(1)
green = [40,60]
[queue_length_2,vehicle_remained_2] =
normal_approach(turn_flag,arrival_2,turn_2,green,yellow,allred,cycle_number,offset)
turn_flag = 3 % left turn
offset = offset_1 + 64
green = [40,60]
[queue_length_3,vehicle_remained_3] =
normal_approach(turn_flag,arrival_3,turn_3,green,yellow,allred,cycle_number,offset)

% form queue length at the intersection
queue_length(1,1:cycle_number) = queue_length_1
queue_length(2,1:cycle_number) = queue_length_2
queue_length(3,1:cycle_number) = queue_length_3
queue_average = mean(queue_length') % averages of columns
standard_deviation = std(queue_length') % std of columns
queue_max = queue_average + 2*standard_deviation
% used for determination if EV-detector distances

% form arrivals at the next intersection
temp = vehicle_remained_2
vehicle_total_2 = length(temp)
temp_vehicle_total_2 = length(temp)
vehicle_total_3 = length(vehicle_remained_3)
arrival_total = 0
pointer = 0
if vehicle_total_3 == 0
    arrival_to_next_intersection = temp
end
for k_3=1:vehicle_total_3
    if length(temp) == 0
        arrival_total = arrival_total + 1
        arrival_to_next_intersection(arrival_total) = vehicle_remained_3(k_3)
        continue
    end
end

```

```

for k_2=1:temp_vehicle_total_2
    if temp(k_2) <= vehicle_remained_3(k_3)
        % vehicle from app.2 earlier than vehicle form app.3
        pointer = pointer + 1
        arrival_total = arrival_total + 1
        arrival_to_next_intersection(arrival_total) = temp(k_2)
        continue % for k_2
    else % vehicle from app.3 earlier than vehicle form app.2
        arrival_total = arrival_total + 1
        arrival_to_next_intersection(arrival_total) = vehicle_remained_3(k_3)
        if k_3 == vehicle_total_3
            pointer = pointer + 1
            arrival_total = arrival_total + 1
            arrival_to_next_intersection(arrival_total) = temp(k_2)
            continue
        end
        break % for k_2
    end % if
end % k_2
temp_vehicle_total_2 = vehicle_total_2 - pointer
temp = []
temp(1:vehicle_total_2-pointer) = vehicle_remained_2(pointer+1:vehicle_total_2)
end % k_3

```

```

temp_1 = vehicle_remained_1
vehicle_total_1 = length(temp_1)
temp_vehicle_total_1 = length(temp_1)
temp_2 = arrival_to_next_intersection
arrival_to_next_intersection = []
vehicle_total_2 = length(temp_2)
arrival_total = 0
pointer = 0
if vehicle_total_2 == 0
    arrival_to_next_intersection = temp_1
end
for k_2=1:vehicle_total_2
    if length(temp_1) == 0
        arrival_total = arrival_total + 1
        arrival_to_next_intersection(arrival_total) = temp_2(k_2)
        continue
    end
    for k_1=1:temp_vehicle_total_1
        if temp_1(k_1) <= temp_2(k_2)
            % vehicle from app.1 earlier than vehicle form app.2
            pointer = pointer + 1

```

```

    arrival_total = arrival_total + 1
    arrival_to_next_intersection(arrival_total) = temp_1(k_1)
    continue % for k_1
else % vehicle from app.2/3 earlier than vehicle from app.1
    arrival_total = arrival_total + 1
    arrival_to_next_intersection(arrival_total) = temp_2(k_2)
    if k_2 == vehicle_total_2
        pointer = pointer + 1
        arrival_total = arrival_total + 1
        arrival_to_next_intersection(arrival_total) = temp_1(k_1)
        continue
    end
    break % for k_1
end % if
end % k_1
temp_vehicle_total_1 = vehicle_total_1 - pointer
temp_1 = []
temp_1(1:vehicle_total_1-pointer) = vehicle_remained_1(pointer+1:vehicle_total_1)
end % k_2

```

```

vehicle_remained = arrival_to_next_intersection + t_link
% instant of time each vehicle arrives at the next intersection

```

```

% produce turn distribution
lt_rate = 4
rt_rate = 6
tva = length(vehicle_remained)
turn = ones(1,tva); % factor for straight-through vehicles
rnn = round(100 * rand(1,tva));
left_turn = find(rnn <= lt_rate)
% indices of arrival where the vehicle is left_turning
left_turn_total = length(left_turn)
turn(left_turn) = 1.4; % factor for left-turn vehicles
right_turn = find((rnn > lt_rate) & (rnn <= lt_rate + rt_rate))
% indices of arrival where the vehicle is right_turning
right_turn_total = length(right_turn)
turn(right_turn) = 1.2; % factor for right-turn vehicles
% save vehicle arrival and turn as delimited ASCII files using:
save arrival_5e.dat vehicle_remained -ascii
save turn_5e.dat turn -ascii

```

```

%-----
function [queue_length,vehicle_remained] =
normal_approach(turn_flag,arrival,turn,green,yellow,allred,cycle_number,offset)
% generate queue length on an approach and traffic remained on the next link

```

```

% queue_length[1,cycle_number]= queue formed on the approach
% vehicle_remained= vehicles remained on the route after the intersection
% turn_flag= 1 for through, 2 for right turn, and 3 for left turn
% arrival= arrival time for each vehicle relative to the beginning of 0 sec
% turn= turning factor corresponding to each arrival with straight-through (1.0),
% left-turn (1.4), or right-turn (1.2)
% offset= beginning of green of an approach relative to the through approach of the
        first intersection

cycle_length = green(1) + yellow(1) + allred(1) + green(2) + yellow(2) + allred(2)
saturation = 2000 / 3600 % in veh/sec
discharge_rate_green = 1 / saturation
% for straight-through vehicles during green in sec/veh
discharge_rate_yellow = 2 * discharge_rate_green
% for straight-through vehicles during yellow in sec/veh
yellow_equival_green = yellow(1) * discharge_rate_green / discharge_rate_yellow
% transfer yellow interval to equivalent green discharging vehicles
queue_length = zeros(1,cycle_number)
queue_last_cycle = 0
remained = 0 % # of vehicles remained on the route
vehicle_remained = []

total_produced = zeros(1,cycle_number)
total_discharged = zeros(1,cycle_number)

pointer = 1
t = arrival(pointer) % arrival time in sec, and the possibly earliest one arrives at 0 sec
queue_start=pointer

for cycle = 1:cycle_number
    if t >= cycle * cycle_length % wait to clear in next cycle
        continue
    end
    queue_green = queue_last_cycle
    % queue_yellow = 0
    queue_red = 0
    discharged_green = 0
    % discharged_yellow = 0
    green_time_available = green(1) + yellow_equival_green
    % yellow_time_available = yellow(1)

    if queue_green > 0 % processing queue from last cycle
        for k = 1:queue_green
            green_time_available = green_time_available - turn(queue_start) *
                discharge_rate_green
        end
    end
end

```

```

if green_time_available < 0
    break
end
discharged_green = k
if ( (turn_flag == 1) & (abs(turn(queue_start+k-1) - 1) < 0.1) ) |
    ( (turn_flag == 2 & abs(turn(queue_start+k-1) - 1.2) < 0.1) ) |
    ( (turn_flag == 3 & abs(turn(queue_start+k-1) - 1.4) < 0.1) )
    % a through vehicle or right turn or left turn
    remained = remained + 1
    vehicle_remaind(remained) = (cycle - 1) * cycle_length + green(1) -
    green_time_available
    % instant of time the vehicle just passed the intersection and on the next link
    % relative to the beginning of green for this approach
    vehicle_remaind(remained) = vehicle_remaind(remained) + offset
    % relative to the beginning of green for the first through approach
end
end % for
total_discharged(cycle) = total_discharged(cycle) + discharged_green
queue_start = queue_start + discharged_green
queue_green = queue_green - discharged_green
end % if queue_green > 0

while t < cycle * cycle_length % processing arrivals during this cycle
    total_produced(cycle) = total_produced(cycle) + 1
    t_quotient = fix(t / cycle_length)
    t_elapsed = t - cycle_length * t_quotient
    % time elapsed from the beginning of green interval
    if (t_elapsed >= 0) && (t_elapsed < green(1) + yellow_equival_green)
        indicator = 'green or first half yellow'
        if queue_green > 0 % there already exists queue that cannot be discharged
            % during this green period
            queue_green = queue_green + 1
        elseif queue_green == 0 % deal with the situation that no queue exists
            if (green(1) - t_elapsed) < green_time_available
                green_time_available = green(1) - t_elapsed
            end
            green_time_available = green_time_available - turn(queue_start) *
            discharge_rate_green
            if green_time_available < 0 % queue produced
                queue_green = queue_green + 1
            elseif green_time_available >= 0
                discharged_green = discharged_green + 1
                if ( (turn_flag == 1) & (abs(turn(queue_start) - 1) < 0.1) ) |
                    ( (turn_flag == 2 & abs(turn(queue_start) - 1.2) < 0.1) ) |
                    ( (turn_flag == 3 & abs(turn(queue_start) - 1.4) < 0.1) )

```

```

    % a through vehicle or right turn or left turn
    remained = remained + 1
    vehicle_remained(remained) = (cycle - 1) * cycle_length + green(1) -
    green_time_available
    % instant of time the vehicle just passed the intersection and
    % on the next link relative to the beginning of green for this approach
    vehicle_remained(remained) = vehicle_remained(remained) + offset
    % relative to the beginning of green for the first through approach
end
total_discharged(cycle) = total_discharged(cycle) + 1
queue_start = queue_start + 1
end
end % if queue_green > 0
elseif (t_elapsed >= green(1) + yellow_equival_green)
    indicator = 'second half yellow or red'
    queue_red = queue_red + 1
end % indicator is 'green' or 'red'
if pointer >= length(arrival)
    break
end
pointer = pointer + 1
t = arrival(pointer)
end % while
undischarged(cycle) = queue_green % undischarged during this green interval
queue_length(cycle) = queue_green + queue_red
queue_last_cycle = queue_length(cycle)
end % for

%-----
function vehicle_combined = combination(array_1, array_2)
% combine array_1 and array_2 into vehicle_combined based on arrival time: from
% earliest to latest
% It is better length(array_1)>length(array_2)
temp_1 = array_1
vehicle_total_1 = length(temp_1)
temp_vehicle_total_1 = length(temp_1)
temp_2 = arrival_2
arrival_to_next_intersection = []
vehicle_total_2 = length(temp_2)
arrival_total = 0
pointer = 0
if vehicle_total_2 == 0
    arrival_to_next_intersection = temp_1
end
for k_2=1:vehicle_total_2

```



```

if length(temp_1) == 0
    arrival_total = arrival_total + 1
    arrival_to_next_intersection(arrival_total) = temp_2(k_2)
    continue
end
for k_1=1:temp_vehicle_total_1
    if temp_1(k_1) <= temp_2(k_2) % vehicle from app.1 earlier than vehicle from
    app.2
        pointer = pointer + 1
        arrival_total = arrival_total + 1
        arrival_to_next_intersection(arrival_total) = temp_1(k_1)
        continue % for k_1
    else % vehicle from app.2/3 earlier than vehicle from app.1
        arrival_total = arrival_total + 1
        arrival_to_next_intersection(arrival_total) = temp_2(k_2)
        if k_2 == vehicle_total_2
            pointer = pointer + 1
            arrival_total = arrival_total + 1
            arrival_to_next_intersection(arrival_total) = temp_1(k_1)
            continue
        end
        break % for k_1
    end % if
end % k_1
temp_vehicle_total_1 = vehicle_total_1 - pointer
temp_1 = []
temp_1(1:vehicle_total_1-pointer) = vehicle_remained_1(pointer+1:vehicle_total_1)
end % k_2
vehicle_combined = arrival_to_next_intersection
% instant of time each vehicle just passed the intersection
vehicle_combined = arrival_to_next_intersection + t_link
% instant of time each vehicle arrives at the next intersection

```

H.2 Fixed held time transitioning for transition 1

```

t_fixed = 50
for instant_entry=1:4
    for cycle_entry=5:10
        fixed_time_route(t_fixed,cycle_entry,instant_entry);
    end
end

function fixed_time_route(t_fixed,cycle_entry,instant_entry);
% coordinated route control
offset_1 = 40 % between intersection 1 and 2
offset_2 = 25 % between intersection 3 and 3
offset_3 = 30 % between intersection 3 and 4

% control for intersection 1
offset = 0
arrival_1 = dlmread('arrival_1e.dat')
% through arrivals at an intersection from eastbound approach to the route
arrival_2 = dlmread('arrival_1n.dat') % right turn to the route
arrival_3 = dlmread('arrival_1s.dat') % left turn to the route
arrival_4 = dlmread('arrival_1w.dat') % through vehicles from westbound approach
turn_1 = dlmread('turn_1e.dat') % turn factor for eastbound through vehicles
turn_2 = dlmread('turn_1n.dat') % right turn
turn_3 = dlmread('turn_1s.dat') % left turn
turn_4 = dlmread('turn_1w.dat') % through vehicles from westbound approach
[queue_length,turn]=fixed_time_intersection(offset,t_fixed,cycle_entry,instant_entry,
arrival_1, turn_1, arrival_2, turn_2, arrival_3, turn_3, arrival_4, turn_4);

% control for intersection 2
offset = offset_1
arrival_1 = dlmread('arrival_2e.dat')
% through arrivals at an intersection from eastbound approach to the route
arrival_2 = dlmread('arrival_2n.dat') % right turn to the route
arrival_3 = dlmread('arrival_2s.dat') % left turn to the route
arrival_4 = dlmread('arrival_2w.dat') % through vehicles from westbound approach
turn_1 = dlmread('turn_2e.dat') % turn factor for eastbound through vehicles
turn_2 = dlmread('turn_2n.dat') % right turn
turn_3 = dlmread('turn_2s.dat') % left turn
turn_4 = dlmread('turn_2w.dat') % through vehicles from westbound approach
[queue_length,turn]=fixed_time_intersection(offset,t_fixed,cycle_entry,instant_entry,
arrival_1, turn_1, arrival_2, turn_2, arrival_3, turn_3, arrival_4, turn_4);

```

```

% control for intersection 3
offset = offset_1 + offset_2
arrival_1 = dlmread('arrival_3e.dat')
% through arrivals at an intersection from eastbound approach to the route
arrival_2 = dlmread('arrival_3n.dat') % right turn to the route
arrival_3 = dlmread('arrival_3s.dat') % left turn to the route
arrival_4 = dlmread('arrival_3w.dat') % through vehicles from westbound approach
turn_1 = dlmread('turn_3e.dat') % turn factor for eastbound through vehicles
turn_2 = dlmread('turn_3n.dat') % right turn
turn_3 = dlmread('turn_3s.dat') % left turn
turn_4 = dlmread('turn_3w.dat') % through vehicles from westbound approach
[queue_length,turn]=fixed_time_intersection(offset,t_fixed,cycle_entry,instant_entry,
arrival_1, turn_1, arrival_2, turn_2, arrival_3, turn_3, arrival_4, turn_4);

% control for intersection 4
offset = offset_1 + offset_2 + offset_3
arrival_1 = dlmread('arrival_4e.dat')
% through arrivals at an intersection from eastbound approach to the route
arrival_2 = dlmread('arrival_4n.dat') % right turn to the route
arrival_3 = dlmread('arrival_4s.dat') % left turn to the route
arrival_4 = dlmread('arrival_4w.dat') % through vehicles from westbound approach
turn_1 = dlmread('turn_4e.dat') % turn factor for eastbound through vehicles
turn_2 = dlmread('turn_4n.dat') % right turn
turn_3 = dlmread('turn_4s.dat') % left turn
turn_4 = dlmread('turn_4w.dat') % through vehicles from westbound approach
[queue_length,turn]=fixed_time_intersection(offset,t_fixed,cycle_entry,instant_entry,
arrival_1, turn_1, arrival_2, turn_2, arrival_3, turn_3, arrival_4, turn_4);

%-----
function [queue_length,turn]=fixed_time_intersection(offset,t_fixed,cycle_entry,
instant_entry,arrival_1, turn_1, arrival_2, turn_2, arrival_3, turn_3, arrival_4, turn_4);
% generate queue length at an intersection under fixed time control strategy
% queue_length[1:4,1:cycle_entry]= queue formed on approach 1 to 4
% turn= turning factor
% offset=offset relative to the beginning of green for the 1st intersection
% t_fixed= fixed held time for EV operation
% cycle_entry= cycle # EV enters the system
% instant_entry= 1 for EV enters at g1=0, 2 for g1/2, 3 for g1, and 4 for g2/2
% arrival_# and turn_#=vehicle arrivals on approach # and turn factors

green = [60,40]
yellow = [3,3]
allred = [1,1]
on_route = 1 % serving EV approach
turn_flag = 1 % through

```

```

offset_1 = offset
[queue_out_1, turn_out_1, undischarged_1, time_spare_out_1] =
fixed_time_approach(on_route, turn_flag, offset_1, arrival_1, turn_1, cycle_entry,
instant_entry, t_fixed, green, yellow, allred)

on_route = 2
turn_flag = 2 % right turn
offset_2 = offset_1 + 64 %green(1)+yellow(1)+allred(1)
green = [40,60]
[queue_out_2, turn_out_2, undischarged_2, time_spare_out_2] =
fixed_time_approach(on_route, turn_flag, offset_2, arrival_2, turn_2, cycle_entry,
instant_entry, t_fixed, green, yellow, allred)

on_route = 2
turn_flag = 3 % left turn
offset_3 = offset_1 + 64
green = [40,60]
[queue_out_3, turn_out_3, undischarged_3, time_spare_out_3] =
fixed_time_approach(on_route, turn_flag, offset_3, arrival_3, turn_3, cycle_entry,
instant_entry, t_fixed, green, yellow, allred)

on_route = 3 % opposing approach
turn_flag = 1 % through
offset_4 = offset_1
green = [60,40]
[queue_out_4, turn_out_4, undischarged_4, time_spare_out_4] =
fixed_time_approach(on_route, turn_flag, offset_4, arrival_4, turn_4, cycle_entry,
instant_entry, t_fixed, green, yellow, allred)

% calculate average spare time, and only time_spare_out_1 is useful
if (cycle_entry == 5) && (instant_entry == 1)
    fid = fopen('time_spare.dat','w') % 1 for standard, 2 for error
else
    fid = fopen('time_spare.dat','a') % append data to the end of the file
end
fprintf(fid,'%6.2f\n',time_spare_out_1)
status = fclose(fid)
if status == -1
    disp(message)
    return
end
if (cycle_entry == 10) && (instant_entry == 4)
    fid = fopen('time_spare.dat','r') % 1 for standard, 2 for error
    if fid == -1 % no such file
        disp(message)
    end
end

```

```

        return
    end
    time_spare_vector = []
    time_spare_vector = fscanf(fid,'%g')
    % app. 1 for one row, totally 24 columns(4 instant_entry x6cycle)
    status =fclose(fid)
    if status == -1 % =0 if successful closed
        disp('error')
        return
    end
    time_spare_average = mean(time_spare_vector) % averages of columns
    standard_deviation = std(time_spare_vector) % std of columns
end

% form queue length at the intersection
queue_length = []
queue_length(1) = length(queue_out_1)
queue_length(2) = length(queue_out_2)
queue_length(3) = length(queue_out_3)
queue_length(4) = length(queue_out_4)
% queue_average = mean(queue_length) % averages of vector
% standard_deviation = std(queue_length) % std of vector
turn = []
turn(1,1:length(turn_out_1)) = turn_out_1
turn(2,1:length(turn_out_2)) = turn_out_2
turn(3,1:length(turn_out_3)) = turn_out_3
turn(4,1:length(turn_out_4)) = turn_out_4

if (cycle_entry == 5) && (instant_entry == 1)
    fid = fopen('queue_fixed.dat','w') % 1 for standard, 2 for error
else
    fid = fopen('queue_fixed.dat','a') % append data to the end of the file
end
fprintf(fid,'%6.2f%6.2f%6.2f%6.2f\n',queue_length)
status = fclose(fid)
if status == -1
    disp(message)
    return
end
if (cycle_entry == 10) && (instant_entry == 4)
    fid = fopen('queue_fixed.dat','r') % 1 for standard, 2 for error
    if fid == -1 % no such file
        disp(message)
        return
    end
end

```

```

        queue_length = []
        queue_length = fscanf(fid,'%g %g %g %g ', [4 inf])
        % app. 1,2, 3 and 4 for each row, totally 24 columns(4 instant_entry x6cycle)
        status = fclose(fid)
        if status == -1 % =0 if successful closed
            disp('error')
            return
        end
        queue_average = mean(queue_length') % averages of columns
        standard_deviation = std(queue_length') % std of columns
    end

%-----
function [queue_out, turn_out, queue_undischarged, time_spare_out] =
fixed_time_approach(on_route, turn_flag, offset, arrival, turn, cycle_entry, instant_entry,
t_fixed, green, yellow, allred)
% generate queue length on an approach under fixed held time control strategy
% queue_out, turn_out[1:?]= queue formed on the approach and turning factors
% queue_undischarged= number of queue ahead of EV undischarged when EV reaches
% the intersection, applicable only if on_route==1
% on_route= 1 for approach serving EV; 2 for crossing road, and 3 for opposing
% approach to serving EV one
% turn_flag= 1 for through, 2 for right turn, and 3 for left turn
% arrival= arrival time for each vehicle relative to the beginning ( 0 sec )
% turn= turning factor corresponding to each arrival with straight-through (1.0),
% left-turn (1.4), or right-turn (1.2)
% offset= beginning of green of an approach relative to green for the through approach
% of the first intersection
% t_fixed= fixed held time for EV operation

cycle_length = green(1) + yellow(1) + allred(1) + green(2) + yellow(2) + allred(2)
saturation = 2000 / 3600 % in veh/sec
discharge_rate_green = 1 / saturation
% for straight-through vehicles during green in sec/veh
discharge_rate_yellow = 2 * discharge_rate_green
% for straight-through vehicles during yellow in sec/veh
yellow_equival_green = yellow(1) * discharge_rate_green / discharge_rate_yellow
% transfer yellow interval to equivalent green discharging vehicles

% instant_entry: 1 for g1=0, 2 for g1/2, 3 for g1, or 4 for g1+y1+allred1+g2/2
if (on_route == 1) || (on_route == 3)
    switch instant_entry
        case 1
            entry_point = 0

```

```

        % time point away from the beginning of green at which EV enters
    case 2
        entry_point = green(1) / 2
    case 3
        entry_point = green(1)
    case 4
        entry_point = green(1) + yellow(1) + allred(1) + green(2)/2
        % only this case in red period
    end % switch
elseif (on_route == 2) || (on_route == 4)
    switch instant_entry
    case 1
        entry_point = green(1) + yellow(1) + allred(1) % in red period
    case 2
        entry_point = green(1) + yellow(1) + allred(1) + green(2) / 2
    case 3
        entry_point = green(1) + yellow(1) + allred(1) + green(2)
    case 4
        entry_point = green(1) / 2 % only this case in green period
    end % switch
end

queue_length = zeros(1,cycle_entry)
total_produced = zeros(1,cycle_entry)
total_discharged = zeros(1,cycle_entry)
queue_last_cycle = 0
pointer = 1
t = arrival(pointer) % arrival time in sec, and the possibly earliest one arrives at 0 sec
if (on_route == 1) || (on_route == 3)
    t = t - offset
end
queue_start=pointer
for cycle = 1:cycle_entry
    if t >= cycle * cycle_length % wait to clear in next cycle
        continue
    end
    if (cycle==cycle_entry) && (instant_entry == 1)
        if (on_route == 1) || (on_route == 3)
            % EV is coming at the beginning of green and no need to calculate other cycles
            break
        end
    end
end
queue_green = queue_last_cycle
queue_red = 0
discharged_green = 0

```

```

green_time_available = green(1) + yellow_equival_green

if queue_green > 0 % processing queue from last cycle
    for k = 1:queue_green
        green_time_available = green_time_available - turn(queue_start) *
            discharge_rate_green
        if green_time_available < 0
            break
        end
        discharged_green = k
    end % for
    total_discharged(cycle) = total_discharged(cycle) + discharged_green
    queue_start = queue_start + discharged_green
    queue_green = queue_green - discharged_green
end % if queue_green > 0

while t < cycle * cycle_length % processing arrivals during this cycle
    total_produced(cycle) = total_produced(cycle) + 1
    t_quotient = fix(t / cycle_length)
    t_elapsed = t - cycle_length * t_quotient
    % time elapsed from the beginning of green interval
    if (cycle == cycle_entry) && (t_elapsed >= entry_point)
        % EV is coming and no need to calculate other cycles
        break
    end
    if (t_elapsed >= 0) && (t_elapsed < green(1) + yellow_equival_green)
        indicator = 'green or first half yellow'
        if queue_green > 0 % there already exists queue that cannot be discharged
            % during this green period
            queue_green = queue_green + 1
        elseif queue_green == 0 % deal with the situation that no queue exists
            if (green(1) - t_elapsed) < green_time_available
                green_time_available = green(1) - t_elapsed
            end
            green_time_available = green_time_available - turn(queue_start) *
                discharge_rate_green
            if green_time_available < 0 % queue produced
                queue_green = queue_green + 1
            elseif green_time_available >= 0
                discharged_green = discharged_green + 1
                total_discharged(cycle) = total_discharged(cycle) + 1
                queue_start = queue_start + 1
            end
        end % if queue_green > 0
        elseif (t_elapsed >= green(1) + yellow_equival_green)

```



```

        indicator = 'second half yellow or red'
        queue_red = queue_red + 1
    end % indicator is 'green' or 'red'
    if pointer >= length(arrival)
        break
    end
    pointer = pointer + 1
    t = arrival(pointer)
    if (on_route == 1) || (on_route == 3)
        t = t - offset
    end
end % while
undischarged(cycle) = queue_green % undischarged during this green interval
queue_length(cycle) = queue_green + queue_red
queue_last_cycle = queue_length(cycle)
if cycle == cycle_entry % EV is coming and no need to calculate other cycles
    break
end
end % for

queue = queue_last_cycle % queue from normal operation
queue_undischarged = 0
time_spare_out = 0

if on_route == 2 % crossing road
    red_time_imposed = t_fixed
    start = (cycle_entry - 1) * cycle_length + entry_point
    while (t <= start + red_time_imposed)
        % produce vehicles during held time, t_fixed
        queue = queue + 1
        if pointer >= length(arrival)
            break
        end
        pointer = pointer + 1
        t = arrival(pointer)
    end % while
    queue_length(cycle_entry) = queue % produced during t_fixed
    queue_out = arrival(queue_start:pointer-1)
    turn_out = turn(queue_start:pointer-1)
elseif on_route == 1 % serving EV approach
    discharged_green = 0
    green_time_available = t_fixed
    if queue > 0 % discharge queue from normal operation
        total_green_time_needed = 0
        for k = 1:queue

```

```

        total_green_time_needed = total_green_time_needed +
            turn(queue_start + k - 1) * discharge_rate_green
    end % for
    for k = 1:queue
        green_time_available = green_time_available -
            turn(queue_start + k - 1) * discharge_rate_green
        if green_time_available < 0
            break
        end
        discharged_green = k
    end % for
    discharged_for_ev = discharged_green
    queue_start = queue_start + discharged_green
    queue = queue - discharged_green
end % if queue > 0

if (queue > 0)
    successful = 'No!' % more time is needed
    time_needed = total_green_time_needed - t_fixed
    queue_undischarged = queue
    queue_out = queue_undischarged + 0.1 % different from successful == 'yes'
    turn_out = turn(queue_start:queue_start+queue-1)
elseif (queue == 0)
    successful = 'Yes!'
    time_spare = green_time_available
    time_spare_out = time_spare
    start = (cycle_entry - 1) * cycle_length + entry_point
    while (t <= start + t_fixed) % produce vehicles during held time, t_fixed
        queue = queue + 1
        if pointer >= length(arrival)
            break
        end
        pointer = pointer + 1
        t = arrival(pointer) - offset
    end % while
    queue_length(cycle_entry) = queue % produced during t_fixed
    queue_out = arrival(queue_start:pointer-1) % produced during t_fixed = queue
    turn_out = turn(queue_start:pointer-1)
end %if

elseif on_route == 3 % opposing approach to serving EV one
    discharged_green = 0
    green_time_available = t_fixed
    if queue > 0 % processing queue from normal operation
        total_green_time_needed = 0
        for k = 1:queue

```

```

    total_green_time_needed = total_green_time_needed +
    turn(queue_start + k - 1) * discharge_rate_green
end % for
for k = 1:queue
    green_time_available = green_time_available -
    turn(queue_start + k - 1) * discharge_rate_green
    if green_time_available < 0
        break
    end
    discharged_green = k
end % for
discharged_for_ev = discharged_green
queue_start = queue_start + discharged_green
queue = queue - discharged_green
end % if queue > 0

start = (cycle_entry - 1) * cycle_length + entry_point
while (t <= start + t_fixed) % produce vehicles during held time.t_fixed
    if queue > 0 % there already exists queue that cannot be discharged
        % during this green period
        queue = queue + 1
    elseif queue == 0 % deal with the situation that no queue exists
        if (start + t_fixed - t) < green_time_available
            green_time_available = start + t_fixed - t
        end
        green_time_available = green_time_available -
        turn(queue_start) * discharge_rate_green
        if green_time_available < 0 % queue produced
            queue = queue + 1
        elseif green_time_available >= 0
            discharged_green = discharged_green + 1
            queue_start = queue_start + 1
        end
    end % if queue > 0
    if pointer >= length(arrival)
        break
    end
    pointer = pointer + 1
    t = arrival(pointer) - offset
end % while
queue_length(cycle_entry) = queue % produced at the end of t_fixed period
queue_out = arrival(queue_start:pointer-1)
turn_out = turn(queue_start:pointer-1)
end % if on_route

```

H.3 Real-time control for transition 1

```

for instant_entry=1:4
    for cycle_entry=5:10
        [queue_length,turn]=real_time_route(cycle_entry,instant_entry);
    end
end

```

```

function [queue_length,turn]=real_time_route(cycle_entry,instant_entry);
% real-time control for coordinated route
offset_1 = 40 % between intersection 1 and 2
offset_2 = 25 % between intersection 3 and 3
offset_3 = 30 % between intersection 3 and 4
% Using detector distance L=450/600/770/785 m at intersection 1/2/3/4

% control for intersection 1
offset = 0
detector_distance = 450
arrival_1 = dlmread('arrival_1e.dat')
% through arrivals at an intersection from eastbound approach to the route
arrival_2 = dlmread('arrival_1n.dat') % right turn to the route
arrival_3 = dlmread('arrival_1s.dat') % left turn to the route
arrival_4 = dlmread('arrival_1w.dat') % through vehicles from westbound approach
turn_1 = dlmread('turn_1e.dat') % turn factor for eastbound through vehicles
turn_2 = dlmread('turn_1n.dat') % right turn
turn_3 = dlmread('turn_1s.dat') % left turn
turn_4 = dlmread('turn_1w.dat') % through vehicles from westbound approach
[queue_length,turn]=real_time_intersection(cycle_entry,instant_entry, offset,
detector_distance,arrival_1,turn_1,arrival_2,turn_2,arrival_3,turn_3,arrival_4,turn_4);

% control for intersection 2
offset = offset_1
detector_distance = 600
arrival_1 = dlmread('arrival_2e.dat')
% through arrivals at an intersection from eastbound approach to the route
arrival_2 = dlmread('arrival_2n.dat') % right turn to the route
arrival_3 = dlmread('arrival_2s.dat') % left turn to the route
arrival_4 = dlmread('arrival_2w.dat') % through vehicles from westbound approach
turn_1 = dlmread('turn_2e.dat') % turn factor for eastbound through vehicles
turn_2 = dlmread('turn_2n.dat') % right turn
turn_3 = dlmread('turn_2s.dat') % left turn
turn_4 = dlmread('turn_2w.dat') % through vehicles from westbound approach

```

```
[queue_length,turn]=real_time_intersection(cycle_entry,instant_entry, offset,
detector_distance,arrival_1,turn_1,arrival_2,turn_2,arrival_3,turn_3,arrival_4,turn_4);
```

```
% control for intersection 3
offset = offset_1+offset_2
detector_distance = 770
arrival_1 = dlmread('arrival_3e.dat')
% through arrivals at an intersection from eastbound approach to the route
arrival_2 = dlmread('arrival_3n.dat') % right turn to the route
arrival_3 = dlmread('arrival_3s.dat') % left turn to the route
arrival_4 = dlmread('arrival_3w.dat') % through vehicles from westbound approach
turn_1 = dlmread('turn_3e.dat') % turn factor for eastbound through vehicles
turn_2 = dlmread('turn_3n.dat') % right turn
turn_3 = dlmread('turn_3s.dat') % left turn
turn_4 = dlmread('turn_3w.dat') % through vehicles from westbound approach
[queue_length,turn]=real_time_intersection(cycle_entry,instant_entry, offset,
detector_distance,arrival_1,turn_1,arrival_2,turn_2,arrival_3,turn_3,arrival_4,turn_4);
```

```
% control for intersection 4
offset = offset_1+ offset_2+ offset_3
detector_distance = 785
arrival_1 = dlmread('arrival_4e.dat')
% through arrivals at an intersection from eastbound approach to the route
arrival_2 = dlmread('arrival_4n.dat') % right turn to the route
arrival_3 = dlmread('arrival_4s.dat') % left turn to the route
arrival_4 = dlmread('arrival_4w.dat') % through vehicles from westbound approach
turn_1 = dlmread('turn_4e.dat') % turn factor for eastbound through vehicles
turn_2 = dlmread('turn_4n.dat') % right turn
turn_3 = dlmread('turn_4s.dat') % left turn
turn_4 = dlmread('turn_4w.dat') % through vehicles from westbound approach
[queue_length,turn]=real_time_intersection(cycle_entry,instant_entry, offset,
detector_distance,arrival_1,turn_1,arrival_2,turn_2,arrival_3,turn_3,arrival_4,turn_4);
```

```
%-----
function [queue_length,turn]=real_time_intersection(cycle_entry,instant_entry,offset,
detector_distance,arrival_1,turn_1,arrival_2,turn_2,arrival_3,turn_3,arrival_4,turn_4);
% generate queue length at an intersection under real time control strategy
% queue_length[1:4,1:cycle_entry]– queue formed on approach 1 to 4
% turn= turning factor
% cycle_entry= cycle # EV enters the system
% instant_entry= 1 for EV enters at g1=0, 2 for g1/2, 3 for g1, and 4 for g2/2
% offset=offset relative to the beginning of green for the 1st intersection
% detector_distance = detector distance away from the intersection
% arrival_# and turn_#=vehicle arrivals on approach # and turn factors
```

```

green = [60,40]
yellow = [3,3]
allred = [1,1]
on_route = 1 % serving EV approach
turn_flag = 1 % through
offset_1 = offset
green_side_in = 0
[queue_out_1, turn_out_1, undischarged_1, green_side_out_1, time_spare_out_1] =
real_time_approach(on_route, turn_flag, offset_1, arrival_1, turn_1, detector_distance,
green_side_in, cycle_entry, instant_entry, green, yellow, allred)

on_route = 2
turn_flag = 2 % right turn
offset_2 = offset_1 + 64 %green(1)+yellow(1)+allred(1)
green = [40,60]
green_side_in = green_side_out_1
[queue_out_2, turn_out_2, undischarged_2, green_side_out, time_spare_out_2] =
real_time_approach(on_route, turn_flag, offset_2, arrival_2, turn_2, detector_distance,
green_side_in, cycle_entry, instant_entry, green, yellow, allred)

on_route = 2
turn_flag = 3 % left turn
offset_3 = offset_1 + 64
green = [40,60]
[queue_out_3, turn_out_3, undischarged_3, green_side_out, time_spare_out_3] =
real_time_approach(on_route, turn_flag, offset_3, arrival_3, turn_3, detector_distance,
green_side_in, cycle_entry, instant_entry, green, yellow, allred)

on_route = 3 % opposing approach
turn_flag = 1 % through
offset_4 = offset_1
green = [60,40]
[queue_out_4, turn_out_4, undischarged_4, green_side_out, time_spare_out_4] =
real_time_approach(on_route, turn_flag, offset_4, arrival_4, turn_4, detector_distance,
green_side_in, cycle_entry, instant_entry, green, yellow, allred)

% calculate average green time available for side roads, and only green_side_out_1 is
% useful
if (cycle_entry == 5) && (instant_entry == 1)
    fid = fopen('green_side.dat','w') % 1 for standard, 2 for error
else
    fid = fopen('green_side.dat','a') % append data to the end of the file
end
fprintf(fid, '%6.2f\n', green_side_out_1)
status = fclose(fid)

```

```

if status == -1
    disp(message)
    return
end
if (cycle_entry == 10) && (instant_entry == 4)
    fid = fopen('green_side.dat','r') % 1 for standard, 2 for error
    if fid == -1 % no such file
        disp(message)
        return
    end
    green_side_vector = []
    green_side_vector = fscanf(fid,'%g')
    % app. 1 for one row, totally 24 columns(4 instant_entry x6cycle)
    status =fclose(fid)
    if status == -1 % =0 if successful closed
        disp('error')
        return
    end
    green_side_average = mean(green_side_vector) % averages of columns
    standard_deviation = std(green_side_vector) % std of columns
end

% calculate average spare time, and only time_spare_out_1 is useful
if (cycle_entry == 5) && (instant_entry == 1)
    fid = fopen('time_spare.dat','w') % 1 for standard, 2 for error
else
    fid = fopen('time_spare.dat','a') % append data to the end of the file
end
fprintf(fid,'%6.2f\n',time_spare_out_1)
status = fclose(fid)
if status == -1
    disp(message)
    return
end
if (cycle_entry == 10) && (instant_entry == 4)
    fid = fopen('time_spare.dat','r') % 1 for standard, 2 for error
    if fid == -1 % no such file
        disp(message)
        return
    end
    time_spare_vector = []
    time_spare_vector = fscanf(fid,'%g')
    % app. 1 for one row, totally 24 columns(4 instant_entry x6cycle)
    status =fclose(fid)
    if status == -1 % =0 if successful closed

```

```

        disp('error')
        return
    end
    time_spare_average = mean(time_spare_vector) % averages of columns
    standard_deviation = std(time_spare_vector) % std of columns
end

% form queue length at the intersection
queue_length = []
queue_length(1) = length(queue_out_1)
queue_length(2) = length(queue_out_2)
queue_length(3) = length(queue_out_3)
queue_length(4) = length(queue_out_4)
% queue_average = mean(queue_length) % averages of vector
% standard_deviation = std(queue_length) % std of vector
turn = []
turn(1,1:length(turn_out_1)) = turn_out_1
turn(2,1:length(turn_out_2)) = turn_out_2
turn(3,1:length(turn_out_3)) = turn_out_3
turn(4,1:length(turn_out_4)) = turn_out_4

if (cycle_entry == 5) && (instant_entry == 1)
    fid = fopen('queue_realtime.dat','w') % 1 for standard, 2 for error
else
    fid = fopen('queue_realtime.dat','a') % append data to the end of the file
end
fprintf(fid,'%6.2f %6.2f %6.2f %6.2f\n',queue_length)
status = fclose(fid)
if status == -1
    disp(message)
    return
end
if (cycle_entry == 10) && (instant_entry == 4)
    fid = fopen('queue_realtime.dat','r') % 1 for standard, 2 for error
    if fid == -1 % no such file
        disp(message)
        return
    end
    queue_length = []
    queue_length = fscanf(fid,'%g %g %g %g', [4 inf])
    % app. 1,2, 3 and 4 for each row, totally 24 columns(4 instant_entry x6cycle)
    status = fclose(fid)
    if status == -1 % =0 if successful closed
        disp('error')
        return
    end
end

```



```

end
    queue_average = mean(queue_length') % averages of columns
    standard_deviation = std(queue_length') % std of columns
end

%-----
function [queue_out, turn_out, queue_undischarged, green_side_out, time_spare_out] =
real_time_approach(on_route, turn_flag, offset, arrival, turn, detector_distance,
green_side_in, cycle_entry, instant_entry, green, yellow, allred)
% generate queue length on an approach under real time control strategy
% queue_out, turn_out[1:2]= queue formed on the approach and turning factors
% queue_undischarged= number of queue ahead of EV undischarged when EV reaches
% the intersection, applicable only if on_route==1
% green_side_out= green time available for side roads during EV operation
% on_route= 1 for approach serving EV; 2 for crossing road, and 3 for opposing
% approach to serving EV one
% turn_flag= 1 for through, 2 for right turn, and 3 for left turn
% arrival= arrival time for each vehicle relative to the beginning ( 0 sec)
% turn= turning factor corresponding to each arrival with straight-through (1.0),
% left-turn (1.4), or right-turn (1.2)
% offset= beginning of green of an approach relative to the through approach of the
% first intersection
% detector_distance= distance away from subject intersection (m)
% green_side_out= green available for side roads, obtained from serving EV approach
% green_side_in= green available for side roads, used for side roads and opposing
% serving EV app.

% For comparison with fixed time control, STI and startup delay will not be included in
% the calculation, i.e., STI=0 and t_startup=0
% EV operating speed SEV = speed_ev = 15 (m/s or 54 Km/h),
% General vehicle mean Smean = speed_mean = 12 (m/s or 43.2 Km/h)
% discharge speed Sdis = speed_dis = 4.44 (m/s or 16 Km/h)=2000*8/3600
% Average spacing between successive vehicles spacing = 8 metre
% Minimum greens for main road gmin(1) = 7 sec, and side road gmin(2) = 10 sec
% Maximum greens for main road gmax(1) = 100 sec, and side road gmax(2) = 100 sec

% successful='Yes!' if EV can cross the intersection, otherwise successful='No!'

cycle_length = green(1) + yellow(1) + allred(1) + green(2) + yellow(2) + allred(2)
saturation = 2000 / 3600 % in veh/sec
discharge_rate_green = 1 / saturation
% for straight-through vehicles during green in sec/veh
discharge_rate_yellow = 2 * discharge_rate_green
% for straight-through vehicles during yellow in sec/veh
yellow_equival_green = yellow(1) * discharge_rate_green / discharge_rate_yellow

```

```

% transfer yellow interval to equivalent green discharging vehicles

ll = detector_distance % detector distance
speed_ev = 15
t_ev = ll / speed_ev % take EV to traverse ll
sti = 0
t_startup = 0
t_ev = t_ev + sti + t_startup
speed_mean = 12 %
spacing = 8
speed_dis = 6
gmin = [7,10]
gmax = [100,100]

% instant_entry: 1 for g1=0, 2 for g1/2, 3 for g1, or 4 for g1+y1+allred1+g2/2
if (on_route == 1) || (on_route == 3)
    switch instant_entry
        case 1
            entry_point = 0
            % time point away from the beginning of green at which EV enters
        case 2
            entry_point = green(1) / 2
        case 3
            entry_point = green(1)
        case 4
            entry_point = green(1) + yellow(1) + allred(1) + green(2)/2
            % only this case in red period
        end % switch
elseif (on_route == 2)
    switch instant_entry
        case 1
            entry_point = green(1) + yellow(1) + allred(1) % in red period
            % time point away from the beginning of green at which EV enters
        case 2
            entry_point = green(1) + yellow(1) + allred(1) + green(2) / 2
        case 3
            entry_point = green(1) + yellow(1) + allred(1) + green(2)
        case 4
            entry_point = green(1) / 2 % only this case in green period
        end % switch
end

queue_length = zeros(1,cycle_entry)
total_produced = zeros(1,cycle_entry)
total_discharged = zeros(1,cycle_entry)

```

```

queue_last_cycle = 0
pointer = 1
t = arrival(pointer) % arrival time in sec, and the possibly earliest one arrives at 0 sec
if (on_route == 1) || (on_route == 3)
    t = t - offset
end
queue_start=pointer
for cycle = 1:cycle_entry
    if t >= cycle * cycle_length % wait to clear in next cycle
        continue
    end
    if (cycle==cycle_entry) && (instant_entry == 1)
        if (on_route == 1) || (on_route == 3)
            % Ev is coming at the beginning of green and no need to calculate other cycles
            break
        end
    end
end
queue_green = queue_last_cycle
queue_red = 0
discharged_green = 0
green_time_available = green(1) + yellow_equival_green + 0.01
% for counting calculation error

if queue_green > 0 % processing queue from last cycle
    for k = 1:queue_green
        green_time_available = green_time_available - turn(queue_start) *
            discharge_rate_green
        if green_time_available < 0
            break
        end
        discharged_green = k
    end % for
    total_discharged(cycle) = total_discharged(cycle) + discharged_green
    queue_start = queue_start + discharged_green
    queue_green = queue_green - discharged_green
end % if queue_green > 0

while t < cycle * cycle_length % processing arrivals during this cycle
    total_produced(cycle) = total_produced(cycle) + 1
    t_quotient = fix(t / cycle_length)
    t_elapsed = t-cycle_length * t_quotient
    % time elapsed from the beginning of green interval
    if (cycle==cycle_entry) && (t_elapsed>=entry_point)
        % EV is coming and no need to calculate other cycles
        break
    end
end

```

```

end
if (t_elapsed >= 0) && (t_elapsed < green(1) + yellow_equival_green)
    indicator = 'green or first half yellow'
    if queue_green > 0
        % there already exists queue that cannot be discharged during this green
period
        queue_green = queue_green + 1
    elseif queue_green == 0 % deal with the situation that no queue exists
        if (green(1) - t_elapsed) < green_time_available
            green_time_available = green(1) - t_elapsed
        end
        green_time_available = green_time_available - turn(queue_start) *
discharge_rate_green
        if green_time_available < 0 % queue produced
            queue_green = queue_green + 1
        elseif green_time_available >= 0
            discharged_green = discharged_green + 1
            total_discharged(cycle) = total_discharged(cycle) + 1
            queue_start = queue_start + 1
        end
    end % if queue_green > 0
elseif (t_elapsed >= green(1) + yellow_equival_green)
    indicator = 'second half yellow or red'
    queue_red = queue_red + 1
end % indicator is 'green' or 'red'
if pointer >= length(arrival)
    break
end
pointer = pointer + 1
t = arrival(pointer)
if (on_route == 1) || (on_route == 3)
    t = t - offset
end
end % while
undischarged(cycle) = queue_green % undischarged during this green interval
queue_length(cycle) = queue_green + queue_red
queue_last_cycle = queue_length(cycle)
if cycle == cycle_entry % EV is coming and no need to calculate other cycles
    break
end
end % for

queue = queue_last_cycle % queue from normal operation
queue_undischarged = 0
green_side_out = 0

```

```

time_spare_out = 0

if on_route == 2 % crossing road
    t_green_side = green_side_in
    switch instant_entry % calculate amber time prior to giving green to side road
        case {1}
            t_amber_1 = 0
        case {2,3}
            t_amber_1 = yellow(2) + allred(2)
        case 4
            t_amber_1 = 0 % only this case in red period
    end % switch
    t_amber_2 = yellow(1) + allred(1) + sti
    % calculate amber time after giving green to side road
    if t_green_side < 0.1 % == 0 for calculation error.
        % all t_ev is green for main road and red for side road.
        t_red_imposed_1 = 0 % red before getting side green
        t_red_imposed_2 = t_ev % red after getting side green
        yellow_equival_green = 0 % no yellow for side road
    else % t_green_side > 0
        t_red_imposed_1 = t_amber_1
        t_red_imposed_2 = t_ev - t_amber_1 - t_green_side
    end % if
    green_time_available = t_green_side + yellow_equival_green + 0.01
    if t_red_imposed_1 > 0 % produce queue prior to side green
        start = (cycle_entry - 1) * cycle_length + entry_point
        while (t < start + t_red_imposed_1)
            queue = queue + 1
            if pointer >= length(arrival)
                break
            end
            pointer = pointer + 1
            t = arrival(pointer)
        end % while
    end

    % processing queue from t_red_imposed_1
    discharged_green = 0
    if queue > 0
        for k = 1:queue
            green_time_available = green_time_available - turn(queue_start + k - 1) *
            discharge_rate_green
            if green_time_available < 0
                break
            end
        end
    end

```

```

        discharged_green = k
    end % for
    discharged_for_ev = discharged_green
    queue_start = queue_start + discharged_green
    queue = queue - discharged_green
end % if queue > 0

% process queue and arrivals after t_red_imposed_1
discharged_green = 0
queue_red = 0
start = (cycle_entry - 1) * cycle_length + entry_point + t_red_imposed_1
while (t >= start) && (t < start + t_green_side + t_red_imposed_2)
    if (t < start + t_green_side + yellow_equival_green)
        indicator = 'green or first half yellow'
        if queue > 0
            % there already exists queue that cannot be discharged during t_green_side
            queue = queue + 1
        elseif queue == 0 % deal with the situation that no queue exists
            if (start + t_green_side - t) < green_time_available
                green_time_available = start + t_green_side - t
            end
            green_time_available = green_time_available - turn(queue_start) *
            discharge_rate_green
            if green_time_available < 0 % queue produced
                queue = queue + 1
            elseif green_time_available >= 0
                discharged_green = discharged_green + 1
                queue_start = queue_start + 1
            end
        end % if queue_green > 0
    elseif (t >= start + t_green_side + yellow_equival_green)
        indicator = 'second half yellow or red'
        queue_red = queue_red + 1
    end % indicator is 'green' or 'red'
    if pointer >= length(arrival)
        break
    end
    pointer = pointer + 1
    t = arrival(pointer)
end % while
queue_formed = queue + queue_red
queue_length(cycle_entry) = queue_formed % produced during t_fixed
queue_out = arrival(queue_start:pointer-1)
turn_out = turn(queue_start:pointer-1)
elseif on_route == 1 % serving EV approach

```

```

discharged_green = 0
queue_equiv_length = 0
if queue == 0
    t_clear = 0
else % queue > 0
    for k = 1:queue
        queue_equiv_length = queue_equiv_length + turn(queue_start + k - 1)
    end

    t_clear_1 = queue_equiv_length * spacing / speed_mean
    % take general traffic to traverse queue length
    t_clear_2 = queue_equiv_length / saturation
    % time for discharging queue length
    if t_clear_1 > t_clear_2
        t_clear = t_clear_1 + t_startup
    else
        t_clear = t_clear_2 + t_startup
    end
    if t_clear > gmax(1)
        t_clear = gmax(1)
        successful = 'no, because clearance time greater than maximal time
        permitted.'
        return
    elseif t_clear < gmin(1)
        t_clear = gmin(1)
    end
end
end
switch instant_entry
    % calculate amber time prior to giving green to side road
    case {1}
        t_amber_1 = 0 % beginning of green
    case {2,3}
        t_amber_1 = yellow(1) + allred(1)
    case 4
        t_amber_1 = 0 % onle this case in red period
    end % switch
    t_amber_2 = yellow(2) + allred(2) + sti
    % calculate amber time after giving green to side road
    if t_clear <= 0.01 % ==0 for calculation error
        t_green_side = t_ev - t_amber_1 - t_amber_2
    else % t_clear>0
        t_green_side = t_ev - t_amber_1 - t_amber_2 - t_clear
    end
    if t_green_side > gmax(2)
        t_green_side = gmax(2)
    end
end

```

```

elseif t_green_side < gmin(2)
    t_green_side = 0
end
green_side_out = t_green_side
if t_green_side == 0 % all t_ev is green for main road and red for side road.
    t_green_main = t_ev
    yellow_equival_green = 0
else % t_green_side > 0
    t_green_main = t_ev - t_amber_1 - t_amber_2 - t_green_side
    switch instant_entry % calculate yellow_equival_green to main road
        case {2,3}
            yellow_equival_green = ( t_amber_1 - allred(1) ) * discharge_rate_green /
            discharge_rate_yellow
        case {1,4}
            yellow_equival_green = 0
        end % switch
    end % if
    green_time_available = t_green_main + yellow_equival_green + 0.01

if queue > 0 % processing queue from normal operation
    total_green_time_needed = 0
    for k = 1:queue
        total_green_time_needed = total_green_time_needed +
        turn(queue_start + k - 1) * discharge_rate_green
    end % for
    for k = 1:queue
        green_time_available = green_time_available -
        turn(queue_start + k - 1) * discharge_rate_green
        if green_time_available < 0
            break
        end
        discharged_green = k
    end % for
    discharged_for_ev = discharged_green
    queue_start = queue_start + discharged_green
    queue = queue - discharged_green
end % if queue > 0

if (queue > 0)
    successful = 'No!' % more time is needed
    time_needed = total_green_time_needed - t_green_main -
        yellow_equival_green
    queue_undischarged = queue
    queue_out = queue_undischarged + 0.1 % different from successful == 'yes'
    turn_out = turn(queue_start:queue_start+queue-1)

```



```

elseif (queue == 0)
    successful = 'Yes!'
    time_spare = green_time_available - yellow_equival_green
    % exclude yellow_equival_green
    time_spare = round(time_spare * 10)/10 % exclude yellow_equival_green
    time_spare_out = time_spare
    start = (cycle_entry - 1) * cycle_length + entry_point
    while (t <= start + t_ev)
        % produce affected vehicles during EV operation: t_ev
        queue = queue + 1
        if pointer >= length(arrival)
            break
        end
        pointer = pointer + 1
        t = arrival(pointer)
    end % while
    queue_length(cycle_entry) = queue % produced during t_fixed
    queue_out = arrival(queue_start:pointer-1)
    turn_out = turn(queue_start:pointer-1)
end %if
elseif on_route == 3 % opposing approach to serving EV one
    discharged_green = 0
    t_green_side = green_side_in
    switch instant_entry % calculate amber time prior to giving green to side road
    case {1}
        t_amber_1 = 0
    case {2,3}
        t_amber_1 = yellow(1) + allred(1)
    case 4
        t_amber_1 = 0 % only this case in red period
    end % switch
    t_amber_2 = yellow(2) + allred(2) + sti
    % calculate amber time after giving green to side road
    if t_green_side < 0.1 % == 0 for calculation error
        % all t_ev is green for main road and red for side road.
        t_red_imposed = 0
        t_green_main = t_ev
        yellow_equival_green = 0
    else % t_green_side > 0
        t_red_imposed = t_amber_1 + t_amber_2 + t_green_side
        t_green_main = t_ev - t_red_imposed
        switch instant_entry % calculate yellow_equival_green to main road
        case {2,3}
            yellow_equival_green = ( t_amber_1 - allred(1) ) *
            discharge_rate_green / discharge_rate_yellow

```

```

        case {1,4}
            yellow_equival_green = 0
        end % switch
    end % if
    green_time_available = t_green_main + yellow_equival_green + 0.01

    if t_red_imposed > 0 % produce queue during t_red_imposed
        start = (cycle_entry - 1) * cycle_length + entry_point
        while (t >= start) && (t <= start + t_red_imposed)
            queue = queue + 1
            if pointer >= length(arrival)
                break
            end
            pointer = pointer + 1
            t = arrival(pointer)
        end % while
    end

    % processing queue from t_red_imposed
    if queue > 0
        for k = 1:queue
            green_time_available = green_time_available - turn(queue_start + k - 1) *
            discharge_rate_green
            if green_time_available < 0
                break
            end
            discharged_green = k
        end % for
        discharged_for_ev = discharged_green
        queue_start = queue_start + discharged_green
        queue = queue - discharged_green
    end % if queue > 0
    queue_formed = queue
    queue_length(cycle_entry) = queue % produced at the end of t_ev period
    queue_out = arrival(queue_start:pointer-1)
    turn_out = turn(queue_start:pointer-1)
end % if on_route

```

H.4 Smooth transitioning for transition 2

```

t_fixed=50
for instant_entry=1:4
    for cycle_entry=5:10
        [queue_length,turn]=smooth_route(t_fixed,cycle_entry,instant_entry);
    end
end

function [queue_length,turn]=smooth_route(t_fixed,cycle_entry,instant_entry);
% smooth tranisioning for coordinated route
offset_1 = 40 % between intersection 1 and 2
offset_2 = 25 % between intersection 3 and 3
offset_3 = 30 % between intersection 3 and 4

% control for intersection 1
offset = 0
arrival_1 = dlmread('arrival_1e.dat')
% through arrivals at an intersection from eastbound approach to the route
arrival_2 = dlmread('arrival_1n.dat') % right turn to the route
arrival_3 = dlmread('arrival_1s.dat') % left turn to the route
arrival_4 = dlmread('arrival_1w.dat') % through vehicles from westbound approach
turn_1 = dlmread('turn_1e.dat') % turn factor for eastbound through vehicles
turn_2 = dlmread('turn_1n.dat') % right turn
turn_3 = dlmread('turn_1s.dat') % left turn
turn_4 = dlmread('turn_1w.dat') % through vehicles from westbound approach
[queue_length,turn]= smooth_intersection(t_fixed,cycle_entry,instant_entry, offset,
arrival_1,turn_1,arrival_2,turn_2,arrival_3,turn_3,arrival_4,turn_4);

% control for intersection 2
offset = offset_1
arrival_1 = dlmread('arrival_2e.dat')
% through arrivals at an intersection from eastbound approach to the route
arrival_2 = dlmread('arrival_2n.dat') % right turn to the route
arrival_3 = dlmread('arrival_2s.dat') % left turn to the route
arrival_4 = dlmread('arrival_2w.dat') % through vehicles from westbound approach
turn_1 = dlmread('turn_2e.dat') % turn factor for eastbound through vehicles
turn_2 = dlmread('turn_2n.dat') % right turn
turn_3 = dlmread('turn_2s.dat') % left turn
turn_4 = dlmread('turn_2w.dat') % through vehicles from westbound approach
[queue_length,turn]= smooth_intersection(t_fixed,cycle_entry,instant_entry, offset,
arrival_1,turn_1,arrival_2,turn_2,arrival_3,turn_3,arrival_4,turn_4);

```

```

% control for intersection 3
offset = offset_1 + offset_2
arrival_1 = dlmread('arrival_3e.dat')
% through arrivals at an intersection from eastbound approach to the route
arrival_2 = dlmread('arrival_3n.dat') % right turn to the route
arrival_3 = dlmread('arrival_3s.dat') % left turn to the route
arrival_4 = dlmread('arrival_3w.dat') % through vehicles from westbound approach
turn_1 = dlmread('turn_3e.dat') % turn factor for eastbound through vehicles
turn_2 = dlmread('turn_3n.dat') % right turn
turn_3 = dlmread('turn_3s.dat') % left turn
turn_4 = dlmread('turn_3w.dat') % through vehicles from westbound approach
[queue_length,turn]= smooth_intersection(t_fixed,cycle_entry,instant_entry, offset,
arrival_1,turn_1,arrival_2,turn_2,arrival_3,turn_3,arrival_4,turn_4);

% control for intersection 4
offset = offset_1 + offset_2 + offset_3
arrival_1 = dlmread('arrival_4e.dat')
% through arrivals at an intersection from eastbound approach to the route
arrival_2 = dlmread('arrival_4n.dat') % right turn to the route
arrival_3 = dlmread('arrival_4s.dat') % left turn to the route
arrival_4 = dlmread('arrival_4w.dat') % through vehicles from westbound approach
turn_1 = dlmread('turn_4e.dat') % turn factor for eastbound through vehicles
turn_2 = dlmread('turn_4n.dat') % right turn
turn_3 = dlmread('turn_4s.dat') % left turn
turn_4 = dlmread('turn_4w.dat') % through vehicles from westbound approach
[queue_length,turn]= smooth_intersection(t_fixed,cycle_entry,instant_entry, offset,
arrival_1,turn_1,arrival_2,turn_2,arrival_3,turn_3,arrival_4,turn_4);

%-----
function [queue_length,turn]= smooth_intersection(t_fixed,cycle_entry,instant_entry,
offset,arrival_1,turn_1,arrival_2,turn_2,arrival_3,turn_3,arrival_4,turn_4);
% generate queue length at an intersection after 50-sec fixed held time for transition 1
% queue_length[1:4,1:cycle_entry]= queue formed on approach 1 to 4
% turn= turning factor
% t_fixed= fixed held time for EV operation
% cycle_entry= cycle # EV enters the system
% instant_entry= 1 for EV enters at g1=0, 2 for g1/2, 3 for g1, and 4 for g2/2
% offset=offset relative to the beginning of green for the 1st intersection
% arrival_# and turn_#=vehicle arrivals on approach # and turn factors

green = [60,40]
yellow = [3,3]
allred = [1,1]
on_route = 1 % serving EV approach
turn_flag = 1 % through

```

```

offset_1 = offset
[queue_out_1, turn_out_1, undischarged_1, time_recovery_out_1] =
smooth_approach(on_route, turn_flag, offset_1, arrival_1, turn_1, cycle_entry, instant_entr
y, t_fixed, green, yellow, allred)

on_route = 2
turn_flag = 2 % right turn
offset_2 = offset_1 + 64 %green(1)+yellow(1)+allred(1)
green = [40,60]
[queue_out_2, turn_out_2, undischarged_2, time_recovery_out_2] =
smooth_approach(on_route, turn_flag, offset_2, arrival_2, turn_2, cycle_entry, instant_entr
y, t_fixed, green, yellow, allred)

on_route = 2
turn_flag = 3 % left turn
offset_3 = offset_1 + 64
green = [40,60]
[queue_out_3, turn_out_3, undischarged_3, time_recovery_out_3] =
smooth_approach(on_route, turn_flag, offset_3, arrival_3, turn_3, cycle_entry, instant_entr
y, t_fixed, green, yellow, allred)

on_route = 3 % opposing approach
turn_flag = 1 % through
offset_4 = offset_1
green = [60,40]
[queue_out_4, turn_out_4, undischarged_4, time_recovery_out_4] =
smooth_approach(on_route, turn_flag, offset_4, arrival_4, turn_4, cycle_entry, instant_entr
y, t_fixed, green, yellow, allred)

% calculate average recovery time, and only time_recovery_out_1 is useful
if (cycle_entry == 5) && (instant_entry == 1)
    fid = fopen('time_recovery.dat','w') % 1 for standard, 2 for error
else
    fid = fopen('time_recovery.dat','a') % append data to the end of the file
end
fprintf(fid, '%6.2f\n', time_recovery_out_1)
status = fclose(fid)
if status == -1
    disp(message)
    return
end
if (cycle_entry == 10) && (instant_entry == 4)
    fid = fopen('time_recovery.dat','r') % 1 for standard, 2 for error
    if fid == -1 % no such file
        disp(message)
    end
end

```

```

        return
    end
    time_recovery_vector = []
    time_recovery_vector = fscanf(fid,'%g')
    % app. 1 for one row, totally 24 columns(4 instant_entry x6cycle)
    status =fclose(fid)
    if status == -1 % =0 if successful closed
        disp('error')
        return
    end
    cycle_length = green(1) + yellow(1) + allred(1) + green(2) + yellow(2) +
allred(2)
    time_recovery_vector = time_recovery_vector * cycle_length
    time_recovery_average = mean(time_recovery_vector) % averages of columns
    standard_deviation = std(time_recovery_vector) % std of columns
end

% form queue length at the intersection
queue_length = []
queue_length(1) = length(queue_out_1)
queue_length(2) = length(queue_out_2)
queue_length(3) = length(queue_out_3)
queue_length(4) = length(queue_out_4)
% queue_average = mean(queue_length) % averages of vector
% standard_deviation = std(queue_length) % std of vector
turn = []
turn(1,1:length(turn_out_1)) = turn_out_1
turn(2,1:length(turn_out_2)) = turn_out_2
turn(3,1:length(turn_out_3)) = turn_out_3
turn(4,1:length(turn_out_4)) = turn_out_4

if (cycle_entry == 5) && (instant_entry == 1)
    fid = fopen('queue_smooth.dat','w') % 1 for standard, 2 for error
else
    fid = fopen('queue_smooth.dat','a') % append data to the end of the file
end
fprintf(fid,'%6.2f%6.2f%6.2f%6.2f\n',queue_length)
status = fclose(fid)
if status == -1
    disp(message)
    return
end
if (cycle_entry == 10) && (instant_entry == 4)
    fid = fopen('queue_smooth.dat','r') % 1 for standard, 2 for error
    if fid == -1 % no such file

```

```

        disp(message)
        return
    end
    queue_length = []
    queue_length = fscanf(fid,'%g %g %g %g ', [4 inf])
    % app. 1,2, 3 and 4 for each row, totally 24 columns(4 instant_entry x6cycle)
    status = fclose(fid)
    if status == -1 % =0 if successful closed
        disp('error')
        return
    end
    queue_average = mean(queue_length') % averages of columns
    standard_deviation = std(queue_length') % std of columns
end

%-----
function [queue_out, turn_out, queue_undischarged, time_recovery_out] =
smooth_approach(on_route, turn_flag, offset, arrival, turn, cycle_entry, instant_entry, t_fixed, green, yellow, allred)
% generate queue length on an approach under smooth control strategy
% after 50-sec fixed held time for transition 1
% queue_out, turn_out[1:2]= queue formed on the approach and turning factors
% queue_undischarged= number of queue ahead of EV undischarged when EV reaches
% the intersection, applicable only if on_route==1
% time_recovery_out= cycle_length*cycle_number_needed for recovery
% on_route= 1 for approach serving EV; 2 for crossing road, and 3 for opposing
% approach to serving EV one
% turn_flag= 1 for through, 2 for right turn, and 3 for left turn
% arrival= arrival time for each vehicle relative to the beginning ( 0 sec)
% turn= turning factor corresponding to each arrival with straight-through (1.0),
% left-turn (1.4), or right-turn (1.2)
% offset= beginning of green of an approach relative to the through approach of the
first
% intersection
% t_fixed= fixed held time for EV operation

cycle_length = green(1) + yellow(1) + allred(1) + green(2) + yellow(2) + allred(2)
saturation = 2000 / 3600 % in veh/sec
discharge_rate_green = 1 / saturation
% for straight-through vehicles during green in sec/veh
discharge_rate_yellow = 2 * discharge_rate_green
% for straight-through vehicles during yellow in sec/veh
yellow_equival_green = yellow(1) * discharge_rate_green / discharge_rate_yellow
% transfer yellow interval to equivalent green discharging vehicles

```

```

gmin = [7,10]
gmax = [100,100]

% instant_entry: 1 for g1=0, 2 for g1/2, 3 for g1, or 4 for g1+y1+allred1+g2/2
if (on_route == 1) || (on_route == 3)
    switch instant_entry
        case 1
            entry_point = 0
            % time point away from the beginning of green at which EV enters
        case 2
            entry_point = green(1) / 2
        case 3
            entry_point = green(1)
        case 4
            entry_point = green(1) + yellow(1) + allred(1) + green(2)/2
            % only this case in red period
        end % switch
elseif (on_route == 2)
    switch instant_entry
        case 1
            entry_point = green(1) + yellow(1) + allred(1) % in red period
            % time point away from the beginning of green at which EV enters
        case 2
            entry_point = green(1) + yellow(1) + allred(1) + green(2) / 2
        case 3
            entry_point = green(1) + yellow(1) + allred(1) + green(2)
        case 4
            entry_point = green(1) / 2 % only this case in green period
        end % switch
end

queue_length = zeros(1,cycle_entry)
total_produced = zeros(1,cycle_entry)
total_discharged = zeros(1,cycle_entry)
queue_last_cycle = 0
pointer = 1
t = arrival(pointer) % arrival time in sec, and the possibly earliest one arrives at 0 sec
if (on_route == 1) || (on_route == 3)
    t = t - offset
end
queue_start = pointer
for cycle = 1:cycle_entry
    if t >= cycle * cycle_length % wait to clear in next cycle
        continue
    end
end

```



```

if (cycle==cycle_entry) && (instant_entry == 1)
    if (on_route == 1) || (on_route == 3)
        % Ev is coming at the beginning of green and no need to calculate other cycles
        break
    end
end
queue_green = queue_last_cycle
queue_red = 0
discharged_green = 0
green_time_available = green(1) + yellow_equival_green

if queue_green > 0 % processing queue from last cycle
    for k = 1:queue_green
        green_time_available = green_time_available - turn(queue_start) *
            discharge_rate_green
        if green_time_available < 0
            break
        end
        discharged_green = k
    end % for
    total_discharged(cycle) = total_discharged(cycle) + discharged_green
    queue_start = queue_start + discharged_green
    queue_green = queue_green - discharged_green
end % if queue_green > 0

while t < cycle * cycle_length % processing arrivals during this cycle
    total_produced(cycle) = total_produced(cycle) + 1
    t_quotient = fix(t / cycle_length)
    t_elapsed = t - cycle_length * t_quotient
    % time elapsed from the beginning of green interval
    if (cycle==cycle_entry) && (t_elapsed >= entry_point)
        % EV is coming and no need to calculate other cycles
        break
    end
    if (t_elapsed >= 0) && (t_elapsed < green(1) + yellow_equival_green)
        indicator = 'green or first half yellow'
        if queue_green > 0
            % there already exists queue that cannot be discharged during this green
period
            queue_green = queue_green + 1
        elseif queue_green == 0 % deal with the situation that no queue exists
            if (green(1) - t_elapsed) < green_time_available
                green_time_available = green(1) - t_elapsed
            end
            green_time_available = green_time_available - turn(queue_start) *

```

```

        discharge_rate_green
        if green_time_available < 0 % queue produced
            queue_green = queue_green + 1
        elseif green_time_available >= 0
            discharged_green = discharged_green + 1
            total_discharged(cycle) = total_discharged(cycle) + 1
            queue_start = queue_start + 1
        end
    end % if queue_green > 0
elseif (t_elapsed >= green(1) + yellow_equival_green)
    indicator = 'second half yellow or red'
    queue_red = queue_red + 1
end % indicator is 'green' or 'red'
if pointer >= length(arrival)
    break
end
pointer = pointer + 1
t = arrival(pointer)
if (on_route == 1) || (on_route == 3)
    t = t - offset
end
end % while
undischarged(cycle) = queue_green % undischarged during this green interval
queue_length(cycle) = queue_green + queue_red
queue_last_cycle = queue_length(cycle)
if cycle == cycle_entry % EV is coming and no need to calculate other cycles
    break
end
end % for

% begin fixed held time transition 1
queue = queue_last_cycle % queue from normal operation
queue_undischarged = 0
time_spare_out = 0

if on_route == 2 % crossing road
    red_time_imposed = t_fixed
    start = (cycle_entry - 1) * cycle_length + entry_point
    while (t <= start + red_time_imposed) % produce vehicles during held time, t_fixed
        queue = queue + 1
        if pointer >= length(arrival)
            break
        end
        pointer = pointer + 1
        t = arrival(pointer)
    end
end

```

```

    end % while
    queue_length(cycle_entry) = queue % produced during t_fixed
elseif on_route == 1 % serving EV approach
    discharged_green = 0
    green_time_available = t_fixed
    if queue > 0 %discharge queue from normal operation
        total_green_time_needed = 0
        for k = 1:queue
            total_green_time_needed = total_green_time_needed + turn(queue_start + k - 1)
*
            discharge_rate_green
        end % for
        for k = 1:queue
            green_time_available = green_time_available - turn(queue_start + k - 1) *
            discharge_rate_green
            if green_time_available < 0
                break
            end
            discharged_green = k
        end % for
        discharged_for_ev = discharged_green
        queue_start = queue_start + discharged_green
        queue = queue - discharged_green
    end % if queue > 0

    if (queue > 0)
        successful = 'No!' % more time is needed
        time_needed = total_green_time_needed - t_fixed
        queue_undischarged = queue
        return
    elseif (queue == 0)
        successful = 'Yes!'
        time_spare = green_time_available
        time_spare_out = time_spare
        start = (cycle_entry - 1) * cycle_length + entry_point
        while (t <= start + t_fixed) % produce vehicles during held time,t_fixed
            queue = queue + 1
            if pointer >= length(arrival)
                break
            end
            pointer = pointer + 1
            t = arrival(pointer) - offset
        end % while
        queue_length(cycle_entry) = queue % produced during t_fixed
    end %if

```

```

elseif on_route == 3 % opposing approach to serving EV one
    discharged_green = 0
    green_time_available = t_fixed
    if queue > 0 % processing queue from normal operation
        total_green_time_needed = 0
        for k = 1:queue
            total_green_time_needed = total_green_time_needed + turn(queue_start + k - 1)
        *
            discharge_rate_green
        end % for
        for k = 1:queue
            green_time_available = green_time_available - turn(queue_start + k - 1) *
            discharge_rate_green
            if green_time_available < 0
                break
            end
            discharged_green = k
        end % for
        discharged_for_ev = discharged_green
        queue_start = queue_start + discharged_green
        queue = queue - discharged_green
    end % if queue > 0

    start = (cycle_entry - 1) * cycle_length + entry_point
    while (t <= start + t_fixed) % produce vehicles during held time,t_fixed
        if queue > 0
            % there already exists queue that cannot be discharged during this green
period
            queue = queue + 1
        elseif queue == 0 % deal with the situation that no queue exists
            if (start + t_fixed - t) < green_time_available
                green_time_available = start + t_fixed - t
            end
            green_time_available = green_time_available - turn(queue_start) *
            discharge_rate_green
            if green_time_available < 0 % queue produced
                queue = queue + 1
            elseif green_time_available >= 0
                discharged_green = discharged_green + 1
                queue_start = queue_start + 1
            end
        end % if queue > 0
        if pointer >= length(arrival)
            break
        end
    end

```

```

        pointer = pointer + 1
        t = arrival(pointer) - offset
    end % while
    queue_length(cycle_entry) = queue % produced at the end of t_fixed period
end % if on_route
queue_last_cycle = queue % queue from t_fixed

% The following begins smooth transition 2.
% output transition period, queue length
time_recovery_out = 0
queue_out = 0
turn_out = 0
switch on_route
case 1 % serving EV approach
    t_fin = entry_point + t_fixed % finishing point of EV operation, t_fin.
    t_fin_quotient = fix(t_fin / cycle_length)
    t_fin_elapsed = t_fin - cycle_length * t_fin_quotient
    % away from the beginning of green
    t_dif = green(1) - t_fin_elapsed
    % time difference between the finishing point and the end of green interval of
    % background cycle
    if abs(t_dif) < 0.1
        indicator = 'no need to adjust, get back to normal operation immediately.'
        cycle_needed = 0
        time_recovery_out = 0
        queue_out = arrival(queue_start:pointer-1)
        % produced at the end of t_fixed period = queue
        turn_out = turn(queue_start:pointer-1)
        return
    elseif t_dif > 0
        indicator = 'in green interval.'
        n_cycle_plus = ceil(t_dif / ( 0.2*green(1) + 0.2* green(2) ) ) % addition option
        t_dif_remained = t_dif
        green_plus(1) = 0
        t_trans_1 = 0
        for k = 1:n_cycle_plus
            if t_dif_remained <= 0.2*green(2)
                red_plus(k) = t_dif_remained
            else
                red_plus(k) = 0.2*green(2)
            end
            if red_plus(k) + green(2) > gmax(2)
                red_plus(k) = gmax(2) - green(2)
            end
            t_dif_remained = t_dif_remained - red_plus(k)
        end
        cycle_needed_plus = k
    end
end

```

```

t_trans_1 = t_trans_1 + red_plus(k)
t_dif_remained = t_dif_remained - red_plus(k)
if abs(t_dif_remained) < 0.1
    break
end
if t_dif_remained <= 0.2*green(1)
    green_plus(k+1) = t_dif_remained
    red_plus(k+1) = 0
else
    green_plus(k+1) = 0.2*green(1)
end
if green_plus(k+1) + green(1) > gmax(1)
    green_plus(k+1) = gmax(1) - green(1)
end
cycle_needed_plus = k+1
t_trans_1 = t_trans_1 + green_plus(k+1)
t_dif_remained = t_dif_remained - green_plus(k+1)
if abs(t_dif_remained) < 0.1
    break
end
end % for addition option
t_trans_2 = 0 % subtraction option
n_cycle_sub = ceil( (green(1) - t_dif) / (0.17*green(1) + 0.17* green(2) ) )
t_dif_remained = green(1) - t_dif
for k = 1:n_cycle_sub
    if t_dif_remained <= 0.17*green(1)
        green_sub(k) = t_dif_remained
    else
        green_sub(k) = 0.17*green(1)
    end
    if green(1) - green_sub(k) < gmin(1)
        green_sub(k) = green(1) - gmin(1)
    end
    cycle_needed_sub = k
    t_trans_2 = t_trans_2 + green_sub(k)
    t_dif_remained = t_dif_remained - green_sub(k)
    if abs(t_dif_remained) < 0.1
        red_sub(k) = 0
        break
    end
    if t_dif_remained <= 0.17*green(2)
        red_sub(k) = t_dif_remained
    else
        red_sub(k) = 0.17*green(2)
    end
end

```

```

    if green(2)- red_sub(k) < gmin(2)
        red_sub(k) = green(2) - gmin(2)
    end
    t_trans_2 = t_trans_2 + red_sub(k)
    t_dif_remained = t_dif_remained - red_sub(k)
    if abs(t_dif_remained) < 0.1
        break
    end
end % for subtraction option
else % t_dif < 0
    indicator = 'in red interval.'
    minus_t_dif = -t_dif
    n_cycle_plus = ceil( (green(2) - minus_t_dif) / (0.2*green(1) + 0.2* green(2) ) )
    t_dif_remained = green(2) - minus_t_dif
    green_plus(1) = 0
    red_plus(1) = 0
    t_trans_1 = 0
    for k = 1:n_cycle_plus % addition option
        if t_dif_remained <= 0.2*green(1)
            green_plus(k+1) = t_dif_remained
        else
            green_plus(k+1) = 0.2*green(1)
        end
        if green_plus(k+1) + green(1) > gmax(1)
            green_plus(k+1) = gmax(1) - green(1)
        end
        cycle_needed_plus = k+1
        t_trans_1 = t_trans_1 + green_plus(k+1)
        t_dif_remained = t_dif_remained - green_plus(k+1)
        if abs(t_dif_remained) < 0.1
            red_plus(k+1) = 0
            break
        end
        if t_dif_remained <= 0.2*green(2)
            red_plus(k+1) = t_dif_remained
        else
            red_plus(k+1) = 0.2*green(2)
        end
        if red_plus(k+1) + green(2) > gmax(2)
            red_plus(k+1) = gmax(2) - green(2)
        end
        t_trans_1 = t_trans_1 + red_plus(k+1)
        t_dif_remained = t_dif_remained - red_plus(k+1)
        if abs(t_dif_remained) < 0.1
            break
        end
    end
end

```

```

    end
end % for addition option
t_trans_2 = 0 % subtraction option
n_cycle_sub = ceil( minus_t_dif / (0.17*green(1) + 0.17* green(2) ) )
t_dif_remained = minus_t_dif
green_sub(1) = 0
for k = 1:n_cycle_sub
    if t_dif_remained <= 0.17*green(2)
        red_sub(k) = t_dif_remained
    else
        red_sub(k) = 0.17*green(2)
    end
    if green(2) - red_sub(k) < gmin(2)
        red_sub(k) = green(2) - gmin(2)
    end
    cycle_needed_sub = k
    t_trans_2 = t_trans_2 + red_sub(k)
    t_dif_remained = t_dif_remained - red_sub(k)
    if abs(t_dif_remained) < 0.1
        break
    end
    if t_dif_remained <= 0.17*green(1)
        green_sub(k+1) = t_dif_remained
    else
        green_sub(k+1) = 0.17*green(1)
    end
    if green(1) - green_sub(k+1) < gmin(1)
        green_sub(k+1) = green(1) - gmin(1)
    end
    cycle_needed_sub = k+1
    t_trans_2 = t_trans_2 + green_sub(k+1)
    t_dif_remained = t_dif_remained - green_sub(k+1)
    if abs(t_dif_remained) < 0.1
        red_sub(k+1) = 0
        break
    end
end
end % for subtraction option
end % if abs(t_dif) < 0.1
if abs(t_dif) > 0.1
    if (cycle_needed_plus < cycle_needed_sub) ||
        ((cycle_needed_plus == cycle_needed_sub) && (t_trans_1 < t_trans_2))
        % choose addition option
        t_transition = t_trans_1
        cycle_needed = cycle_needed_plus
        for k = 1:cycle_needed

```



```

        green_adj(k) = green_plus(k)
        red_adj(k) = red_plus(k)
    end
elseif (cycle_needed_plus > cycle_needed_sub) ||
((cycle_needed_plus == cycle_needed_sub) && (t_trans_1 >= t_trans_2))
    % choose subtraction option
    t_transition = t_trans_2
    cycle_needed = cycle_needed_sub
    for k = 1:cycle_needed
        green_adj(k) = -green_sub(k)
        red_adj(k) = -red_sub(k)
    end
else
    indicator = 'situation impossible happened.'
    return
end
end
time_recovery_out = cycle_needed
if (cycle_entry == 5) && (instant_entry == 1)
    fid = fopen('trans_1.dat','w') % 1 for standard, 2 for error
elseif (cycle_entry == 5) && (instant_entry > 1)
    fid = fopen('trans_1.dat','a') % append data to the end of the file
end
if cycle_entry == 5
    fprintf(fid,'%d \n', cycle_needed)
    if cycle_needed > 0
        fprintf(fid,'%6.2f \n',green_adj)
        fprintf(fid,'%6.2f \n',red_adj)
    end
    status = fclose(fid)
    if status == -1 % =0 if successful closed
        disp('error')
        return
    end
end
end

for k = 1: cycle_needed
    if green_adj(k) == 0
        green_time(k) = 0
    else
        green_time(k) = green(1) + green_adj(k)
    end
    if red_adj(k) == 0
        red_time(k) = 0
    else

```

```

        red_time(k) = green(2) + red_adj(k)
    end
end
if (cycle_entry == 5) && (instant_entry == 1)
    fid = fopen('trans_1_timing.dat','w') % 1 for standard, 2 for error
elseif (cycle_entry == 5) && (instant_entry > 1)
    fid = fopen('trans_1_timing.dat','a') % append data to the end of the file
end
if cycle_entry == 5
    fprintf(fid,'%d \n', cycle_needed)
    if cycle_needed > 0
        fprintf(fid,'%6.2f \n',green_time)
        fprintf(fid,'%6.2f \n',red_time)
    end
    status = fclose(fid)
    if status == -1 % =0 if successful closed
        disp('error')
        return
    end
end
if cycle_needed == 0
    queue_formed = queue_last_cycle %queue from transition 1
    queue_out = arrival(queue_start:pointer-1)
    % produced at the end of t_fixed period =queue
    turn_out = turn(queue_start:pointer-1)
    return
end
start_point = (cycle_entry - 1) * cycle_length + entry_point + t_fixed
for cycle = 1:cycle_needed
    queue_green = queue_last_cycle % queue from transition 1
    if green_time(cycle)==0 && (red_adj(cycle) < 0 || red_adj(cycle) == 0)
        % falls in red interval
        % red_adj(cycle) < 0: in first half red; red_adj(cycle) == 0: in second half red
        % red_adj(cycle) > 0: in second half green
        amber(1) = 0
        green_time_available = 0
    else % falls in green interval
        amber(1) = yellow(1) + allred(1)
        green_time_available = green_time(cycle) + yellow_equival_green
    end
    if (red_time(cycle) == 0) && (cycle == cycle_needed)
        % in the last cycle and no adjustment to red
        red_time_imposed = green(2) + yellow(2) + allred(2)
    else
        red_time_imposed = red_time(cycle) + yellow(2) + allred(2)
    end
end

```

```

end
queue_red = 0
discharged_green = 0
total_discharged = 0
if (queue_green > 0) && (green_time_available > 0)
    % processing queue from transition 1
    for k = 1:queue_green
        green_time_available = green_time_available - turn(queue_start) *
        discharge_rate_green
        if green_time_available < 0
            break
        end
        discharged_green = k
    end % for
    total_discharged = total_discharged + discharged_green
    queue_start = queue_start + discharged_green
    queue_green = queue_green - discharged_green
end % if queue_green > 0
while (t >= start_point) &&
(t < start_point + green_time(cycle) + amber(1) + red_time_imposed)
    if t < start_point + green_time(cycle) + amber(1)
        indicator = 'green or first half yellow'
        if queue_green > 0 % there already exists queue that cannot be discharged
            % during this green period
            queue_green = queue_green + 1
        elseif queue_green == 0 % deal with the situation that no queue exists
            if (start_point + green_time(cycle) - t) < green_time_available
                green_time_available = start_point + green_time(cycle) - t
            end
            green_time_available = green_time_available - turn(queue_start) *
            discharge_rate_green
            if green_time_available < 0 % queue produced
                queue_green = queue_green + 1
            elseif green_time_available >= 0
                discharged_green = discharged_green + 1
                total_discharged = total_discharged + 1
                queue_start = queue_start + 1
            end
        end % if queue_green > 0
    elseif t >= start_point + green_time(cycle) + amber(1)
        indicator = 'second half yellow or red'
        queue_red = queue_red + 1
    end % indicator is 'green' or 'red'
    if pointer >= length(arrival)
        break
    end
end

```

```

        end
        pointer = pointer + 1
        t = arrival(pointer) - offset
    end % while
    queue_formed = queue_green + queue_red % queue formed in this cycle
    queue_last_cycle = queue_formed
    start_point = start_point + green_time(cycle) + amber(1) + red_time_imposed
    queue_out = arrival(queue_start:pointer-1)
    turn_out = turn(queue_start:pointer-1)
end % for cycle
case 2 % crossing roads
% get phase timing for transiting from approach 1
fid = fopen('trans_1_timing.dat','r') % 1 for standard, 2 for error
if fid == -1 % no such file
    disp(message)
    return
end
for instant = 1:instant_entry
    cycle_needed = fscanf(fid,'%d', 1)
    if cycle_needed == 0
        continue
    end
    red_time = fscanf(fid,'%f',cycle_needed)
    % green for main roads is red for side roads
    green_time = fscanf(fid,'%f',cycle_needed)
    % red for main roads is green for side roads
    % in order to starts and ends transition 2 on side roads at the same time as on
    % main roads, signal cycle for side roads starts at red=4 sec, followed by green
    % period, and then ends at red=4 sec, where green(1) starts and ends.
end
status = fclose(fid)
if status == -1 % =0 if successful closed
    disp('error')
    return
end
if cycle_needed == 0
    queue_formed = queue % queue from t_fixed transition 1
    queue_out = arrival(queue_start:pointer-1)
    % produced at the end of t_fixed period = queue
    turn_out = turn(queue_start:pointer-1)
    return
end
queue_last_cycle = queue % queue from t_fixed
t_fin = entry_point + t_fixed % finishing point of EV operation, t_fin.
t_fin_quotient = fix(t_fin / cycle_length)

```

```

t_fin_elapsed = t_fin - cycle_length * t_fin_quotient
% away from the beginning of green
t_dif = green(1) + yellow(1) + allred(1) + green(2) - t_fin_elapsed
% time difference between the finishing point and the end of green(1) interval of
% background cycle
if abs(t_dif) < 0.1
    indicator = 'no need to adjust, get back to normal operation immediately.'
    queue_out = arrival(queue_start:pointer-1)
    turn_out = turn(queue_start:pointer-1)
    return
end

start_point = (cycle_entry - 1) * cycle_length + entry_point + t_fixed
for cycle = 1:cycle_needed
    queue_red = queue_last_cycle % queue from transition 1
    red_time_imposed = red_time(cycle) + yellow(2) + allred(2)
    while (t >= start_point) && (t < start_point + red_time_imposed)
        indicator = 'in red'
        queue_red = queue_red + 1
        if pointer >= length(arrival)
            break
        end
        pointer = pointer + 1
        t = arrival(pointer)
    end % while
    start_point = start_point + red_time_imposed
    if green_time(cycle) == 0 && cycle == cycle_needed
        green_time_imposed = green(1)
    else
        green_time_imposed = green_time(cycle)
    end
    green_time_available = green_time_imposed + yellow_equival_green
    amber(1) = yellow(1) + allred(1)
    queue_green = queue_red
    queue_red = 0
    discharged_green = 0
    total_discharged = 0
    if (queue_green > 0) && (green_time_available > 0)
        % processing queue from red interval
        for k = 1:queue_green
            green_time_available = green_time_available - turn(queue_start) *
            discharge_rate_green
            if green_time_available < 0
                break
            end
        end
    end
end

```

```

    discharged_green = k
  end % for
  total_discharged = total_discharged + discharged_green
  queue_start = queue_start + discharged_green
  queue_green = queue_green - discharged_green
end % if queue_green > 0
while (t >= start_point) && (t < start_point + green_time_imposed + amber(1))
  if t < start_point + green_time_imposed + yellow(1)/2
    indicator = 'green or first half yellow'
    if queue_green > 0 % there already exists queue that cannot be discharged
      % during this green period
      queue_green = queue_green + 1
    elseif queue_green == 0 % deal with the situation that no queue exists
      if (start_point + green_time_imposed - t) < green_time_available
        green_time_available = start_point + green_time_imposed - t
      end
      green_time_available = green_time_available - turn(queue_start) *
      discharge_rate_green
      if green_time_available < 0 % queue produced
        queue_green = queue_green + 1
      elseif green_time_available >= 0
        discharged_green = discharged_green + 1
        total_discharged = total_discharged + 1
        queue_start = queue_start + 1
      end
    end % if queue_green > 0
  elseif t >= start_point + green_time_imposed + yellow(1)/2
    indicator = 'second half yellow or red'
    queue_red = queue_red + 1
  end % indicator is 'green' or 'red'
  if pointer >= length(arrival)
    break
  end
  pointer = pointer + 1
  t = arrival(pointer)
end % while
queue_formed = queue_green + queue_red % queue formed in this cycle
queue_last_cycle = queue_formed
start_point = start_point + green_time_imposed + amber(1)
queue_out = arrival(queue_start:pointer-1)
turn_out = turn(queue_start:pointer-1)
end % for cycle
case 3 % opposing serving EV approach
  % get phase timing for transiting from approach 1
  fid = fopen('trans_1.dat','r') % 1 for standard, 2 for error

```

```

if fid == -1 % no such file
    disp(message)
    return
end
for instant = 1:instant_entry
    cycle_needed = fscanf(fid,'%d', 1)
    if cycle_needed == 0
        continue
    end
    green_adj = fscanf(fid,'%f',cycle_needed)
    red_adj = fscanf(fid,'%f',cycle_needed)
end
status = fclose(fid)
if status == -1 % =0 if successful closed
    disp('error')
    return
end
if cycle_needed == 0
    queue_formed = queue % queue from t_fixed
    queue_out = arrival(queue_start:pointer-1)
    turn_out = turn(queue_start:pointer-1)
    return
end
queue_last_cycle = queue % queue from t_fixed
t_fin = entry_point + t_fixed % finishing point of EV operation, t_fin.
t_fin_quotient = fix(t_fin / cycle_length)
t_fin_elapsed = t_fin - cycle_length*t_fin_quotient
% away from the beginning of green
t_dif = green(1) - t_fin_elapsed
% time difference between the finishing point and the end of green interval of
% background cycle
if abs(t_dif) < 0.1
    indicator = 'no need to adjust, get back to normal operation immediately.'
    queue_out = arrival(queue_start:pointer-1) % queue from t_fixed
    turn_out = turn(queue_start:pointer-1)
    return
end

for k = 1: cycle_needed
    if green_adj(k) == 0
        green_time(k) = 0
    else
        green_time(k) = green(1) + green_adj(k)
    end
    if red_adj(k) == 0

```

```

        red_time(k) = 0
    else
        red_time(k) = green(2) + red_adj(k)
    end
end
start_point = (cycle_entry - 1) * cycle_length + entry_point + t_fixed
for cycle = 1:cycle_needed
    queue_green = queue_last_cycle % queue from transition 1
    if green_time(cycle)==0 && (red_adj(cycle) < 0 || red_adj(cycle) == 0)
        % falls in red interval
        % red_adj(cycle) < 0: in first half red; red_adj(cycle) == 0: in second half red
        % red_adj(cycle) > 0: in second half green
        amber(1) = 0
        green_time_available = 0
    else % falls in green interval
        amber(1) = yellow(1) + allred(1)
        green_time_available = green_time(cycle) + yellow_equival_green
    end
    if (red_time(cycle) == 0) && (cycle == cycle_needed)
        % in the last cycle and no adjustment to red
        red_time_imposed = green(2) + yellow(2) + allred(2)
    else
        red_time_imposed = red_time(cycle) + yellow(2) + allred(2)
    end
    queue_red = 0
    discharged_green = 0
    total_discharged = 0
    if (queue_green > 0) && (green_time_available > 0)
        % processing queue from transition 1
        for k = 1:queue_green
            green_time_available = green_time_available - turn(queue_start) *
            discharge_rate_green
            if green_time_available < 0
                break
            end
            discharged_green = k
        end % for
        total_discharged = total_discharged + discharged_green
        queue_start = queue_start + discharged_green
        queue_green = queue_green - discharged_green
    end % if queue_green > 0
    while (t >= start_point) &&
    (t < start_point + green_time(cycle) + amber(1) + red_time_imposed)
        if t < start_point + green_time(cycle) + amber(1)
            indicator = 'green or first half yellow'

```



```

if queue_green > 0 % there already exists queue that cannot be discharged
    % during this green period
    queue_green = queue_green + 1
elseif queue_green == 0 % deal with the situation that no queue exists
    if (start_point + green_time(cycle) - t) < green_time_available
        green_time_available = start_point + green_time(cycle) - t
    end
    green_time_available = green_time_available - turn(queue_start) *
    discharge_rate_green
    if green_time_available < 0 % queue produced
        queue_green = queue_green + 1
    elseif green_time_available >= 0
        discharged_green = discharged_green + 1
        total_discharged = total_discharged + 1
        queue_start = queue_start + 1
    end
end % if queue_green > 0
elseif t >= start_point + green_time(cycle) + amber(1)
    indicator = 'second half yellow or red'
    queue_red = queue_red + 1
end % indicator is 'green' or 'red'
if pointer >= length(arrival)
    break
end
pointer = pointer + 1
t = arrival(pointer) - offset
end % while
queue_formed = queue_green + queue_red % queue formed in this cycle
queue_last_cycle = queue_formed
start_point = start_point + green_time(cycle) + amber(1) + red_time_imposed
queue_out = arrival(queue_start:pointer-1)
turn_out = turn(queue_start:pointer-1)
end % for cycle
end % switch

```

H.5 Optimal control for transition 2

```

% open a file to record results
[fid,message] = fopen('queue_length.dat', 'w');
status = fclose(fid);
if status == -1 % =0 if successful closed
    disp('error')
    return
end
for instant_entry=1:4
    for cycle_entry=5:10
        [queue_length]=optimal_route(cycle_entry,instant_entry);
    end
end

function [queue_length]=smooth_route(cycle_entry,instant_entry);
% optimal control for coordinated route
t_fixed=50 % 50-second fixed held time for transition 1
offset_1 = 40 % between intersection 1 and 2
offset_2 = 25 % between intersection 3 and 3
offset_3 = 30 % between intersection 3 and 4

% control for intersection 1
intersection# = 1
offset = 0
arrival_1 = dlmread('arrival_1e.dat')
% through arrivals at an intersection from eastbound approach to the route
arrival_2 = dlmread('arrival_1n.dat') % right turn to the route
arrival_3 = dlmread('arrival_1s.dat') % left turn to the route
arrival_4 = dlmread('arrival_1w.dat') % through vehicles from westbound approach
turn_1 = dlmread('turn_1e.dat') % turn factor for eastbound through vehicles
turn_2 = dlmread('turn_1n.dat') % right turn
turn_3 = dlmread('turn_1s.dat') % left turn
turn_4 = dlmread('turn_1w.dat') % through vehicles from westbound approach
[queue_length]= optimal_intersection(intersection#,t_fixed,cycle_entry,instant_entry,
offset,arrival_1,turn_1,arrival_2,turn_2,arrival_3,turn_3,arrival_4,turn_4);

% control for intersection 2
intersection# = 2
offset = offset_1
arrival_1 = dlmread('arrival_2e.dat')
% through arrivals at an intersection from eastbound approach to the route
arrival_2 = dlmread('arrival_2n.dat') % right turn to the route

```

```

arrival_3 = dlmread('arrival_2s.dat') % left turn to the route
arrival_4 = dlmread('arrival_2w.dat') % through vehicles from westbound approach
turn_1 = dlmread('turn_2e.dat') % turn factor for eastbound through vehicles
turn_2 = dlmread('turn_2n.dat') % right turn
turn_3 = dlmread('turn_2s.dat') % left turn
turn_4 = dlmread('turn_2w.dat') % through vehicles from westbound approach
[queue_length]= optimal_intersection(intersection#,t_fixed,cycle_entry,instant_entry,
offset,arrival_1,turn_1,arrival_2,turn_2,arrival_3,turn_3,arrival_4,turn_4);

% control for intersection 3
intersection# = 3
offset = offset_1 + offset_2
arrival_1 = dlmread('arrival_3e.dat')
% through arrivals at an intersection from eastbound approach to the route
arrival_2 = dlmread('arrival_3n.dat') % right turn to the route
arrival_3 = dlmread('arrival_3s.dat') % left turn to the route
arrival_4 = dlmread('arrival_3w.dat') % through vehicles from westbound approach
turn_1 = dlmread('turn_3e.dat') % turn factor for eastbound through vehicles
turn_2 = dlmread('turn_3n.dat') % right turn
turn_3 = dlmread('turn_3s.dat') % left turn
turn_4 = dlmread('turn_3w.dat') % through vehicles from westbound approach
[queue_length]= optimal_intersection(intersection#,t_fixed,cycle_entry,instant_entry,
offset,arrival_1,turn_1,arrival_2,turn_2,arrival_3,turn_3,arrival_4,turn_4);

% control for intersection 4
intersection# = 4
offset = offset_1 + offset_2 + offset_3
arrival_1 = dlmread('arrival_4e.dat')
% through arrivals at an intersection from eastbound approach to the route
arrival_2 = dlmread('arrival_4n.dat') % right turn to the route
arrival_3 = dlmread('arrival_4s.dat') % left turn to the route
arrival_4 = dlmread('arrival_4w.dat') % through vehicles from westbound approach
turn_1 = dlmread('turn_4e.dat') % turn factor for eastbound through vehicles
turn_2 = dlmread('turn_4n.dat') % right turn
turn_3 = dlmread('turn_4s.dat') % left turn
turn_4 = dlmread('turn_4w.dat') % through vehicles from westbound approach
[queue_length]= optimal_intersection(intersection#,t_fixed,cycle_entry,instant_entry,
offset,arrival_1,turn_1,arrival_2,turn_2,arrival_3,turn_3,arrival_4,turn_4);

%-----
function [queue_length] = optimal_intersection(intersection#,t_fixed,cycle_entry,
instant_entry,offset,arrival_1,turn_1,arrival_2,turn_2,arrival_3,turn_3,arrival_4,turn_4);
% generate queue length on approach 1-4 under optimal control for transition 2
% intersection# = 1 to 4, number of subject intersection
% t_fixed= fixed held time for EV operation

```

```

% cycle_entry=the number of cycle in which EV arrives
% instant_entry=the time instant at which EV arrives, 1-4 for at 0,green(1)/2,green(1),
%           and green(1)+yellow(1)+allred(1)+green(2)/2, respectively.
% offset=offset relative to the beginning of green for the 1st intersection
% arrival_# and turn_#=vehicle arrivals on approach # and turn factors

% using level_1,level_2,level_3,level_4 one-dimensional array as the data structure for
% control sequence
global level_1 level_2 level_3 level_4 green yellow allred a_matrix b_vector unknown
global green_max green_min
global pp_matrix wx_matrix wuu_matrix r_matrix s_matrix sg_matrix sy_matrix
global volume a_1 a_2 a_3 a_4
switch intersection# % input mean queues at intersection# under normal operation
    case 1
        queue_mean = [11.45, 28.7, 7.8, 11.05]
    case 2
        queue_mean = [15.95, 0.3, 5.95, 10.65]
    case 3
        queue_mean = [11.8, 5.35, 0.7, 10.3]
    case 4
        queue_mean = [14.45, 8.35, 18.15, 10.35]
end % switch
green = [60,40]
yellow = [3,3]
allred = [1,1]
cycle_length = green(1) + yellow(1) + allred(1) + green(2) + yellow(2) + allred(2);
% generate queue_in under fixed held time control t_fixed
on_route = 1 % serving EV approach
turn_flag = 1 % through
offset_1 = offset
[queue_out_1, turn_out_1, undischarged_1, time_spare_out_1] =
fixed_time_approach(on_route, turn_flag, offset_1, arrival_1, turn_1, cycle_entry,
instant_entry, t_fixed, green, yellow, allred)

on_route = 2
turn_flag = 2 % right turn
offset_2 = offset_1 + 64 %green(1)+yellow(1)+allred(1)
green = [40,60]
[queue_out_2, turn_out_2, undischarged_2, time_spare_out_2] =
fixed_time_approach(on_route, turn_flag, offset_2, arrival_2, turn_2, cycle_entry,
instant_entry, t_fixed, green, yellow, allred)

on_route = 2
turn_flag = 3 % left turn
offset_3 = offset_1 + 64

```

```

green = [40,60]
[queue_out_3, turn_out_3,undischarged_3,time_spare_out_3] =
fixed_time_approach(on_route,turn_flag,offset_3,arrival_3,turn_3,cycle_entry,
instant_entry,t_fixed,green,yellow,allred)

on_route = 3 % opposing approach
turn_flag = 1 % through
offset_4 = 0
green = [60,40]
[queue_out_4, turn_out_4,undischarged_4,time_spare_out_4] =
fixed_time_approach(on_route,turn_flag,offset_4,arrival_4,turn_4,cycle_entry,
instant_entry,t_fixed,green,yellow,allred)

% form queue length at the intersection
queue_length = []
queue_length(1) = length(queue_out_1)
queue_length(2) = length(queue_out_2)
queue_length(3) = length(queue_out_3)
queue_length(4) = length(queue_out_4)

queue_in = queue_length
% get initial queue on each approach corresponding to input cycle_entry and
instant_entry

% begin optimal control
green = [60,40]
% 1 Select  $P=P'=R=I(4 \times 4 \text{ identity})$  and  $S=I(2 \times 2 \text{ identity})$ 
p_matrix = eye(4);
pp_matrix = eye(4); % pp_matrix stands for p' matrix
r_matrix = eye(4);
s_matrix = eye(2);
green_max = [100,100];
green_min = [7,10];
saturation(1:4) = 2000 / 3600; % in veh/sec
for k=1:4
    discharge_rate_green(k) = 1 / saturation(k);
    % for straight-through vehicles during green in sec/veh
end

discharge_rate_yellow = 2 * discharge_rate_green;
% for straight-through vehicles during yellow in sec/veh
yellow_equival_green = yellow(1)/2;
% transfer yellow interval to equivalent green one
cycle_number = 20
tt = cycle_length * cycle_number; % total simulation duration in sec.

```

```

wu_matrix = green;
wuu_matrix = [green(1),0; 0,green(2); green(1),0; 0,green(2)];
% muu_matrix stands for mu'_matrix
wx_matrix = eye(4);
for k=1:4
    wx_matrix(k,k) = wx_matrix(k,k)/queue_mean(k);
end
sg_matrix = eye(4);
for k=1:4
    sg_matrix(k,k) = saturation(k);
end
sy_matrix = sg_matrix/2;
green_min = [7,10];
max_min = green_min(1)
if max_min < green(2)
    max_min = green_min(2)
end
t_step_max = 2 * max_min + yellow(1) + yellow(2) + allred(1) + allred(2)
t_step_min = green_min(1) + green_min(2) + yellow(1) + yellow(2) + allred(1) +
allred(2)
switch instant_entry
    case 1
        entry_point = 0 % time point away from the beginning of green at which EV
enters
    case 2
        entry_point = green(1) / 2
    case 3
        entry_point = green(1)
    case 4
        entry_point = green(1) + yellow(1) + allred(1) + green(2)/2
        % only this case in red period
end % switch
t = entry_point + t_fixed
t_quotient = fix(t / cycle_length)
t_elapsed = t - cycle_length * t_quotient
% time elapsed from the beginning of green interval
rnc = cycle_length - t_elapsed % Remained current Normal Cycle length
if rnc < t_step_min
    rnc = rnc + cycle_length
end
step_needed = round(2* rnc/(t_step_max + t_step_min))
t_step = rnc / step_needed % in second
% If queue lengths are still too long to get back to normal operation,
% another normal cycle length will be used for recovery.
% This procedure can proceed until recovery is completed.

```

```

% 5 Generate equivalent through vehicle arrivals, A_1/2/3/4(i), for each time step
start_point = cycle_entry * cycle_length - rmc
a_1(step_needed) = 0; % arrivals on approach 1 during each time step
a_2(step_needed) = 0;
a_3(step_needed) = 0;
a_4(step_needed) = 0;
a_1 = arrival_produce(arrival_1,turn_1,start_point,step_needed,t_step);
a_2 = arrival_produce(arrival_2,turn_2,start_point,step_needed,t_step);
a_3 = arrival_produce(arrival_3,turn_3,start_point,step_needed,t_step);
a_4 = arrival_produce(arrival_4,turn_4,start_point,step_needed,t_step);
% 6 Produce combination of delta1(i) and delta2(i) for current M-step sequences.
using
% M=3 for testing
% immediatly after transition 1, indication is in green on main road, so
option_1 = [1,2,3,4,9,10,11];
tn_1 = 0;
tn_2 = 0;
tn_3 = 0;
tn_4 = 0;
for k = 1:step_needed
    level_1 = [];
    level_2 = [];
    level_3 = [];
    level_4 = [];
    tn_1 = length(option_1);
    if k == step_needed
        % only level_1 applicable, construct data structure,level_1, for it
        tn_2 = data_structure_1(option_1);
    elseif k + 1 == step_needed
        tn_2 = data_structure_1(option_1);
        tn_3 = data_structure_2(tn_2);
    elseif k + 2 == step_needed
        tn_2 = data_structure_1(option_1);
        tn_3 = data_structure_2(tn_2);
        tn_4 = data_structure_3(tn_3);
    else % k + 2 < step_needed , set other data:
        % level_4 = [delta_1(4),delta_2(4),g_1(4),g_2(4)]
        tn_2 = data_structure_1(option_1);
        tn_3 = data_structure_2(tn_2);
        tn_4 = data_structure_3(tn_3);
        data_structure_4(t_step);
        level_4(1,1) = step_needed;
    end
    list_option(k,step_needed,tn_1,tn_2,tn_3,tn_4);
end

```

```

if k==1
    level_1(1:tn_1,13) = 0; % gc_1(k)
    level_1(1:tn_1,14) = 0; % gc_2(k)
elseif k>1
    level_1(1:tn_1,12) = operation(k-1,12) + operation(k-1,2);
    % gc_1(k)= gc_1(k-1)+g1(k-1)
    level_1(1:tn_1,13) = operation(k-1,13) + operation(k-1,3);
    % gc_2(k)= gc_2(k-1)+g2(k-1)
end
pi_optimal = 1000;
% first construct coefficient matrix, a_matrix, and right-hand side column vector,
% b_vector: a_matrix(equation#,variable#), b_vector(equation#)
if k == 1
    queue_initial = queue_in;
end
% different at each instant_entry and cycle_entry. queue formed at the end of
% transition 1
queue_desired = queue_mean;
equation_tn = 0;
a_matrix = [];
b_vector = [];
equation_tn = eq_initial(equation_tn,queue_initial);
% construct eqs for initial queues (eq 12)
list_equation_coefficient;
step_remained = step_needed - k + 1;
equation_tn = eq_final(equation_tn,queue_desired,step_remained);
% construct eqs for final desired queues (eq 13)
list_equation_coefficient;
eq_fixed_0 = equation_tn % used for reset a_matrix,b_vector after each iteration
for k_1 = 1:tn_1
    [g_1(1),g_2(1),delta_1(1),delta_2(1)] = get_data(1,k_1); % indicator=1 for
level_1
    equation_tn = equation_add(1,eq_fixed_0,g_1(1),g_2(1));
    % indicator=1 for level_1
    % construct eqs for q_1(1)=0 or g_2(1)=0
    list_equation_coefficient;
    step_n = k;
    equation_tn = eq_level(1,equation_tn,t_step,step_needed,step_n,delta_1(1),
    delta_2(1),a_1,a_2,a_3,a_4);
    % construct eqs for level_1; (indicator=1)
    list_equation_coefficient;
    eq_fixed_1 = equation_tn
    gc_1(step_n) = level_1(k_1, 13);
    gc_2(step_n) = level_1(k_1, 14);
    seq(1) = level_1(k_1,1) % sequence# for this option

```



```

if k == step_needed
    unknown = a_matrix \ b_vector';
    unknown = unknown'
    if unknown(1) < -0.1 % set g_1 = green_min(1) while g_1 is (applicable & <0)
        a_matrix(equation_tn+1,1:19) = 0;
        a_matrix(equation_tn+1,1) = 1;
        b_vector(equation_tn+1) = green_min(1);
        equation_tn = equation_tn + 1;
        unknown = a_matrix \ b_vector';
        unknown = unknown'
    end
    if unknown(2) < -0.1 % set g_2 = green_min(2) while g_2 is (applicable & <0)
        a_matrix(equation_tn+1,1:19) = 0;
        a_matrix(equation_tn+1,2) = 1;
        b_vector(equation_tn+1) = green_min(2);
        equation_tn = equation_tn + 1;
        unknown = a_matrix \ b_vector';
        unknown = unknown'
    end
    [gc_1,gc_2] = constraint_ver(1,seq,gc_1,gc_2,t_step,step_n);
    % verification of constraints
    option_entry = [];
    option_entry = k_1
    fill_back(1,option_entry);
    % fill back data structure for level 1(indicator=1)
    pi = pi_calculate(1,queue_mean);
    % calculate PI for only one level (indicator=1)
    if pi < pi_optimal % select optimal option
        pi_optimal = pi;
        level_1(k_1,10) = pi; % fill back pi into data structure
        optimal_list(1,1:21) = level_1(k_1,1:21); % for level 1
    end
    initialize(eq_fixed_0);
    continue % goto next k_1
end
tn_2 = level_1(k_1,11)
for k_2 = 1:tn_2
    pointer_2 = level_1(k_1,21) + k_2 - 1;
    % pointer to sub_option position in level_2
    [g_1(2),g_2(2),delta_1(2),delta_2(2)] = get_data(2,pointer_2);
    equation_tn = equation_add(2,eq_fixed_1,g_1(2),g_2(2));
    equation_tn = eq_level(2,equation_tn,t_step,step_needed,step_n,delta_1(2),
    delta_2(2),a_1,a_2,a_3,a_4);
    % construct eqs for level_2 (indicator=2)
    list_equation_coefficient

```

```

eq fixed 2 = equation tn
seq(2) = level_2(pointer_2,1) % sequence# for this option in level_2
if k + 1 == step_needed
    unknown = a_matrix \ b_vector';
    unknown = unknown'
    if unknown(1)<-0.1 % set g_1 = green_min(1) while g_1 is (applicable & <0)
        a_matrix(equation_tn+1,1:19) =0;
        a_matrix(equation_tn+1,1) =1;
        b_vector(equation_tn+1) =green_min(1);
        equation_tn = equation_tn +1;
        unknown = a_matrix \ b_vector';
        unknown = unknown'
    end
    if unknown(2)<-0.1 % set g_2 = green_min(2) while g_2 is (applicable & <0)
        a_matrix(equation_tn+1,1:19) =0;
        a_matrix(equation_tn+1,2) =1;
        b_vector(equation_tn+1) =green_min(2);
        equation_tn = equation_tn +1;
        unknown = a_matrix \ b_vector';
        unknown = unknown'
    end
    if unknown(19+1)<-0.1
        % set g_1 = green_min(1) while g_1 is (applicable & <0)
        a_matrix(equation_tn+1,1:19+11) =0;
        a_matrix(equation_tn+1,19+1) =1;
        b_vector(equation_tn+1) =green_min(1);
        equation_tn = equation_tn +1;
        unknown = a_matrix \ b_vector';
        unknown = unknown'
    end
    if unknown(19+2)<-0.1
        % set g_2 = green_min(2) while g_2 is (applicable & <0)
        a_matrix(equation_tn+1,1:19+11) =0;
        a_matrix(equation_tn+1,19+2) =1;
        b_vector(equation_tn+1) =green_min(2);
        equation_tn = equation_tn +1;
        unknown = a_matrix \ b_vector';
        unknown = unknown'
    end
    [gc_1,gc_2] = constraint_ver(2,seq,gc_1,gc_2,t_step,step_n);
    % verification of constraints
    option_entry = [];
    option_entry = [k_1,pointer_2];
    fill_back(2,option_entry);
    % fill back data structure for level 1&2

```

```

pi = pi_calculate(2,queue_mean);
% calculate PI for only two levels (indicator=2)
if pi < pi_optimal % select optimal option
    pi_optimal = pi;
    level_1(k_1,10) = pi; % fill back pi into data structure
    optimal_list(1,1:21) = level_1(k_1,1:21); % for level 1
    optimal_list(2,1:21) = level_2(pointer_2,1:21); % for level 2
    % no level 3: optimal_list(3,1:21) = level_3(k_3,1:21)
end
initialize(eq_fixed_1);
continue % goto next k_2
end
tn_3 = level_2(k_2,11);
for k_3 = 1:tn_3
    pointer_3 = level_2(k_2,21) + k_3 - 1;
    % pointer to sub-option position in level_3
    [g_1(3),g_2(3),delta_1(3),delta_2(3)] = get_data(3,pointer_3);
    equation_tn = equation_add(3,eq_fixed_2,g_1(3),g_2(3));
    equation_tn = eq_level(3,equation_tn,t_step,step_needed,step_n,delta_1(3),
    delta_2(3),a_1,a_2,a_3,a_4);
    % construct eqs for level_3 (indicator=3)
    eq_fixed_3 = equation_tn
    if k + 2 < step_needed
        g_1(4)=level_4(1,2);
        g_2(4)=level_4(1,3);
        delta_1(4)=1;
        delta_2(4)=1;
        for k_4 = (k + 3):step_needed
            % construct eqs for fixed g(1) and g(2)
            a_matrix(equation_tn,1) = 1;
            b_vector(equation_tn) = g_1(4);
            equation_tn = equation_tn + 1;
            a_matrix(equation_tn,2) = 1;
            b_vector(equation_tn) = g_2(4);
            equation_tn = equation_tn + 1;
            %construct eqs for level k_4
            equation_tn = eq_level(4,equation_tn,t_step,step_needed,step_n,
            delta_1(4),delta_2(4),a_1,a_2,a_3,a_4);
        end % for k_4 (indicator=4)
    end
end
unknown = a_matrix \ b_vector';
unknown = unknown'
if unknown(1)<-0.1 % set g_1 = green_min(1) while g_1 is (applicable & <0)
    a_matrix(equation_tn+1,1:19) = 0;
    a_matrix(equation_tn+1,1) = 1;

```

```

    b_vector(equation_tn+1)=green_min(1);
    equation_tn = equation_tn +1;
    unknown = a_matrix \ b_vector';
    unknown = unknown'
end
if unknown(2)<-0.1 % set g_2 = green_min(2) while g_2 is (applicable & <0)
    a_matrix(equation_tn+1,1:19)=0;
    a_matrix(equation_tn+1,2)=1;
    b_vector(equation_tn+1)=green_min(2);
    equation_tn = equation_tn +1;
    unknown = a_matrix \ b_vector';
    unknown = unknown'
end
if unknown(19+1)<-0.1
    % set g_1 = green_min(1) while g_1 is (applicable & <0)
    a_matrix(equation_tn+1,1:19+11)=0;
    a_matrix(equation_tn+1,19+1)=1;
    b_vector(equation_tn+1)=green_min(1);
    equation_tn = equation_tn +1;
    unknown = a_matrix \ b_vector';
    unknown = unknown'
end
if unknown(19+2)<-0.1
    % set g_2 = green_min(2) while g_2 is (applicable & <0)
    a_matrix(equation_tn+1,1:19+11)=0;
    a_matrix(equation_tn+1,19+2)=1;
    b_vector(equation_tn+1)=green_min(2);
    equation_tn = equation_tn +1;
    unknown = a_matrix \ b_vector';
    unknown = unknown'
end
if unknown(19+11+1)<-0.1
    % set g_1 = green_min(1) while g_1 is (applicable & <0)
    a_matrix(equation_tn+1,1:19+11+11)=0;
    a_matrix(equation_tn+1,19+11+1)=1;
    b_vector(equation_tn+1)=green_min(1);
    equation_tn = equation_tn +1;
    unknown = a_matrix \ b_vector';
    unknown = unknown'
end
if unknown(19+11+2)<-0.1
    % set g_2 = green_min(2) while g_2 is (applicable & <0)
    a_matrix(equation_tn+1,1:19+11+11)=0;
    a_matrix(equation_tn+1,19+11+2)=1;
    b_vector(equation_tn+1)=green_min(2);

```

```

        equation_tn = equation_tn + 1;
        unknown = a_matrix \ b_vector';
        unknown = unknown'
    end
    seq(3) = level_3(pointer_3,1)
    [gc_1,gc_2] = constraint_ver(3,seq,gc_1,gc_2,t_step,step_n);
    % verification of constraints
    option_entry = [];
    option_entry = [k_1,pointer_2,pointer_3];
    fill_back(3,option_entry);
    % fill back data structure for level 3
    pi = pi_calculate(3,queue_mean);
    % calculate PI for 3 levels (indicator=3)
    if pi < pi_optimal % select optimal option
        pi_optimal = pi;
        level_1(k_1,10) = pi; % fill back pi into data structure
        optimal_list(1,1:21) = level_1(k_1,1:21); % for level 1
        optimal_list(2,1:21) = level_2(k_2,1:21); % for level 2
        optimal_list(3,1:21) = level_3(k_3,1:21); % for level 3
    end
    initialize(eq_fixed_2);
end % for k_3
initialize(eq_fixed_1);
end % for k_2
initialize(eq_fixed_0);
end % for k_1
% change queue_initial
queue_initial(1:4) = [];
queue_initial(1:4) = optimal_list(1,4:7)
% change option_1
option_1 = [];
sub_tn = optimal_list(1,11); % should ==: tn_1 = length(option_1)
pointer = optimal_list(1,21);
for kk = 1:sub_tn
    option_1(kk) = level_2(pointer,1);
    pointer = pointer + 1;
end
operation(k,1:21) = optimal_list(1,1:21) % the final operation adopted for step k
% clear level_1/2/3/4 or use
end % for k = 1:step_needed
queue_length = 0
for k = 1:4
    queue_length(k) = operation(step_needed, 3+k) % queue at the end of transition 2
end
list_queue(instant_entry,cycle_entry,queue_length)

```

```

[fid,message] = fopen('queue_length.dat', 'a'); % 1 for standard, 2 for error
fprintf(fid,'instant_entry = %-3d ',instant_entry);
fprintf(fid,' cycle_entry = %-3d \n',cycle_entry);
fprintf(fid,' initial queue length on:\n');
for kk = 1:4
    fprintf(fid,' approach %-3d',kk);
    fprintf(fid,'= %-6.2g ',queue_in(kk));
end
fprintf(fid,' \n');
fprintf(fid,' final queue length on:\n');
for kk = 1:4
    fprintf(fid,' approach %-3d',kk);
    fprintf(fid,'= %-6.2g ',queue_length(kk));
end
fprintf(fid,' \n');
fprintf(fid,' vehicle arrivals during each time_step on:\n');
fprintf(fid,' approach 1 = ');
for kk = 1:step_needed
    fprintf(fid,'%6.2g ',a_1(kk));
end
fprintf(fid,' \n');
fprintf(fid,' approach 2 = ');
for kk = 1:step_needed
    fprintf(fid,'%6.2g ',a_2(kk));
end
fprintf(fid,' \n');
fprintf(fid,' approach 3 = ');
for kk = 1:step_needed
    fprintf(fid,'%6.2g ',a_3(kk));
end
fprintf(fid,' \n');
fprintf(fid,' approach 4 = ');
for kk = 1:step_needed
    fprintf(fid,'%6.2g ',a_4(kk));
end
fprintf(fid,' \n');
status = fclose(fid);
if status == -1 % =0 if successful closed
    disp('error')
    return
end

if (cycle_entry == 5) && (instant_entry == 1)
    fid = fopen('queue_optimal.dat','w') % 1 for standard, 2 for error
else

```

```

        fid = fopen('queue_optimal.dat','a') % append data to the end of the file
    end
    fprintf(fid,'%6.2f%6.2f%6.2f%6.2f\n',queue_length)
    status = fclose(fid)
    if status == -1
        disp(message)
        return
    end
    if (cycle_entry == 10) && (instant_entry == 4)
        fid = fopen('queue_optimal.dat','r') % 1 for standard, 2 for error
        if fid == -1 % no such file
            disp(message)
            return
        end
        queue_length = []
        queue_length = fscanf(fid,'%g %g %g %g ', [4 inf]) % app. 1,2, 3 and 4 for
each row, totally 24 columns(4 instant_entry x6cycle)
        status = fclose(fid)
        if status == -1 % =0 if successful closed
            disp('error')
            return
        end
        queue_average = mean(queue_length) % averages of columns
        standard_deviation = std(queue_length) % std of columns
    end

    % calculate average recovery time, and only time_recovery_out_1 is useful
    if (cycle_entry == 5) && (instant_entry == 1)
        fid = fopen('time_recovery.dat','w') % 1 for standard, 2 for error
    else
        fid = fopen('time_recovery.dat','a') % append data to the end of the file
    end
    fprintf(fid,'%6.2f\n',rnc)
    status = fclose(fid)
    if status == -1
        disp(message)
        return
    end
    if (cycle_entry == 10) && (instant_entry == 4)
        fid = fopen('time_recovery.dat','r') % 1 for standard, 2 for error
        if fid == -1 % no such file
            disp(message)
            return
        end
        time_recovery_vector = []

```

```

        time_recovery_vector = fscanf(fid,'%g') % app. 1 for one row, totally 24
        columns(4 instant_entry x6cycle)
        status =fclose(fid)
        if status == -1 % =0 if successful closed
            disp('error')
        return
    end
    % cycle_length = green(1) + yellow(1) + allred(1) + green(2) + yellow(2) +
    allred(2)
    % time_recovery_vector = time_recovery_vector * cycle_length
    % time_recovery_vector = time_recovery_vector
    time_recovery_average = mean(time_recovery_vector) % averages of columns
    standard_deviation = std(time_recovery_vector) % std of columns
end
% End of optimal_intersection
% All other functions used in optimal_intersection are the same as in Appendix G and
% not listed here

```


Appendix I Computer Program for Multiple EVSP Occurrences

Control

```
% set operation parameters
s_ev = 15 % operating speed of ev,m/sec
s_dis = 4.44 % discharge rate at the intersection, veh/sec
s_mean = 12 % mean speed of general traffic, m/sec
s_spacing = 8 %average spacing between successive vehicles in a queue, m
d = 960 % ev-detector position away from the intersection, m
tp_ev = d / s_ev % transition period of real time control when only s_ev is considered
tp_ev = tp_ev - 5
% minus STI=3 and startup delay=2
green = [60,40]
yellow = [3,3]
allred = [1,1]
cycle_length = green(1) + yellow(1) + allred(1) + green(2) + yellow(2) + allred(2);
green_min = [7,10];

% time headway = 16,32,64,80 seconds respectively, between successive EVs
ev_max = 3 % maximal number of EVs to be processed at a time
% input headway as required
t_headway = 15 % seconds for scenario 1, 2nd EV enters at tp/4 for 1st EV
%t_headway = 30 % for scenario 2, 2nd EV enters at tp/2 for 1st EV
% t_headway = 60 % for scenario 3, 2nd EV enters when 1st EV just crossed int.
% t_headway = 80 % for scenario 4, 2nd EV enters during rp for 1st EV
entrytime_ev(1) = 0 % relative to entry time of the 1st EV at d
entrytime_ev(2) = entrytime_ev(1) + t_headway
entrytime_ev(3) = entrytime_ev(2) + t_headway
entrytime_ev(ev_max + 1) = 1000
% to identify ev_max as the last EV to be processed in a row for computation
% convenience

% generate vehicular arrivals and turns
volume_1 = 1000 % on approach 1
volume_3 = 800 % on approach 3
volume_2 = 400 % on approach 2
volume_4 = 650 % on approach 4
[rn,arrival,turn] = vehicle_arrival(volume_1)
save arrival_1.dat arrival -ascii
save turn_1.dat turn -ascii
[rn,arrival,turn] = vehicle_arrival(volume_2)
```

```

save arrival_2.dat arrival -ascii
save turn_2.dat turn -ascii
[rn,arrival,turn] = vehicle_arrival(volume_3)
save arrival_3.dat arrival -ascii
save turn_3.dat turn -ascii
[rn,arrival,turn] = vehicle_arrival(volume_4)
save arrival_4.dat arrival -ascii
save turn_4.dat turn -ascii

for instant_entry=1:4
    switch instant_entry
        case 1
            entry_point = 0
            % time point away from the beginning of green at which EV enters
        case 2
            entry_point = green(1) / 2
        case 3
            entry_point = green(1)
        case 4
            entry_point = green(1) + yellow(1) + allred(1) + green(2)/2
            % only this case in red period
    end % switch
for cycle_entry=5:10
    % initialization
    t_n = 0
    % starting time of a period for normal operation, relative to 0 (start of simulation)
    queue_1 = zeros(1,4)
    % number of vehicles in queue immediately after normal operation
    pointer_1 = ones(1,4)
    % pointer to queue start in the array of vehicle arrivals for each approach
    i = 1 % i-th ev is processed
    ev_isolated = 1 % current ev is an isolated evsp
    ev_rp = 1 % current ev arrives at d during recovery period for optimal control
    ev_tp = 1 % current ev arrives at d during transition period for real time control
    while (i <= ev_max)
        if ev_isolated == 1
            queue_2 = zeros(1,4)
            pointer_2 = zeros(1,4)
            [queue_2,pointer_2]=normal_intersection(cycle_entry,instant_entry,t_n,
            entrytime_ev(i),queue_1,pointer_1);
            queue_qsqm = queue_2(1) % queue on approach 1
        end
        if (ev_isolated == 1) || (ev_rp == 1)
            tp = queue_qsqm * s_spacing / s_dis
            % transition period of real time control when only s_dis is considered
        end
    end
end

```

```

        if tp < tp_ev
            tp = tp_ev
        end
        tp_remaining = tp % remaining transition time for i-th ev after (i+1)th ev
        % crossed intersection
    end
    if (ev_isolated == 1) || (ev_rp == 1) || (ev_tp == 1)
        queue_3 = zeros(1,4)
        pointer_3 = zeros(1,4)
        [queue_3,pointer_3]=real_time_intersection(cycle_entry,instant_entry,
        entrytime_ev(i),tp,queue_2,pointer_2);
    end
    if (entrytime_ev(i+1) <= entrytime_ev(i) + tp)
        % (i+1)th ev between d and intersection when i-th ev just crosses intersection
        queue_3 = zeros(1,4)
        pointer_3 = zeros(1,4)
        t_elapsed = entrytime_ev(i+1) - entrytime_ev(i)
        % transition time elapsed for i-th ev when (i+1)th ev arrives at d
        [queue_3,pointer_3]=real_time_intersection_2(cycle_entry,instant_entry,
        entrytime_ev(i),tp,t_elapsed,queue_2,pointer_2);
        queue_qsqm = queue_3(1)
        % queue on approach 1 ev when (i+1)th ev arrives at d
        tp_tep = queue_qsqm * s_spacing / s_dis
        % total transition time needed for (i+1)th ev
        if tp_tep < tp_ev
            tp_tep = tp_ev
        end
        tp = tp_tep
        queue_2 = round(queue_3)
        pointer_2 = pointer_3
        i = i + 1
        ev_isolated = 0
        ev_rp = 0
        ev_tp = 1
        % current ev arrives at d during transition period for real time control
        continue % next while i
    end
    % begin to process situation when entrytime_ev(i+1) > entrytime_ev(i) + tp
    % calculate rp
    max_min = green_min(1)
    if max_min < green(2)
        max_min = green_min(2)
    end
    t_step_max = 2 * max_min + yellow(1) + yellow(2) + allred(1) + allred(2)
    t_step_min = green_min(1) + green_min(2) + yellow(1) + yellow(2) +

```

```

allred(1) + allred(2)
t = entry_point + tp_ev
t_quotient = fix(t / cycle_length)
t_elapsed = t - cycle_length * t_quotient
% time elapsed from the beginning of this green interval
rnc = cycle_length - t_elapsed % Remaining current Normal Cycle length
rp = rnc
if rp < t_step_min
    rp = rp + cycle_length
end
if (entrytime_ev(i+1) <= entrytime_ev(i) + tp + rp)
    % (i+1)th ev arrives at d during recovery period for i-th ev
    % recovery period=[entrytime_ev(i)+tp:entrytime_ev(i+1)]
    queue_4 = zeros(1,4)
    pointer_4 = zeros(1,4)
    time_end = entrytime_ev(i+1)
    flag = 0 % no cycle_length used for extension of recovery period
    [queue_4,pointer_4]=optimal_intersection(flag,cycle_entry,instant_entry,
    queue_3,pointer_3,entrytime_ev(i),entrytime_ev(i+1),tp);
    queue_qsqm = round(queue_4(1))
    % queue on approach 1 when (i+1)th ev arrives at d
    queue_2 = round(queue_4)
    pointer_2 = pointer_4
    i = i + 1
    ev_isolated = 0
    ev_rp = 1 % current ev arrives at d during recovery period for optimal control
    ev_tp = 1
    % current ev arrives at d during transition period for real time control
    continue % next while i
end
% begin to process situation of isolated evsp
% recovery period=[entrytime_ev(i)+tp:entrytime_ev(i)+tp+rp]
time_end = entrytime_ev(i) + tp + rnc
queue_4 = zeros(1,4)
pointer_4 = zeros(1,4)
flag = 1 % one cycle_length may be used for extension of recovery period
[queue_4,pointer_4]=optimal_intersection(flag,cycle_entry,instant_entry,
queue_3,pointer_3,entrytime_ev(i),time_end,tp);
if i==ev_max
    break % break while
end
t_n = entry_point + cycle_entry * cycle_length
% absolute value of entry time of ev(i) relative to 0
t_n = t_n + entrytime_ev(i) + tp + rp
% absolute value of end time of ev(i) relative to 0

```

```

    queue_1 = round(queue_4)
    pointer_1 = pointer_4
    i = i + 1
    ev_isolated = 1 % current ev is an isolated evsp
    ev_rp = 1 % current ev arrives at d during recovery period for optimal control
    ev_tp = 1 % current ev arrives at d during transition period for real time control
    % begin next normal operation
end % while i
% form queue length at the intersection
queue_length = []
queue_length(1) = queue_4(1)
queue_length(2) = queue_4(2)
queue_length(3) = queue_4(3)
queue_length(4) = queue_4(4)
if (cycle_entry == 5) && (instant_entry == 1)
    fid = fopen('queue_final.dat','w') % 1 for standard, 2 for error
else
    fid = fopen('queue_final.dat','a') % append data to the end of the file
end
fprintf(fid,'%6.2f%6.2f%6.2f%6.2f\n',queue_length)
status = fclose(fid)
if status == -1
    disp(message)
    return
end
end % cycle_entry
end % instant_entry

fid = fopen('queue_final.dat','r') % 1 for standard, 2 for error
if fid == -1 % no such file
    disp(message)
    return
end
queue_length = []
queue_length = fscanf(fid,'%g %g %g %g ', [4 inf])
% app. 1,2, 3 and 4 for each row, totally 24 columns(4 instant_entry x6cycle)
status = fclose(fid)
if status == -1 % =0 if successful closed
    disp('error')
    return
end
% calculate average for this scenario
queue_average = mean(queue_length) % averages of columns
standard_deviation = std(queue_length) % std of columns

```

```

%-----end of main program
%-----following are functions used in main program

function [queue_out,pointer_out] =
normal_intersection(cycle_entry,instant_entry,t_n,entrytime_ev,queue_in,pointer_in);
% generate queue length at an intersection for normal operation, from t_n to
% entrytime_ev+cycle_entry*cycle_length+entry_point
% queue_out[1:4]= queue formed on approach 1 to 4
% pointer_out[1:4]=pointer to head of queue on approach 1 to 4
% cycle_entry= cycle # EV enters the system
% instant_entry= 1 for EV enters at g1=0, 2 for g1/2, 3 for g1, and 4 for g2/2
% t_n=absolute start time of normal operation, relative to beginning of simulation (0)
% entrytime_ev=relative arriving time of ev, relative to cycle_entry and instant_entry
% time

% Using detector distance L=960 m
detector_distance = 960

arrival_1 = dlmread('arrival_1.dat')
% through arrivals at an intersection from eastbound approach to the route
arrival_4 = dlmread('arrival_4.dat') % right turn to the route
arrival_2 = dlmread('arrival_2.dat') % left turn to the route
arrival_3 = dlmread('arrival_3.dat') % through vehicles from westbound approach
turn_1 = dlmread('turn_1.dat') % turn factor for eastbound through vehicles
turn_4 = dlmread('turn_4.dat') % right turn
turn_2 = dlmread('turn_2.dat') % left turn
turn_3 = dlmread('turn_3.dat') % through vehicles from westbound approach
green = [60,40]
yellow = [3,3]
allred = [1,1]

on_route = 1 % serving EV approach
turn_flag = 1 % through
offset_1 = 0
[queue_out_1, pointer_out_1] =
normal_approach(on_route,turn_flag,offset_1, arrival_1,turn_1,detector_distance,
cycle_entry,instant_entry,green,yellow,allred,t_n,entrytime_ev,queue_in(1),pointer_in(
1))

on_route = 2 % cross road
turn_flag = 2 % right turn
offset_4 = offset_1 + 64 %green(1)+yellow(1)+allred(1)
green = [40,60]

```

```
[queue_out_4, pointer_out_4] =
normal_approach(on_route,turn_flag,offset_4,arrival_4,turn_4,detector_distance,
cycle_entry,instant_entry,green,yellow,allred,t_n,entrytime_ev,queue_in(4),pointer_in(
4))
```

```
on_route = 2 % cross road
turn_flag = 3 % left turn
offset_2= offset_1 + 64
green = [40,60]
[queue_out_2, pointer_out_2] =
normal_approach(on_route,turn_flag,offset_2,arrival_2,turn_2,detector_distance,
cycle_entry,instant_entry,green,yellow,allred,t_n,entrytime_ev,queue_in(2),pointer_in(
2))
```

```
on_route = 3 % opposing approach
turn_flag = 1 % through
offset_3 = 0
green = [60,40]
[queue_out_3, pointer_out_3] =
normal_approach(on_route,turn_flag,offset_3,arrival_3,turn_3,detector_distance,
cycle_entry,instant_entry,green,yellow,allred,t_n,entrytime_ev,queue_in(3),pointer_in(
3))
```

```
% form queue length at the intersection
queue_out = []
queue_out(1) = queue_out_1
queue_out(2) = queue_out_2
queue_out(3) = queue_out_3
queue_out(4) = queue_out_4
pointer_out = []
pointer_out(1) = pointer_out_1
pointer_out(2) = pointer_out_2
pointer_out(3) = pointer_out_3
pointer_out(4) = pointer_out_4
```

```
%-----
function [queue_out, pointer_out] =
normal_approach(on_route,turn_flag,offset,arrival,turn,detector_distance,cycle_entry,
instant_entry,green,yellow,allred,t_n,entrytime_ev,queue_in,pointer_in)
% generate queue length on an approach for normal operation
% queue_out, pointer_out= queue formed on the approach and pointer to queue head
% on_route= 1 for approach serving EV; 2 for crossing road, and 3 for opposing
% approach to serving EV one
% turn_flag= 1 for through, 2 for right turn, and 3 for left turn
% arrival= arrival time for each vehicle relative to the beginning of 0 sec
```

```

% turn= turning factor corresponding to each arrival with straight-through (1.0),
% left-turn (1.4), or right-turn (1.2)
% offset= beginning of green of an approach relative to the through approach
% detector_distance= distance away from subject intersection (m)

cycle_length = green(1) + yellow(1) + allred(1) + green(2) + yellow(2) + allred(2)
saturation = 2000 / 3600 % in veh/sec
discharge_rate_green = 1 / saturation
% for straight-through vehicles during green in sec/veh
discharge_rate_yellow = 2 * discharge_rate_green
% for straight-through vehicles during yellow in sec/veh
yellow_equival_green = yellow(1) * discharge_rate_green / discharge_rate_yellow

t_quotient = fix(t_n / cycle_length)
cycle_begin = 1 + t_quotient % start cycle # for normal operation,
% normal operation always begins from the beginning of a cycle
% instant_entry: 1 for g1=0, 2 for g1/2, 3 for g1, or 4 for g1+y1+allred1+g2/2
if (on_route == 1) || (on_route == 3)
    switch instant_entry
        case 1
            entry_point = 0
            % time point away from the beginning of green at which EV enters
        case 2
            entry_point = green(1) / 2
        case 3
            entry_point = green(1)
        case 4
            entry_point = green(1) + yellow(1) + allred(1) + green(2)/2
            % only this case in red period
    end % switch
elseif (on_route == 2)
    switch instant_entry
        case 1
            entry_point = green(1) + yellow(1) + allred(1) % in red period
        case 2
            entry_point = green(1) + yellow(1) + allred(1) + green(2) / 2
        case 3
            entry_point = green(1) + yellow(1) + allred(1) + green(2)
        case 4
            entry_point = green(1) / 2 % only this case in green period
    end % switch
end
entry_point = entry_point + entrytime_ev % relative to beginning of green
t_quotient = fix(entry_point / cycle_length)
cycle_entry = cycle_entry + t_quotient % actual entry cycle #

```



```

entry_point = entry_point - cycle_length * t_quotient
% time elapsed from the beginning of green interval

queue_length = zeros(1,cycle_entry)
total_produced = zeros(1,cycle_entry)
total_discharged = zeros(1,cycle_entry)
queue_last_cycle = queue_in
pointer = pointer_in % pointer to queue head
t = arrival(pointer) % arrival time in sec, and the possibly earliest one arrives at 0 sec
if (on_route == 1) || (on_route == 3)
    t = t - offset
end
queue_start = pointer
for cycle = cycle_begin:cycle_entry
    if t >= cycle * cycle_length % wait to clear in next cycle
        continue
    end
    if (cycle == cycle_entry) && (instant_entry == 1)
        if (on_route == 1) || (on_route == 3)
            % Ev is coming at the beginning of green and no need to calculate other cycles
            break
        end
    end
    queue_green = queue_last_cycle
    queue_red = 0
    discharged_green = 0
    green_time_available = green(1) + yellow_equival_green + 0.01
    % for counting calculation error

    if queue_green > 0 % processing queue from last cycle
        for k = 1:queue_green
            green_time_available = green_time_available - turn(queue_start) *
            discharge_rate_green
            if green_time_available < 0
                break
            end
            discharged_green = k
        end % for
        total_discharged(cycle) = total_discharged(cycle) + discharged_green
        queue_start = queue_start + discharged_green
        queue_green = queue_green - discharged_green
    end % if queue_green > 0

    while t < cycle * cycle_length % processing arrivals during this cycle
        total_produced(cycle) = total_produced(cycle) + 1
    end
end

```

```

t_quotient = fix(t / cycle_length)
t_elapsed = t - cycle_length * t_quotient
% time elapsed from the beginning of green interval
if (cycle == cycle_entry) && (t_elapsed >= entry_point)
    % EV is coming and no need to calculate other cycles
    break
end
if (t_elapsed >= 0) && (t_elapsed < green(1) + yellow_equival_green)
    indicator = 'green or first half yellow'
    if queue_green > 0 % there already exists queue that cannot be discharged
        % during this green period
        queue_green = queue_green + 1
    elseif queue_green == 0 % deal with the situation that no queue exists
        if (green(1) - t_elapsed) < green_time_available
            green_time_available = green(1) - t_elapsed
        end
        green_time_available = green_time_available - turn(queue_start) *
        discharge_rate_green
        if green_time_available < 0 % queue produced
            queue_green = queue_green + 1
        elseif green_time_available >= 0
            discharged_green = discharged_green + 1
            total_discharged(cycle) = total_discharged(cycle) + 1
            queue_start = queue_start + 1
        end
    end % if queue_green > 0
elseif (t_elapsed >= green(1) + yellow_equival_green)
    indicator = 'second half yellow or red'
    queue_red = queue_red + 1
end % indicator is 'green' or 'red'
if pointer >= length(arrival)
    break
end
pointer = pointer + 1
t = arrival(pointer)
if (on_route == 1) || (on_route == 3)
    t = t - offset
end
end % while
undischarged(cycle) = queue_green % undischarged during this green interval
queue_length(cycle) = queue_green + queue_red
queue_last_cycle = queue_length(cycle)
if cycle == cycle_entry % EV is coming and no need to calculate other cycles
    break
end
end

```

```

end % for

queue = queue_last_cycle % queue from normal operation
queue_out = queue_last_cycle
pointer_out = queue_start
% queue_start = pointer to the first, or will be the first vehicle in the queue

%-----
function [queue_out,pointer_out] =
real_time_intersection(cycle_entry,instant_entry,entrytime_ev,tp,queue_in,pointer_in);
% generate queue length at an intersection for real time control,from entrytime_ev to
% entrytime_ev+tp ( relative to cycle_entry*cycle_length+entry_point)
% queue_out[1:4]= queue formed on approach 1 to 4
% pointer_out[1:4]=pointer to head of queue on approach 1 to 4
% cycle_entry= cycle # EV enters the system
% instant_entry= 1 for EV enters at g1=0, 2 for g1/2, 3 for g1, and 4 for g2/2
% entrytime_ev=relative arriving time of EV (cycle_entry and instant_entry time)
% tp=transotion period for real time control

% Using detector distance L=960 m
detector_distance = 960

arrival_1 = dlmread('arrival_1.dat')
% through arrivals at an intersection from eastbound approach to the route
arrival_4 = dlmread('arrival_4.dat') % right turn to the route
arrival_2 = dlmread('arrival_2.dat') % left turn to the route
arrival_3 = dlmread('arrival_3.dat') % through vehicles from westbound approach
turn_1 = dlmread('turn_1.dat') % turn factor for eastbound through vehicles
turn_4 = dlmread('turn_4.dat') % right turn
turn_2 = dlmread('turn_2.dat') % left turn
turn_3 = dlmread('turn_3.dat') % through vehicles from westbound approach
green = [60,40]
yellow = [3,3]
allred = [1,1]

on_route = 1 % serving EV approach
turn_flag = 1 % through
offset_1 = 0
green_side_in = 0
[queue_out_1, pointer_out_1,undischarged_1,green_side_out_1,time_spare_out_1] =
real_time_approach(on_route,turn_flag,offset_1,arrival_1,turn_1,detector_distance,
green_side_in,cycle_entry,instant_entry,green,yellow,allred,entrytime_ev,tp,queue_in(1
),pointer_in(1))

on_route = 2 % cross road

```

```

turn_flag = 2 % right turn
offset_4 = offset_1 + 64 %green(1)+yellow(1)+allred(1)
green = [40,60]
green_side_in = green_side_out_1
[queue_out_4, pointer_out_4,undischarged_4,green_side_out,time_spare_out_4] =
real_time_approach(on_route,turn_flag,offset_4,arrival_4,turn_4,detector_distance,
green_side_in,cycle_entry,instant_entry,green,yellow,allred,entrytime_ev,tp,queue_in(4
),pointer_in(4))

on_route = 2 % cross road
turn_flag = 3 % left turn
offset_2= offset_1 + 64
green = [40,60]
[queue_out_2, pointer_out_2,undischarged_2,green_side_out,time_spare_out_2] =
real_time_approach(on_route,turn_flag,offset_2,arrival_2,turn_2,detector_distance,
green_side_in,cycle_entry,instant_entry,green,yellow,allred,entrytime_ev,tp,queue_in(2
),pointer_in(2))

on_route = 3 % opposing approach
turn_flag = 1 % through
offset_3 = 0
green = [60,40]
[queue_out_3, pointer_out_3,undischarged_3,green_side_out,time_spare_out_3] =
real_time_approach(on_route,turn_flag,offset_3,arrival_3,turn_3,detector_distance,
green_side_in,cycle_entry,instant_entry,green,yellow,allred,entrytime_ev,tp,queue_in(3
),pointer_in(3))

% calculate average green time available for side roads, and only green_side_out_1 is
% useful
if (cycle_entry == 5) && (instant_entry == 1)
    fid = fopen('green_side.dat','w') % 1 for standard, 2 for error
else
    fid = fopen('green_side.dat','a') % append data to the end of the file
end
fprintf(fid,'%6.2f\n',green_side_out_1)
status = fclose(fid)
if status == -1
    disp(message)
    return
end
if (cycle_entry == 10) && (instant_entry == 4)
    fid = fopen('green_side.dat','r') % 1 for standard, 2 for error
    if fid == -1 % no such file
        disp(message)
        return
    end
end

```

```

end
green_side_vector = []
green_side_vector = fscanf(fid,'%g')
% app. 1 for one row, totally 24 columns(4 instant_entry x6cycle)
status =fclose(fid)
if status == -1 % =0 if successful closed
    disp('error')
    return
end
green_side_average = mean(green_side_vector) % averages of columns
standard_deviation = std(green_side_vector) % std of columns
end

% calculate average spare time, and only time_spare_out_1 is useful
if (cycle_entry == 5) && (instant_entry == 1)
    fid = fopen('time_spare.dat','w') % 1 for standard, 2 for error
else
    fid = fopen('time_spare.dat','a') % append data to the end of the file
end
fprintf(fid,'%6.2f\n',time_spare_out_1)
status = fclose(fid)
if status == -1
    disp(message)
    return
end
if (cycle_entry == 10) && (instant_entry == 4)
    fid = fopen('time_spare.dat','r') % 1 for standard, 2 for error
    if fid == -1 % no such file
        disp(message)
        return
    end
    time_spare_vector = []
    time_spare_vector = fscanf(fid,'%g')
    % app. 1 for one row, totally 24 columns(4 instant_entry x6cycle) for each ev
    status =fclose(fid)
    if status == -1 % =0 if successful closed
        disp('error')
        return
    end
    time_spare_average = mean(time_spare_vector) % averages of columns
    standard_deviation = std(time_spare_vector) % std of columns
end

% form queue length at the intersection
queue_out = []

```

```

queue_out(1) = queue_out_1
queue_out(2) = queue_out_2
queue_out(3) = queue_out_3
queue_out(4) = queue_out_4
% queue_average = mean(queue_length) % averages of vector
% standard_deviation = std(queue_length) % std of vector
pointer_out = []
pointer_out(1) = pointer_out_1
pointer_out(2) = pointer_out_2
pointer_out(3) = pointer_out_3
pointer_out(4) = pointer_out_4

if (cycle_entry == 5) && (instant_entry == 1)
    fid = fopen('queue_realtime.dat','w') % 1 for standard, 2 for error
else
    fid = fopen('queue_realtime.dat','a') % append data to the end of the file
end
fprintf(fid,'%6.2f%6.2f%6.2f%6.2f\n',queue_out)
status = fclose(fid)
if status == -1
    disp(message)
    return
end
if (cycle_entry == 10) && (instant_entry == 4)
    fid = fopen('queue_realtime.dat','r') % 1 for standard, 2 for error
    if fid == -1 % no such file
        disp(message)
        return
    end
    queue_length = []
    queue_length = fscanf(fid,'%g %g %g %g ', [4 inf]) % app. 1,2, 3 and 4 for each
row,
    % totally 24 columns(4 instant_entry x6cycle) for each ev
    status =fclose(fid)
    if status == -1 % =0 if successful closed
        disp('error')
        return
    end
    queue_average = mean(queue_length') % averages of columns
    standard_deviation = std(queue_length') % std of columns
end

%-----
function [queue_out, pointer_out,queue_undischarged,green_side_out,time_spare_out]
= real_time_approach(on_route,turn_flag,offset,arrival,turn,detector_distance,

```

```

green_side_in,cycle_entry,instant_entry,green,yellow,allred,entrytime_ev,tp,
queue_in,pointer_in)
% generate queue length on an approach for real time control strategy,
% from entrytime_ev to entrytime_ev+tp
% ( relative to cycle_entry*cycle_length+entry_point)
% queue_out, pointer_out= queue formed on the approach and pointer to queue head
% queue_undischarged= number of queue ahead of EV undischarged when EV reaches
% the intersection, applicable only if on_route==1
% green_side_out= green time available for side roads during EV operation
% on_route= 1 for approach serving EV; 2 for crossing road, and 3 for opposing
% approach to serving EV one
% turn_flag= 1 for through, 2 for right turn, and 3 for left turn
% arrival= arrival time for each vehicle relative to the beginning of 0 sec
% turn= turning factor corresponding to each arrival with straight-through (1.0),
% left-turn (1.4), or right-turn (1.2)
% offset= beginning of green of an approach relative to the through approach
% detector_distance= distance away from subject intersection (m)
% green_side_out= green available for side roads, obtained from serving EV approach
% green_side_in= green available for side roads, used for side roads and opposing
% serving EV app.
% successful='Yes!' if EV can cross the intersection, otherwise successful='No!'

cycle_length = green(1) + yellow(1) + allred(1) + green(2) + yellow(2) + allred(2)
saturation = 2000 / 3600 % in veh/sec
discharge_rate_green = 1 / saturation
% for straight-through vehicles during green in sec/veh
discharge_rate_yellow = 2 * discharge_rate_green
% for straight-through vehicles during green in sec/veh
yellow_equival_green = yellow(1) * discharge_rate_green / discharge_rate_yellow

t_ev = tp
sti = 0
t_startup = 0
% t_ev = t_ev -5 % minus STI=3 and startup delay=2 for comparison with fixed held
% time control
t_ev = t_ev - sti - t_startup
speed_mean = 12 % General vehicle mean Smean (m/s or 43.2 Km/h)
spacing = 8 % Average spacing between successive vehicles
speed_dis = 4.44 % discharge speed Sdis= 4.44 m/s or 16 Km/h
gmin = [7,10]
gmax = [100,100]

% instant_entry: 1 for g1=0, 2 for g1/2, 3 for g1, or 4 for g1+y1+allred1+g2/2
if (on_route == 1) || (on_route == 3)
    switch instant_entry

```

```

case 1
    entry_point = 0
    % time point away from the beginning of green at which EV enters
case 2
    entry_point = green(1) / 2
case 3
    entry_point = green(1)
case 4
    entry_point = green(1) + yellow(1) + allred(1) + green(2)/2
    % only this case in red period
end % switch
elseif (on_route == 2)
    switch instant_entry
    case 1
        entry_point = green(1) + yellow(1) + allred(1) % in red period
    case 2
        entry_point = green(1) + yellow(1) + allred(1) + green(2) / 2
    case 3
        entry_point = green(1) + yellow(1) + allred(1) + green(2)
    case 4
        entry_point = green(1) / 2 % only this case in green period
    end % switch
end
entry_point = entry_point + entrytime_ev % relative to beginning of green
t_quotient = fix(entry_point / cycle_length)
cycle_entry = cycle_entry + t_quotient % actual entry cycle #
entry_point = entry_point - cycle_length * t_quotient
% time elapsed from the beginning of green interval

queue_length = zeros(1,cycle_entry)
total_produced = zeros(1,cycle_entry)
total_discharged = zeros(1,cycle_entry)
queue_last_cycle = queue_in % from normal operation
pointer = pointer_in
t = arrival(pointer) % arrival time in sec, and the possibly earliest one arrives at 0 sec
if (on_route == 1) || (on_route == 3)
    t = t - offset
end
queue_start=pointer

queue = queue_last_cycle % queue from normal operation
queue_undischarged = 0
green_side_out = 0
time_spare_out = 0

```



```

if on_route == 2 % crossing road
    t_green_side = green_side_in
    switch instant_entry % calculate amber time prior to giving green to side road
        case {1}
            t_amber_1 = 0
        case {2,3}
            t_amber_1 = yellow(2) + allred(2)
        case 4
            t_amber_1 = 0 % onle this case in red period
    end % switch
    t_amber_2 = yellow(1) + allred(1) + sti
    % calculate amber time after giving green to side road
    if t_green_side < 0.1 % == 0 for calculation error.
        % all t_ev is green for main road and red for side road.
        t_red_imposed_1 = 0 % red before getting side green
        t_red_imposed_2 = t_ev % red after getting side green
        yellow_equival_green = 0 % no yellow for side road
    else % t_green_side > 0
        t_red_imposed_1 = t_amber_1
        t_red_imposed_2 = t_ev - t_amber_1 - t_green_side
    end % if
    green_time_available = t_green_side + yellow_equival_green + 0.01

    if t_red_imposed_1 > 0 % produce queue prior to side green
        start = (cycle_entry - 1) * cycle_length + entry_point
        while (t < start + t_red_imposed_1)
            queue = queue + 1
            if pointer >= length(arrival)
                break
            end
            pointer = pointer + 1
            t = arrival(pointer)
        end % while
    end

    % processing queue from t_red_imposed_1
    discharged_green = 0
    if queue > 0
        for k = 1:queue
            green_time_available = green_time_available - turn(queue_start + k - 1) *
            discharge_rate_green
            if green_time_available < 0
                break
            end
            discharged_green = k
        end
    end

```

```

end % for
discharged_for_ev = discharged_green
queue_start = queue_start + discharged_green
queue = queue - discharged_green
end % if queue > 0

% process queue and arrivals after t_red_imposed_1
discharged_green = 0
queue_red = 0
start = (cycle_entry - 1) * cycle_length + entry_point + t_red_imposed_1
while (t < start + t_green_side + t_red_imposed_2)
    if (t < start + t_green_side + yellow_equival_green)
        indicator = 'green or first half yellow'
        if queue > 0 % there already exists queue that cannot be discharged during
            % t_green_side
            queue = queue + 1
        elseif queue == 0 % deal with the situation that no queue exists
            if (start + t_green_side - t) < green_time_available
                green_time_available = start + t_green_side - t
            end
            green_time_available = green_time_available - turn(queue_start) *
            discharge_rate_green
            if green_time_available < 0 % queue produced
                queue = queue + 1
            elseif green_time_available >= 0
                discharged_green = discharged_green + 1
                queue_start = queue_start + 1
            end
        end % if queue_green > 0
    elseif (t >= start + t_green_side + yellow_equival_green)
        indicator = 'second half yellow or red'
        queue_red = queue_red + 1
    end % indicator is 'green' or 'red'
    if pointer >= length(arrival)
        break
    end
    pointer = pointer + 1
    t = arrival(pointer)
end % while
queue_formed = queue + queue_red
queue_length(cycle_entry) = queue_formed % produced during t_fixed
turn_out = turn(queue_start:pointer-1)
queue_out = queue_formed
pointer_out = queue_start
elseif on_route == 1 % serving EV approach

```

```

discharged_green = 0
queue_equiv_length = 0
if queue == 0
    t_clear = 0
else % queue > 0
    for k = 1:queue
        queue_equiv_length = queue_equiv_length + turn(queue_start + k - 1)
    end

    t_clear_1 = queue_equiv_length * spacing / speed_mean
    % take general traffic to traverse queue length
    t_clear_2 = queue_equiv_length / saturation % time for discharging queue length
    if t_clear_1 > t_clear_2
        t_clear = t_clear_1 + t_startup
    else
        t_clear = t_clear_2 + t_startup
    end
    if t_clear > gmax(1)
        t_clear = gmax(1)
        successful = 'no, because clearance time greater than maximal time permitted.'
        turn_out = turn(queue_start:pointer-1)
        queue_out = queue
        pointer_out = queue_start
        fprintf('clearance time greater than maximal time permitted!!  press any key to
        continue \n')
        pause
    elseif t_clear < gmin(1)
        t_clear = gmin(1)
    end % if
end % if queue>0
switch instant_entry % calculate amber time prior to giving green to side road
    case {1}
        t_amber_1 = 0 % beginning of green
    case {2,3}
        t_amber_1 = yellow(1) + allred(1)
    case 4
        t_amber_1 = 0 % only this case in red period
end % switch
t_amber_2 = yellow(2) + allred(2) + sti
% calculate amber time after giving green to side road
if t_clear <= 0.01 % ==0 for calculation error
    t_green_side = t_ev - t_amber_1 - t_amber_2
else % t_clear>0
    t_green_side = t_ev - t_amber_1 - t_amber_2 - t_clear
end

```

```

if t_green_side > gmax(2)
    t_green_side = gmax(2)
elseif t_green_side < gmin(2)
    t_green_side = 0
end
green_side_out = t_green_side
if t_green_side == 0 % all t_ev is green for main road and red for side road.
    t_green_main = t_ev
    yellow_equival_green = 0
else % t_green_side > 0
    t_green_main = t_ev - t_amber_1 - t_amber_2 - t_green_side
    switch instant_entry % calculate yellow_equival_green to main road
        case {2,3}
            yellow_equival_green = ( t_amber_1 - allred(1) ) *
                discharge_rate_green / discharge_rate_yellow
        case {1,4}
            yellow_equival_green = 0
        end % switch
    end % if
    green_time_available = t_green_main + yellow_equival_green + 0.01

if queue > 0 % processing queue from normal operation
    total_green_time_needed = 0
    for k = 1:queue
        total_green_time_needed = total_green_time_needed + turn(queue_start + k - 1)
    *
        discharge_rate_green
    end % for
    for k = 1:queue
        green_time_available = green_time_available - turn(queue_start + k - 1) *
            discharge_rate_green
        if green_time_available < 0
            break
        end
        discharged_green = k
    end % for
    discharged_for_ev = discharged_green
    queue_start = queue_start + discharged_green
    queue = queue - discharged_green
end % if queue > 0

if (queue > 0)
    successful = 'No!' % more time is needed
    time_needed = total_green_time_needed - t_green_main - yellow_equival_green
    queue_undischarged = queue

```

```

fprintf('more time is needed!! press any key to continue \n')
pause
pointer_out = queue_start
queue_out = queue_undischarged+0.1 % different from successful=='yes'
turn_out = turn(queue_start:queue_start+queue-1)
elseif (queue == 0)
    successful = 'Yes!'
    time_spare = green_time_available - yellow_equival_green
    % exclude yellow_equival_green
    time_spare = round(time_spare *10)/10 % exclude yellow_equival_green
    time_spare_out = time_spare
    start = (cycle_entry - 1) * cycle_length + entry_point
    while (t <= start + t_ev) % produce affected vehicles during EV operation: t_ev
        queue = queue + 1
        if pointer >= length(arrival)
            break
        end
        pointer = pointer + 1
        t = arrival(pointer)
    end % while
    queue_length(cycle_entry) = queue % produced during t_ev
    turn_out = turn(queue_start:pointer-1)
    queue_out = queue
    pointer_out = queue_start
end %if
elseif on_route == 3 % opposing approach to serving EV one
    discharged_green = 0
    t_green_side = green_side in
    switch instant_entry % calculate amber time prior to giving green to side road
        case {1}
            t_amber_1 = 0
        case {2,3}
            t_amber_1 = yellow(1) + allred(1)
        case 4
            t_amber_1 = 0 % only this case in red period
    end % switch
    t_amber_2 = yellow(2) + allred(2) + sti
    % calculate amber time after giving green to side road
    if t_green_side < 0.1 % == 0 for calculation error % all t_ev is green for main road
        % and red for side road.
        t_red_imposed = 0
        t_green_main = t_ev
        yellow_equival_green = 0
    else % t_green_side > 0
        t_red_imposed = t_amber_1 + t_amber_2 + t_green_side
    end
end

```

```

t_green_main = t_ev - t_red_imposed
switch instant_entry % calculate yellow_equival_green to main road
case {2,3}
    yellow_equival_green = ( t_amber_1 - allred(1) ) *
        discharge_rate_green / discharge_rate_yellow
case {1,4}
    yellow_equival_green = 0
end % switch
end % if
green_time_available = t_green_main + yellow_equival_green + 0.01

if t_red_imposed > 0 % produce queue during t_red_imposed
    start = (cycle_entry - 1) * cycle_length + entry_point
    while (t <= start + t_red_imposed)
        queue = queue + 1
        if pointer >= length(arrival)
            break
        end
        pointer = pointer + 1
        t = arrival(pointer)
    end % while
end

% processing queue from t_red_imposed
if queue > 0
    for k = 1:queue
        green_time_available = green_time_available - turn(queue_start + k - 1) *
            discharge_rate_green
        if green_time_available < 0
            break
        end
        discharged_green = k
    end % for
    discharged_for_ev = discharged_green
    queue_start = queue_start + discharged_green
    queue = queue - discharged_green
end % if queue > 0
queue_formed = queue
queue_length(cycle_entry) = queue % produced at the end of t_ev period
turn_out = turn(queue_start:pointer-1)
queue_out = queue_formed
pointer_out = queue_start
end % if on_route

```

```

%-----
function [queue_out,pointer_out]=real_time_intersection_2(cycle_entry,instant_entry,
entrytime_ev,tp,t_elapsed,queue_in,pointer_in);
% run real time control for transition 1 to discharge (not clear) queue ahead of ev(i)
until
% ev(i+1) arrives at d, and then plus arrivals between ev(i) and ev(i+1) as final
% queue_out and final pointer_out, i.e., from entrytime_ev to entrytime_ev+t_elapsed
% ( relative to cycle_entry*cycle_length+entry_point)
% queue_out[1:4]= queue formed on approach 1 to 4
% pointer_out[1:4]=pointer to head of queue on approach 1 to 4
% cycle_entry= cycle # EV enters the system
% instant_entry= 1 for EV enters at g1=0, 2 for g1/2, 3 for g1, and 4 for g2/2
% entrytime_ev=relative arriving time of ev (cycle_entry and instant_entry time)
% tp=t_ev
%t_elapsed=entrytime_ev(i+1) - entrytime_ev(i)

% Using detector distance L=960 m
detector_distance = 960

arrival_1 = dlmread('arrival_1.dat')
% through arrivals at an intersection from eastbound approach to the route
arrival_4 = dlmread('arrival_4.dat') % right turn to the route
arrival_2 = dlmread('arrival_2.dat') % left turn to the route
arrival_3 = dlmread('arrival_3.dat') % through vehicles from westbound approach
turn_1 = dlmread('turn_1.dat') % turn factor for eastbound through vehicles
turn_4 = dlmread('turn_4.dat') % right turn
turn_2 = dlmread('turn_2.dat') % left turn
turn_3 = dlmread('turn_3.dat') % through vehicles from westbound approach
green = [60,40]
yellow = [3,3]
allred = [1,1]

on_route = 1 % serving EV approach
turn_flag = 1 % through
offset_1 = 0
green_side_in = 0
[queue_out_1, pointer_out_1,undischarged_1,green_side_out_1,time_spare_out_1] =
real_time_approach_2(on_route,turn_flag,offset_1,arrival_1,turn_1,detector_distance,
green_side_in,cycle_entry,instant_entry,green,yellow,allred,entrytime_ev,tp,t_elapsed,
queue_in(1),pointer_in(1))

on_route = 2 % cross road
turn_flag = 2 % right turn
offset_4 = offset_1 + 64 %green(1)+yellow(1)+allred(1)
green = [40,60]

```

```

green_side_in = green_side_out_1
[queue_out_4, pointer_out_4, undischarged_4, green_side_out, time_spare_out_4] =
real_time_approach_2(on_route, turn_flag, offset_4, arrival_4, turn_4, detector_distance,
green_side_in, cycle_entry, instant_entry, green, yellow, allred, entrytime_ev, tp, t_elapsed, q
ueue_in(4), pointer_in(4))

on_route = 2 % cross road
turn_flag = 3 % left turn
offset_2 = offset_1 + 64
green = [40, 60]
[queue_out_2, pointer_out_2, undischarged_2, green_side_out, time_spare_out_2] =
real_time_approach_2(on_route, turn_flag, offset_2, arrival_2, turn_2, detector_distance,
green_side_in, cycle_entry, instant_entry, green, yellow, allred, entrytime_ev, tp, t_elapsed,
queue_in(2), pointer_in(2))

on_route = 3 % opposing approach
turn_flag = 1 % through
offset_3 = 0
green = [60, 40]
[queue_out_3, pointer_out_3, undischarged_3, green_side_out, time_spare_out_3] =
real_time_approach_2(on_route, turn_flag, offset_3, arrival_3, turn_3, detector_distance,
green_side_in, cycle_entry, instant_entry, green, yellow, allred, entrytime_ev, tp, t_elapsed,
queue_in(3), pointer_in(3))

% form queue length at the intersection
queue_out = []
queue_out(1) = queue_out_1
queue_out(2) = queue_out_2
queue_out(3) = queue_out_3
queue_out(4) = queue_out_4
% queue_average = mean(queue_length) % averages of vector
% standard_deviation = std(queue_length) % std of vector
pointer_out = []
pointer_out(1) = pointer_out_1
pointer_out(2) = pointer_out_2
pointer_out(3) = pointer_out_3
pointer_out(4) = pointer_out_4

%-----
function [queue_out, pointer_out, queue_undischarged, green_side_out, time_spare_out]
= real_time_approach_2(on_route, turn_flag, offset, arrival, turn, detector_distance,
green_side_in, cycle_entry, instant_entry, green, yellow, allred, entrytime_ev, tp,
t_elapsed, queue_in, pointer_in)
% run real time control for transition 1 to discharge (not clear) queue ahead of ev(i)
until

```



```

% ev(i+1) arrives at d, and then plus arrivals between ev(i) and ev(i+1) as final
% queue_out and final pointer_out, i.e., from entrytime_ev to entrytime_ev+t_elapsed
% ( relative to cycle_entry*cycle_length+entry_point)
% queue_out, pointer_out= queue formed on the approach and pointer to queue head
% queue_undischarged= number of queue ahead of EV undischarged when EV reaches
% the intersection, applicable only if on_route==1
% green_side_out= green time available for side roads during EV operation
% on_route= 1 for approach serving EV; 2 for crossing road, and 3 for opposing
% approach to serving EV one
% turn_flag= 1 for through, 2 for right turn, and 3 for left turn
% arrival= arrival time for each vehicle relative to the beginning of 0 sec
% turn= turning factor corresponding to each arrival with straight-through (1.0),
% left-turn (1.4), or right-turn (1.2)
% offset= beginning of green of an approach relative to the through approach
% detector_distance= distance away from subject intersection (m)
% green_side_out= green available for side roads, obtained from serving EV approach
% green_side_in= green available for side roads, used for side roads and opposing
% serving EV app.
% successful='Yes!' if EV can cross the intersection, otherwise successful='No!'

cycle_length = green(1) + yellow(1) + allred(1) + green(2) + yellow(2) + allred(2)
saturation = 2000 / 3600 % in veh/sec
discharge_rate_green = 1 / saturation
% for straight-through vehicles during green in sec/veh
discharge_rate_yellow = 2 * discharge_rate_green
% for straight-through vehicles during yellow in sec/veh
yellow_equival_green = yellow(1) * discharge_rate_green / discharge_rate_yellow

t_ev = tp
sti = 0
t_startup = 0
t_ev = t_ev - sti - t_startup % minus STI=0 and startup delay=0 for comparison with
% fixed held time control
speed_mean = 12 % General vehicle mean Smean (m/s or 43.2 Km/h)
spacing = 8 % Average spacing between successive vehicles spacing
speed_dis = 4.44 % discharge speed Sdis= 4.44 m/s or 16 Km/h
gmin = [7,10]
gmax = [100,100]

% instant_entry: 1 for gl=0, 2 for gl/2, 3 for gl, or 4 for gl+yl+allredl+g2/2
if (on_route == 1) || (on_route == 3)
    switch instant_entry
        case 1
            entry_point = 0
            % time point away from the beginning of green at which EV enters

```

```

case 2
    entry_point = green(1) / 2
case 3
    entry_point = green(1)
case 4
    entry_point = green(1) + yellow(1) + allred(1) + green(2)/2
    % only this case in red period
end % switch
elseif (on_route == 2)
    switch instant_entry
    case 1
        entry_point = green(1) + yellow(1) + allred(1) % in red period
    case 2
        entry_point = green(1) + yellow(1) + allred(1) + green(2) / 2
    case 3
        entry_point = green(1) + yellow(1) + allred(1) + green(2)
    case 4
        entry_point = green(1) / 2 % only this case in green period
    end % switch
end
entry_point = entry_point + entrytime_ev % relative to beginning of green
t_quotient = fix(entry_point / cycle_length)
cycle_entry = cycle_entry + t_quotient % actual entry cycle #
entry_point = entry_point - cycle_length * t_quotient
% time elapsed from the beginning of green interval

queue_length = zeros(1,cycle_entry)
total_produced = zeros(1,cycle_entry)
total_discharged = zeros(1,cycle_entry)
queue_last_cycle = queue_in % from normal operation
pointer = pointer_in
t = arrival(pointer) % arrival time in sec, and the possibly earliest one arrives at 0 sec
if (on_route == 1) || (on_route == 3)
    t = t - offset
end
queue_start=pointer

queue = queue_last_cycle % queue from normal operation
queue_undischarged = 0
green_side_out = 0
time_spare_out = 0

if on_route == 2 % crossing road
    t_green_side = green_side_in
    switch instant_entry % calculate amber time prior to giving green to side road

```

```

case {1}
    t_amber_1 = 0
case {2,3}
    t_amber_1 = yellow(2) + allred(2)
case 4
    t_amber_1 = 0 % onle this case in red period
end % switch
t_amber_2 = yellow(1) + allred(1) + sti
% calculate amber time after giving green to side road
if t_green_side < 0.1 % == 0 for calculation error.
    % all t_ev is green for main road and red for side road.
    t_red_imposed_1 = 0 % red before getting side green
    t_red_imposed_2 = t_ev % red after getting side green
    yellow_equival_green = 0 % no yellow for side road
else % t_green_side > 0
    t_red_imposed_1 = t_amber_1
    t_red_imposed_2 = t_ev - t_amber_1 - t_green_side
end % if
green_time_available = t_green_side + yellow_equival_green + 0.01

%diferent from real_time_approach() below
if t_green_side < 0.1 % == 0 for calculation error.
    % all t_elapsed is green for main road and red for side road.
    t_red_imposed_1 = 0 % red before getting side green
    t_red_imposed_2 = t_elapsed % red after getting side green
    yellow_equival_green = 0 % no yellow for side road
    green_time_available = 0
elseif t_green_side <= t_elapsed % green return to main road before ev(i+1) at d
    t_red_imposed_1 = t_amber_1
    t_red_imposed_2 = t_elapsed - t_amber_1 - t_green_side
    % for side road until ev(i+1) at d
    green_time_available = t_green_side + yellow_equival_green + 0.01
else % t_green_side > t_elapsed % side road face green when ev(i+1) at d
    t_red_imposed_1 = t_amber_1
    t_red_imposed_2 = t_amber_2
    green_time_available = t_elapsed - t_amber_1 - t_amber_2
end % if
green_time_available_actual = green_time_available % during t_elapsed
if t_red_imposed_1 > 0 % produce queue prior to side green
    start = (cycle_entry - 1) * cycle_length + entry_point
    while (t < start + t_red_imposed_1)
        queue = queue + 1
        if pointer >= length(arrival)
            break
        end
    end
end

```

```

        pointer = pointer + 1
        t = arrival(pointer)
    end % while
end

% processing queue from t_red_imposed_1
discharged_green = 0
if queue > 0
    for k = 1:queue
        green_time_available = green_time_available - turn(queue_start + k - 1) *
            discharge_rate_green
        if green_time_available < 0
            break
        end
        discharged_green = k
    end % for
    discharged_for_ev = discharged_green
    queue_start = queue_start + discharged_green
    queue = queue - discharged_green
end % if queue > 0

% process queue and arrivals after t_red_imposed_1 and during t_elapsed
discharged_green = 0
queue_red = 0
start = (cycle_entry - 1) * cycle_length + entry_point
while (t < start + t_elapsed)
    if (t < start + green_time_available_actual)
        indicator = 'green or first half yellow'
        if queue > 0 % there already exists queue that cannot be discharged during
            % t_green_side
            queue = queue + 1
        elseif queue == 0 % deal with the situation that no queue exists
            if (start + green_time_available_actual - t) < green_time_available
                green_time_available = start + green_time_available_actual - t
            end
            green_time_available = green_time_available - turn(queue_start) *
                discharge_rate_green
            if green_time_available < 0 % queue produced
                queue = queue + 1
            elseif green_time_available >= 0
                discharged_green = discharged_green + 1
                queue_start = queue_start + 1
            end
        end % if queue_green > 0
    elseif (t >= start + green_time_available_actual)

```

```

        indicator = 'second half yellow or red'
        queue_red = queue_red + 1
    end % indicator is 'green' or 'red'
    if pointer >= length(arrival)
        break
    end
    pointer = pointer + 1
    t = arrival(pointer)
end % while
queue_formed = queue + queue_red
queue_length(cycle_entry) = queue_formed % produced during t_ev - t_elapsed
turn_out = turn(queue_start:pointer-1)
queue_out = queue_formed
pointer_out = queue_start
elseif on_route == 1 % serving EV approach
    discharged_green = 0
    queue_equiv_length = 0
    if queue == 0
        t_clear = 0
    else % queue > 0
        for k = 1:queue
            queue_equiv_length = queue_equiv_length + turn(queue_start + k - 1)
        end

        t_clear_1 = queue_equiv_length * spacing / speed_mean
        % take general traffic to traverse queue length
        t_clear_2 = queue_equiv_length / saturation % time for discharging queue length
        if t_clear_1 > t_clear_2
            t_clear = t_clear_1 + t_startup
        else
            t_clear = t_clear_2 + t_startup
        end
        if t_clear > gmax(1)
            t_clear = gmax(1)
            successful = 'no, because clearance time greater than maximal time permitted.'
            turn_out = turn(queue_start:pointer-1)
            queue_out = queue
            pointer_out = queue_start
            fprintf('press any key to continue \n')
            pause
        elseif t_clear < gmin(1)
            t_clear = gmin(1)
        end
    end % if queue>0
    switch instant_entry % calculate amber time prior to giving green to side road

```

```

case {1}
    t_amber_1 = 0 % beginning of green
case {2,3}
    t_amber_1 = yellow(1) + allred(1)
case 4
    t_amber_1 = 0 % onle this case in red period
end % switch
t_amber_2 = yellow(2) + allred(2) + sti
% calculate amber time after giving green to side road
if t_clear <= 0.01 % ==0 for calculation error
    t_green_side = t_ev - t_amber_1 - t_amber_2
else % t_clear>0
    t_green_side = t_ev - t_amber_1 - t_amber_2 - t_clear
end
if t_green_side > gmax(2)
    t_green_side = gmax(2)
elseif t_green_side < gmin(2)
    t_green_side = 0
end
green_side_out = t_green_side
if t_green_side == 0 % all t_ev is green for main road and red for side road.
    t_green_main = t_ev
    yellow_equival_green = 0
else % t_green_side > 0
    t_green_main = t_ev - t_amber_1 - t_amber_2 - t_green_side
    switch instant_entry % calculate yellow_equival_green to main road
        case {2,3}
            yellow_equival_green = ( t_amber_1 - allred(1) ) *
                discharge_rate_green / discharge_rate_yellow
        case {1,4}
            yellow_equival_green = 0
        end % switch
    end % if
    green_time_available = t_green_main + yellow_equival_green + 0.01

% ===== different from real_time_approach() below
if t_green_side < 0.01 % ==0
    green_time_avaiable = t_elapsed % for ev(i) until ev(i+) at d
elseif (t_green_side <= t_elapsed)
    % green return to ev route before ev(i+1) at d, and then no change for green time
for
    % side rode
    t_green_main = t_elapsed - t_green_side
    green_time_available = t_green_main + yellow_equival_green + 0.01
    % green_time_available for ev(i) until ev(i+1) at d

```

```

else % (t_green_side>t_elapsed)
    % all t_elapsed used as green time for side road until ev(i+1) at d, and then
    green_time_avaialble = 0 % until ev(i+1) at d
end

start = (cycle_entry - 1) * cycle_length + entry_point
while (t <= start + t_elapsed) % produce vehicles during t_elapsed
    queue = queue + 1
    if pointer >= length(arrival)
        break
    end
    pointer = pointer + 1
    t = arrival(pointer)
end % while
if queue > 0 % discharge queue untile ev(i+1) at d
    % no need to check total_green_time_needed
    total_green_time_needed = 0
    for k = 1:queue
        green_time_available = green_time_available - turn(queue_start + k - 1) *
        discharge_rate_green
        if green_time_available < 0
            break
        end
        discharged_green = k
    end % for
    discharged_for_ev = discharged_green
    queue_start = queue_start + discharged_green
    queue = queue - discharged_green
end % if queue > 0

time_spare = green_time_available - yellow_equival_green
% exclude yellow_equival_green
time_spare = round(time_spare * 10)/10 % exclude yellow_equival_green
time_spare_out = time_spare
queue_length(cycle_entry) = queue % produced during t_elapsed
turn_out = turn(queue_start:pointer-1)
queue_out = queue
pointer_out = queue_start
elseif on_route == 3 % opposing approach to serving EV one
    discharged_green = 0
    t_green_side = green_side_in
    switch instant_entry % calculate amber time prior to giving green to side road
        case {1}
            t_amber_1 = 0
        case {2,3}

```

```

        t_amber_1 = yellow(1) + allred(1)
    case 4
        t_amber_1 = 0 % onle this case in red period
    end % switch
    t_amber_2 = yellow(2) + allred(2) + sti
    % calculate amber time after giving green to side road
    if t_green_side < 0.1 % == 0 for calculation error
        % all t_ev is green for main road and red for side road.
        t_red_imposed = 0
        t_green_main = t_ev
        yellow_equival_green = 0
    else % t_green_side > 0
        t_red_imposed = t_amber_1 + t_amber_2 + t_green_side
        t_green_main = t_ev - t_red_imposed
        switch instant_entry % calculate yellow_equival_green to main road
            case {2,3}
                yellow_equival_green = ( t_amber_1 - allred(1) ) * discharge_rate_green /
                    discharge_rate_yellow
            case {1,4}
                yellow_equival_green = 0
        end % switch
    end % if
    green_time_available = t_green_main + yellow_equival_green + 0.01

    % ===== different from real_time_approach() below
    if t_green_side < 0.01 % ==0
        t_red_imposed = 0
        green_time_avaialble = t_elapsed % for ev(i) until ev(i+) at d
    elseif (t_green_side <= t_elapsed)
        % no change for green time on side road, and then green for main road is
        t_red_imposed = t_amber_1 + t_amber_2 + t_green_side
        green_time_avaialble = t_elapsed - t_red_imposed % for ev(i) until ev(i+) at d
    else % (t_green_side > t_elapsed)
        % all t_elapsed used as green for side road, and then green for main road is
        t_red_imposed = t_elapsed
        green_time_avaialble = 0 % for ev(i) untils ev(i+1) at d
    end

    % produce queue during t_elapsed
    start = (cycle_entry - 1) * cycle_length + entry_point
    while (t <= start + t_elapsed)
        queue = queue + 1
        if pointer >= length(arrival)
            break
        end
    end

```



```

        pointer = pointer + 1
        t = arrival(pointer)
    end % while
    if queue > 0
        for k = 1:queue
            green_time_available = green_time_available - turn(queue_start + k - 1) *
            discharge_rate_green
            if green_time_available < 0
                break
            end
            discharged_green = k
        end % for
        discharged_for_ev = discharged_green
        queue_start = queue_start + discharged_green
        queue = queue - discharged_green
    end % if queue > 0
    queue_formed = queue
    queue_length(cycle_entry) = queue % produced at the end of t_ev period
    turn_out = turn(queue_start:pointer-1)
    queue_out = queue_formed
    pointer_out = queue_start
end % if on_route

```

```

%-----
function [queue_out,pointer_out] = optimal_intersection(flag,cycle_entry_in,
instant_entry,queue_3,pointer_3,entrytime_ev,time_end,tp);
% generate queue length on approach 1-4 under optimal control for transition 2
% from entrytime_ev+tp to time_end
% ( relative to cycle_entry_in*cycle_length+entry_point)
% flag=0 or 1, no/one additional cycle length used for extension of recovery period
% tp=transition time for real time control
% cycle_entry_in=the number of cycle in which EV arrives
% instant_entry=the time instant at which EV arrives, 1-4 for at 0,green(1)/2,green(1),
%             and green(1)+yellow(1)+allred(1)+green(2)/2, respectively.
% using level_1,level_2,level_3,level_4 one-dimensional array as the data structure for
% control sequence
global level_1 level_2 level_3 level_4 green yellow allred a_matrix b_vector unknown
global green_max green_min
global pp_matrix wx_matrix wuu_matrix r_matrix s_matrix sg_matrix sy_matrix
global volume a_1 a_2 a_3 a_4

detector_distance = 960
speed_ev = 15 % EV operating speed SEV(m/s or 54 Km/h)
t_ev = tp

```

```

% get initial queue on each approach corresponding to input cycle_entry_in and
% instant_entry
queue_in = queue_3
pointer_in = pointer_3

arrival_1 = dlmread('arrival_1.dat');
arrival_2 = dlmread('arrival_2.dat');
arrival_3 = dlmread('arrival_3.dat');
arrival_4 = dlmread('arrival_4.dat');
turn_1 = dlmread('turn_1.dat');
turn_2 = dlmread('turn_2.dat');
turn_3 = dlmread('turn_3.dat');
turn_4 = dlmread('turn_4.dat');
% turn=turning factor corresponding to each arrival with straight-through (1.0),
% left-turn (1.4), or right-turn (1.2)
offset = 0 % relative to green for EV approach
queue_mean = [18.8, 7.25, 11.8, 13.55]
% Average queue (veh) under normal operation

green = [60,40]
yellow = [3,3]
allred = [1,1]
cycle_length = green(1) + yellow(1) + allred(1) + green(2) + yellow(2) + allred(2);

% begin optimal control
% 1 Select P=P'=R=I(4x4 identity) and S= I (2x2 identity)
p_matrix = eye(4);
pp_matrix = eye(4); % pp_matrix stands for p' matrix
r_matrix = eye(4);
s_matrix = eye(2);
green_max = [100,100];
green_min = [7,10];
saturation(1:4) = 2000 / 3600; % in veh/sec
for k=1:4
    discharge_rate_green(k) = 1 / saturation(k);
    % for straight-through vehicles during green in sec/veh
end

discharge_rate_yellow = 2 * discharge_rate_green;
% for straight-through vehicles during yellow in sec/veh
yellow_equival_green = yellow(1)/2;
% transfer yellow interval to equivalent green one
cycle_number = 20
tt = cycle_length * cycle_number; % total simulation duration in sec.

```

```

wu_matrix = green;
wuu_matrix = [green(1),0; 0,green(2); green(1),0; 0,green(2)];
% muu_matrix stands for mu' matrix
wx_matrix = eye(4);
for k=1:4
    wx_matrix(k,k) = wx_matrix(k,k)/queue_mean(k);
end
sg_matrix = eye(4);
for k=1:4
    sg_matrix(k,k) = saturation(k);
end
sy_matrix = sg_matrix/2;
% 4 Decide time step duration,t_step, and the total number of time step, step_needed
green_min = [7,10];
max_min = green_min(1)
if max_min < green(2)
    max_min = green_min(2)
end
t_step_max = 2 * max_min + yellow(1) + yellow(2) + allred(1) + allred(2)
t_step_min = green_min(1) + green_min(2) + yellow(1) + yellow(2) + allred(1) +
allred(2)
switch instant_entry
    case 1
        entry_point = 0 % time point away from the beginning of green at which EV
enters
    case 2
        entry_point = green(1) / 2
    case 3
        entry_point = green(1)
    case 4
        entry_point = green(1) + yellow(1) + allred(1) + green(2)/2
        % only this case in red period
end % switch
entry_point = entry_point + entrytime_ev % relative to beginning of green
t_quotient = fix(entry_point / cycle_length)
cycle_entry = cycle_entry_in
cycle_entry = cycle_entry + t_quotient % actual entry cycle #
entry_point = entry_point - cycle_length * t_quotient
% time elapsed from the beginning of green interval
rnc = time_end - (entrytime_ev + tp) % Remaining transition period
if rnc < t_step_min
    rnc = rnc + cycle_length * flag
end
step_needed = round(2* rnc/(t_step_max + t_step_min))
if step_needed==0

```

```

    step_needed = 1
end
t_step = rnc / step_needed % in second

% 5 Generate equivalent through vehicle arrivals, A_1/2/3/4(i), for each time step
start_point = (cycle_entry - 1) * cycle_length + entry_point
a_1(step_needed) = 0; % arrivals on approach 1 during each time step
a_2(step_needed) = 0;
a_3(step_needed) = 0;
a_4(step_needed) = 0;
a_1 = arrival_produce(arrival_1, turn_1, start_point, step_needed, t_step);
a_2 = arrival_produce(arrival_2, turn_2, start_point, step_needed, t_step);
a_3 = arrival_produce(arrival_3, turn_3, start_point, step_needed, t_step);
a_4 = arrival_produce(arrival_4, turn_4, start_point, step_needed, t_step);
% 6 Produce combination of delta1(i) and delta2(i) for current M-step sequences
% immediatly after transition 1, indication is in green on main road, so
option_1 = [1,2,3,4,9,10,11];
tn_1 = 0;
tn_2 = 0;
tn_3 = 0;
tn_4 = 0;
for k = 1:step_needed
    level_1 = [];
    level_2 = [];
    level_3 = [];
    level_4 = [];
    tn_1 = length(option_1);
    if k == step_needed
        % only level_1 applicable. construct data structure.level_1. for it
        tn_2 = data_structure_1(option_1);
    elseif k + 1 == step_needed
        tn_2 = data_structure_1(option_1);
        tn_3 = data_structure_2(tn_2);
    elseif k + 2 == step_needed
        tn_2 = data_structure_1(option_1);
        tn_3 = data_structure_2(tn_2);
        tn_4 = data_structure_3(tn_3);
    else % k + 2 < step_needed , set other data:
        level_4 = [delta_1(4), delta_2(4), g_1(4), g_2(4)]
        tn_2 = data_structure_1(option_1);
        tn_3 = data_structure_2(tn_2);
        tn_4 = data_structure_3(tn_3);
        data_structure_4(t_step);
        level_4(1,1) = step_needed;
    end
end

```

```

list_option(k,step_needed,tn_1,tn_2,tn_3,tn_4);
if k==1
    level_1(1:tn_1,13) = 0; % gc_1(k)
    level_1(1:tn_1,14) = 0; % gc_2(k)
elseif k>1
    level_1(1:tn_1,12) = operation(k-1,12) + operation(k-1,2);
    % gc_1(k)= gc_1(k-1)+g1(k-1)
    level_1(1:tn_1,13) = operation(k-1,13) + operation(k-1,3);
    % gc_2(k)= gc_2(k-1)+g2(k-1)
end
pi_optimal = 1000;
% unknown vector=(g_1(i),g_2(i),q_1(i),q_2(i),q_3(i),q_4(i),q_1(i+1),q_2(i+1),
% q_3(i+1),q_4(i+1)) = lambda_1(i),lambda_2(i),lambda_3(i),lambda_4(i),
% lambda_1(i+1),lambda_2(i+1),lambda_3(i+1),
% lambda_4(i+1),sigma(i))'
% first construct coefficient matrix, a_matrix, and right-hand side column vector,
% b_vector: a_matrix(equation#,variable#), b_vector(equation#)
if k == 1
    queue_initial = queue_in;
end
% different at each instant_entry and cycle_entry, queue formed at the end of
% transition 1
queue_desired = queue_mean;
equation_tn = 0;
a_matrix = [];
b_vector = [];
equation_tn = eq_initial(equation_tn,queue_initial);
% construct eqs for initial queues (eq 12)
list_equation_coefficient;
step_remained = step_needed - k + 1;
equation_tn = eq_final(equation_tn,queue_desired,step_remained);
% construct eqs for final desired queues (eq 13)
list_equation_coefficient;
eq_fixed_0 = equation_tn % used for reset a_matrix,b_vector after each iteration
for k_1 = 1:tn_1
    [g_1(1),g_2(1),delta_1(1),delta_2(1)] = get_data(1,k_1); % indicator=1 for
level_1
    equation_tn = equation_add(1,eq_fixed_0,g_1(1),g_2(1));
    % indicator=1 for level_1
    % construct eqs for q_1(1)=0 or g_2(1)=0
    list_equation_coefficient;
    step_n = k;
    equation_tn = eq_level(1,equation_tn,t_step,step_needed,step_n,delta_1(1),
    delta_2(1),a_1,a_2,a_3,a_4);
    % construct eqs for level_1; (indicator=1)

```

```

list_equation_coefficient;
eq_fixed_1 = equation_tn
gc_1(step_n) = level_1(k_1, 13);
gc_2(step_n) = level_1(k_1, 14);
seq(1) = level_1(k_1, 1) % sequence# for this option
if k == step_needed
    unknown = a_matrix \ b_vector';
    unknown = unknown'
    if unknown(1) < -0.1 % set g_1 = green_min(1) while g_1 is (applicable & < 0)
        a_matrix(equation_tn+1, 1:19) = 0;
        a_matrix(equation_tn+1, 1) = 1;
        b_vector(equation_tn+1) = green_min(1);
        equation_tn = equation_tn + 1;
        unknown = a_matrix \ b_vector';
        unknown = unknown'
    end
    if unknown(2) < -0.1 % set g_2 = green_min(2) while g_2 is (applicable & < 0)
        a_matrix(equation_tn+1, 1:19) = 0;
        a_matrix(equation_tn+1, 2) = 1;
        b_vector(equation_tn+1) = green_min(2);
        equation_tn = equation_tn + 1;
        unknown = a_matrix \ b_vector';
        unknown = unknown'
    end
    [gc_1, gc_2] = constraint_ver(1, seq, gc_1, gc_2, t_step, step_n);
    % verification of constraints
    option_entry = [];
    option_entry = k_1
    fill_back(1, option_entry);
    % fill back data structure for level 1 (indicator=1)
    pi = pi_calculate(1, queue_mean);
    % calculate PI for only one level (indicator=1)
    if pi < pi_optimal % select optimal option
        pi_optimal = pi;
        level_1(k_1, 10) = pi; % fill back pi into data structure
        optimal_list(1, 1:21) = level_1(k_1, 1:21); % for level 1
    end
    initialize(eq_fixed_0);
    continue % goto next k_1
end
tn_2 = level_1(k_1, 11)
for k_2 = 1:tn_2
    pointer_2 = level_1(k_1, 21) + k_2 - 1;
    % pointer to sub_option position in level_2
    [g_1(2), g_2(2), delta_1(2), delta_2(2)] = get_data(2, pointer_2);

```

```

equation_tn = equation_add(2,eq_fixed_1,g_1(2),g_2(2));
equation_tn = eq_level(2,equation_tn,t_step,step_needed,step_n,delta_1(2),
delta_2(2),a_1,a_2,a_3,a_4);
% construct eqs for level_2 (indicator=2)
list_equation_coefficient
eq_fixed_2 = equation_tn
seq(2) = level_2(pointer_2,1) % sequence# for this option in level_2
if k + 1 == step_needed
    unknown = a_matrix \ b_vector';
    unknown = unknown'
    if unknown(1)<-0.1 % set g_1 = green_min(1) while g_1 is (applicable & <0)
        a_matrix(equation_tn+1,1:19)=0;
        a_matrix(equation_tn+1,1)=1;
        b_vector(equation_tn+1)=green_min(1);
        equation_tn = equation_tn + 1;
        unknown = a_matrix \ b_vector';
        unknown = unknown'
    end
    if unknown(2)<-0.1 % set g_2 = green_min(2) while g_2 is (applicable & <0)
        a_matrix(equation_tn+1,1:19)=0;
        a_matrix(equation_tn+1,2)=1;
        b_vector(equation_tn+1)=green_min(2);
        equation_tn = equation_tn + 1;
        unknown = a_matrix \ b_vector';
        unknown = unknown'
    end
    if unknown(19+1)<-0.1
        % set g_1 = green_min(1) while g_1 is (applicable & <0)
        a_matrix(equation_tn+1,1:19+11)=0;
        a_matrix(equation_tn+1,19+1)=1;
        b_vector(equation_tn+1)=green_min(1);
        equation_tn = equation_tn + 1;
        unknown = a_matrix \ b_vector';
        unknown = unknown'
    end
    if unknown(19+2)<-0.1
        % set g_2 = green_min(2) while g_2 is (applicable & <0)
        a_matrix(equation_tn+1,1:19+11)=0;
        a_matrix(equation_tn+1,19+2)=1;
        b_vector(equation_tn+1)=green_min(2);
        equation_tn = equation_tn + 1;
        unknown = a_matrix \ b_vector';
        unknown = unknown'
    end
    [gc_1,gc_2] = constraint_ver(2,seq,gc_1,gc_2,t_step,step_n);

```

```

% verification of constraints
option_entry = [];
option_entry = [k_1,pointer_2];
fill_back(2,option_entry);
% fill back data structure for level 1&2
pi = pi_calculate(2,queue_mean);
% calculate PI for only two levels (indicator=2)
if pi < pi_optimal % select optimal option
    pi_optimal = pi;
    level_1(k_1,10) = pi; % fill back pi into data structure
    optimal_list(1,1:21) = level_1(k_1,1:21); % for level 1
    optimal_list(2,1:21) = level_2(pointer_2,1:21); % for level 2
end
initialize(eq_fixed_1);
continue % goto next k_2
end
tn_3 = level_2(k_2,11);
for k_3 = 1:tn_3
    pointer_3 = level_2(k_2,21) + k_3 - 1;
    % pointer to sub-option position in level_3
    [g_1(3),g_2(3),delta_1(3),delta_2(3)] = get_data(3,pointer_3);
    equation_tn = equation_add(3,eq_fixed_2,g_1(3),g_2(3));
    equation_tn = eq_level(3,equation_tn,t_step,step_needed,step_n,delta_1(3),
    delta_2(3),a_1,a_2,a_3,a_4);
    % construct eqs for level_3 (indicator=3)
    eq_fixed_3 = equation_tn
    if k + 2 < step_needed
        g_1(4)=level_4(1,2);
        g_2(4)=level_4(1,3);
        delta_1(4)=1;
        delta_2(4)=1;
        for k_4 = (k + 3):step_needed
            % construct eqs for fixed g(1) and g(2)
            a_matrix(equation_tn,1) = 1;
            b_vector(equation_tn) = g_1(4);
            equation_tn = equation_tn + 1;
            a_matrix(equation_tn,2) = 1;
            b_vector(equation_tn) = g_2(4);
            equation_tn = equation_tn + 1;
            %construct eqs for level k_4
            equation_tn = eq_level(4,equation_tn,t_step,step_needed,step_n,
            delta_1(4),delta_2(4),a_1,a_2,a_3,a_4);
        end % for k_4 (indicator=4)
    end
end
unknown = a_matrix \ b_vector';

```



```

unknown = unknown'
if unknown(1)<-0.1 % set g_1 = green_min(1) while g_1 is (applicable & <0)
    a_matrix(equation_tn+1,1:19)=0;
    a_matrix(equation_tn+1,1)=1;
    b_vector(equation_tn+1)=green_min(1);
    equation_tn = equation_tn +1;
    unknown = a_matrix \ b_vector';
    unknown = unknown'
end
if unknown(2)<-0.1 % set g_2 = green_min(2) while g_2 is (applicable & <0)
    a_matrix(equation_tn+1,1:19)=0;
    a_matrix(equation_tn+1,2)=1;
    b_vector(equation_tn+1)=green_min(2);
    equation_tn = equation_tn +1;
    unknown = a_matrix \ b_vector';
    unknown = unknown'
end
if unknown(19+1)<-0.1
    % set g_1 = green_min(1) while g_1 is (applicable & <0)
    a_matrix(equation_tn+1,1:19+11)=0;
    a_matrix(equation_tn+1,19+1)=1;
    b_vector(equation_tn+1)=green_min(1);
    equation_tn = equation_tn +1;
    unknown = a_matrix \ b_vector';
    unknown = unknown'
end
if unknown(19+2)<-0.1
    % set g_2 = green_min(2) while g_2 is (applicable & <0)
    a_matrix(equation_tn+1,1:19+11)=0;
    a_matrix(equation_tn+1,19+2)=1;
    b_vector(equation_tn+1)=green_min(2);
    equation_tn = equation_tn +1;
    unknown = a_matrix \ b_vector';
    unknown = unknown'
end
if unknown(19+11+1)<-0.1
    % set g_1 = green_min(1) while g_1 is (applicable & <0)
    a_matrix(equation_tn+1,1:19+11+11)=0;
    a_matrix(equation_tn+1,19+11+1)=1;
    b_vector(equation_tn+1)=green_min(1);
    equation_tn = equation_tn +1;
    unknown = a_matrix \ b_vector';
    unknown = unknown'
end
if unknown(19+11+2)<-0.1

```

```

    % set g_2 = green_min(2) while g_2 is (applicable & <0)
    a_matrix(equation_tn+1,1:19+11+11)=0;
    a_matrix(equation_tn+1,19+11+2)=1;
    b_vector(equation_tn+1)=green_min(2);
    equation_tn = equation_tn +1;
    unknown = a_matrix \ b_vector';
    unknown = unknown'
end
seq(3) = level_3(pointer_3,1)
[gc_1,gc_2] = constraint_ver(3,seq,gc_1,gc_2,t_step,step_n);
% verification of constraints
option_entry = [];
option_entry = [k_1,pointer_2,pointer_3];
fill_back(3,option_entry);
% fill back data structure for level 3
pi = pi_calculate(3,queue_mean);
% calculate PI for 3 levels (indicator=3)
if pi < pi_optimal % select optimal option
    pi_optimal = pi;
    level_1(k_1,10) = pi; % fill back pi into data structure
    optimal_list(1,1:21) = level_1(k_1,1:21); % for level 1
    optimal_list(2,1:21) = level_2(k_2,1:21); % for level 2
    optimal_list(3,1:21) = level_3(k_3,1:21); % for level 3
end
initialize(eq_fixed_2);
end % for k_3
initialize(eq_fixed_1);
end % for k_2
initialize(eq_fixed_0);
end % for k_1
% change queue_initial
queue_initial(1:4) = [];
queue_initial(1:4) = optimal_list(1,4:7)
% change option_1
option_1 = [];
sub_tn = optimal_list(1,11); % should ==: tn_1 = length(option_1)
pointer = optimal_list(1,21);
for kk = 1:sub_tn
    option_1(kk) = level_2(pointer,1);
    pointer = pointer +1;
end
operation(k,1:21) = optimal_list(1,1:21) % the final operation adopted for step k
end % for k = 1:step_needed
queue_length = 0
for k = 1:4

```

```

    queue_length(k) = operation(step_needed, 3+k) % queue at the end of transition 2
end
queue_out = queue_length
% calculate total vehicle arrivals
vehicle_arrival = zeros(1,4)
for kk = 1:step_needed
    vehicle_arrival(1) = vehicle_arrival(1) + a_1(kk)
    vehicle_arrival(2) = vehicle_arrival(2) + a_2(kk)
    vehicle_arrival(3) = vehicle_arrival(3) + a_3(kk)
    vehicle_arrival(4) = vehicle_arrival(4) + a_4(kk)
end
% total discharged
vehicle_discharged = zeros(1,4)
vehicle_discharged = queue_in + vehicle_arrival - queue_length
vehicle_discharged_interger = zeros(1,4)
vehicle_discharged_interger(1) = round(vehicle_discharged(1))
vehicle_discharged_interger(2) = round(vehicle_discharged(2))
vehicle_discharged_interger(3) = round(vehicle_discharged(3))
vehicle_discharged_interger(4) = round(vehicle_discharged(4))
% pointer to queue head after optimal control
pointer_out = pointer_in + vehicle_discharged_interger
list_queue(instant_entry,cycle_entry,queue_length)
[fid,message] = fopen('queue_length.dat', 'a'); % 1 for standard, 2 for error
fprintf(fid,'instant_entry = %-3d ',instant_entry);
fprintf(fid,' cycle_entry = %-3d \n',cycle_entry);
fprintf(fid,' initial queue length on:\n');
for kk = 1:4
    fprintf(fid,' approach %-3d',kk);
    fprintf(fid,'= %-6.2g ',queue_in(kk));
end
fprintf(fid,' \n');
fprintf(fid,' final queue length on:\n');
for kk = 1:4
    fprintf(fid,' approach %-3d',kk);
    fprintf(fid,'= %-6.2g ',queue_length(kk));
end
fprintf(fid,' \n');
fprintf(fid,' vehicle arrivals during each time_step on:\n');
fprintf(fid,' approach 1 = ');
for kk = 1:step_needed
    fprintf(fid,'%6.2g ',a_1(kk));
end
fprintf(fid,' \n');
fprintf(fid,' approach 2 = ');
for kk = 1:step_needed

```

```

        fprintf(fid,'%6.2g ',a_2(kk));
    end
    fprintf(fid,'\n');
    fprintf(fid,' approach 3 = ');
    for kk = 1:step_needed
        fprintf(fid,'%6.2g ',a_3(kk));
    end
    fprintf(fid,'\n');
    fprintf(fid,' approach 4 = ');
    for kk = 1:step_needed
        fprintf(fid,'%6.2g ',a_4(kk));
    end
    fprintf(fid,'\n');
    status = fclose(fid);
    if status == -1 % =0 if successful closed
        disp('error')
        return
    end

    if (cycle_entry_in == 5) && (instant_entry == 1)
        fid = fopen('queue_optimal.dat','w') % 1 for standard, 2 for error
    else
        fid = fopen('queue_optimal.dat','a') % append data to the end of the file
    end
    fprintf(fid,'%6.2f%6.2f%6.2f%6.2f\n',queue_length)
    status = fclose(fid)
    if status == -1
        disp(message)
        return
    end
    if (cycle_entry_in == 10) && (instant_entry == 4)
        fid = fopen('queue_optimal.dat','r') % 1 for standard, 2 for error
        if fid == -1 % no such file
            disp(message)
            return
        end
        queue_length = []
        queue_length = fscanf(fid,'%g %g %g %g ', [4 inf]) % app. 1,2, 3 and 4 for each
row,
        % totally 24 columns(4 instant_entry x6cycle)
        status =fclose(fid)
        if status == -1 % =0 if successful closed
            disp('error')
            return
        end
end

```

```

    queue_average = mean(queue_length') % averages of columns
    standard_deviation = std(queue_length') % std of columns
end

% calculate average recovery time, and only time_recovery_out_1 is useful
if (cycle_entry_in == 5) && (instant_entry == 1)
    fid = fopen('time_recovery.dat','w') % 1 for standard, 2 for error
else
    fid = fopen('time_recovery.dat','a') % append data to the end of the file
end
fprintf(fid,'%6.2f\n',rnc)
status = fclose(fid)
if status == -1
    disp(message)
    return
end
if (cycle_entry_in == 10) && (instant_entry == 4)
    fid = fopen('time_recovery.dat','r') % 1 for standard, 2 for error
    if fid == -1 % no such file
        disp(message)
        return
    end
    time_recovery_vector = []
    time_recovery_vector = fscanf(fid,'%g')
    % app. 1 for one row, totally 24 columns(4 instant_entry x6cycle)
    status = fclose(fid)
    if status == -1 % =0 if successful closed
        disp('error')
        return
    end
    time_recovery_average = mean(time_recovery_vector) % averages of columns
    standard_deviation = std(time_recovery_vector) % std of columns
end

%-----
% Other functions used in optimal_intersection are the same as in Appendix G, and not
% listed here

```