Expressive Real-time Intersection Scheduling: New Methods for Adaptive Traffic Signal Control

Rick Goldstein

CMU-RI-TR-21-01

January 2021

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics.

For my parents, Hallie & Steven Goldstein

Abstract

Traffic congestion is a widespread problem throughout global metropolitan areas. In this thesis, we consider methods to optimize the performance of traffic signals to reduce congestion. We begin by presenting Expressive Real-time Intersection Scheduling (ERIS), a schedule-driven intersection control strategy that runs independently on each intersection in a traffic network. For each intersection, ERIS maintains separate estimates of the vehicles on each lane, allowing it to more accurately estimate the effects of scheduling decisions than previous schedule-driven control approaches. ERIS outperforms a less expressive schedule-driven control approach and a fully actuated control method in a variety of simulated traffic environments.

We examine several limitations to ERIS and discuss methods to improve schedule quality. First, we consider that when each intersection minimizes a local objective based on delay, global delay is not necessarily minimized. We propose Centralized Agent Reducing Intersection Congestion (CARIC), a centralized method that augments ERIS. CARIC generates potential green wave coordination plans, asks individual intersections to score these plans, and selects a plan that minimizes global delay. As a result of coordination, vehicles are able to travel across the network while encountering successive green lights, ultimately reducing delay and fuel usage when compared against a decentralized control strategy. Additionally, we examine the impact of green wave coordination plans across a variety of demand profiles and train a model estimating when green wave coordination plans are beneficial. Integrating this model with CARIC improves performance when demand is highly variable.

Second, we present two additional limitations of ERIS that lead to inefficient decision making: its finite sensing horizon and its objective function. We examine three extensions to ERIS to address these limitations: generating expected vehicle clusters based on average arrival rates, adapting the objective function to include a term for schedule makespan, and adapting the objective function to weight vehicles travelling along each movement (direction) differently. We notice considerable improvement when adapting the objective function to weight vehicles and examine this extension in further detail. We run additional experiments on a larger network demonstrating improvement, present a library of rules for when adapting the objective function to weight vehicles is beneficial, and develop a queuing model that supports the results of these experiments.

Contents

1 Introduction **1**

1.1 Motivation . . . 1

1.2 The Intersection Control Problem . . . 1

1.2.1 Overview . . . 1

1.2.2 Fixed Timing Plans . . . 2

1.3 Real-time Intersection Control . . . 3

1.3.1 Vehicle Detection . . . 4

1.3.2 Schedule-Driven Control . . . 4

1.4 Challenges of Real-time Schedule-Driven Control . . . 5

1.5 Thesis Statement . . . 6

1.6 Reading Guide to the Thesis . . . 7

2 Background **9**

2.1 Parametric Methods . . . 9

2.2 Schedule-Driven Methods . . . 10

2.2.1 Overview of Schedule-Driven Methods . . . 10

2.2.2 Detailed Discussion of Surtrac . . . 11

2.2.3 Surtrac2012 Limitations . . . 13

2.2.4 Recent Improvements to Surtrac2012 . . . 14

2.3 Other Real-time Methods . . . 14

2.3.1 Machine Learning Techniques . . . 14

2.3.2 Max Pressure . . . 14

2.3.3 Shared Global Timing Plans with Decentralized Adaptive Management . 15

2.3.4 Actuated Control . . . 15

2.4 Methods for Autonomous Vehicles . . . 15

2.4.1 Adapting Speed . . . 15

2.4.2 Elimination of Traffic Lights . . . 16

2.5 Simulator Overview . . . 16

I Expressive Real-time Intersection Scheduling **17**

3 Expressive Real-time Intersection Scheduling **18**

3.1 Overview . . . 18

3.2 Traffic Light Controller Assumptions 19
3.3 Related Work 20
3.4 Expressive Real-time Intersection Scheduling 23
3.4.1 Overview 23
3.4.2 Cluster Construction Example 24
3.5 Scheduling Module 25
3.5.1 Search Overview 26
3.5.2 Search States 26
3.5.3 Start State 27
3.5.4 Goal State 27
3.5.5 State Transitions 27
3.5.6 Heuristic Function 29
3.5.7 Closed List & Dominance Checks 32
3.6 Experimental Evaluation on Synthetic Networks 33
3.6.1 Perfect Information Connected Vehicle Environment 33
3.6.2 Baselines 34
3.6.3 Simulation Implementation Details 34
3.6.4 Single Intersection 35
3.6.5 3 x 3 Grid of Intersections 36
3.6.6 Statistical Significance 36
3.6.7 Timing Analysis 37
3.7 St. Albert Network 37
3.7.1 Network Overview 37
3.7.2 Hour-long Experiments 39
3.7.3 Three Hour-Long Experiment 42
3.7.4 ERIS Setting Examination 42
3.7.5 Relaxing the 100% Connectivity Assumption 44
3.8 Conclusion 45

II Green Wave Coordination 47

4 Integrating Green Wave Coordination with ERIS 48
4.1 Overview 48
4.1.1 Motivating Example 48
4.1.2 Chapter Outline 49
4.2 Background 49
4.2.1 Green Waves 50
4.2.2 Related Work 51
4.3 CARIC 52
4.3.1 Green Wave Definitions 53
4.3.2 CARIC Architecture & Planning Pipeline 54
4.3.3 Planning Pseudo-code 55
4.3.4 CARIC Activities 59

4.4 Modifications to ERIS 61
4.4.1 Satisfying Local Green Wave Constraints 62
4.4.2 Aggressive Cluster Splitting 63
4.4.3 Delaying Green Signals at the First Intersection 64
4.5 Experimental Analysis on the Portland-Franklin Network 64
4.5.1 Network Overview 65
4.5.2 Experimental Setup 65
4.5.3 Examining the Re-Planning Frequency 66
4.5.4 Validation Experiment 68
4.5.5 Evaluation using Scaled Down Demands 69
4.5.6 Evaluation using Scaled Down Wave Demands 70
4.5.7 Discussion 72
4.6 Conclusion 72

5 Considerations for Applying Green Wave Coordination 73
5.1 Overview 73
5.2 Network Considerations 73
5.2.1 Overview 73
5.2.2 Experimental Setup 75
5.2.3 Results 75
5.2.4 Discussion 76
5.3 Learning when Green Wave Coordination Plans are Beneficial 77
5.3.1 Network and Demand Overview 77
5.3.2 Training the Support Vector Machine 78
5.3.3 Outputted SVM Model 78
5.3.4 Adaptive Green Wave Strategy 79
5.3.5 Experimental Evaluation 80
5.3.6 Discussion 81
5.4 Conclusion 81

III Addressing Inefficiencies of the ERIS Scheduler 84

6 Three Proposed Extensions to Reduce Inefficiencies of the ERIS Scheduler 85
6.1 Overview 85
6.2 Motivating Example 86
6.3 Proposed Extensions 87
6.3.1 Phantom Clusters 87
6.3.2 Penalizing Higher Makespans 89
6.3.3 Weighting Approaches 91
6.4 Related Work 92
6.5 Experimental Analysis on Two-Movement Intersections 93
6.5.1 1x1 and 2x2 Intersections 93
6.5.2 2x1 Intersection 95

6.5.3 Discussion . . . 96
6.6 Critical Movements . . . 97
6.6.1 Approximate Definition of Critical Movements . . . 97
6.6.2 Formal Definition of Critical Movements . . . 97
6.6.3 Example Calculation . . . 99
6.6.4 Additional Considerations . . . 99
6.7 Experimental Analysis on Eight-Movement Intersection . . . 100
6.7.1 Phantom Clusters - Parameter Sweep . . . 101
6.7.2 Penalizing Higher Makespans - Parameter Sweep . . . 102
6.7.3 Weighting Approaches - Parameter Sweep I . . . 103
6.7.4 Weighting Approaches - Parameter Sweep II . . . 105
6.7.5 Validating and Combining the Three Extensions . . . 106
6.7.6 Weighting Approaches - Validation Experiment . . . 108
6.7.7 Summary . . . 109
6.8 Conclusion . . . 109
6.9 Appendix . . . 110
6.9.1 Critical Movement Experiment for Different Vehicle Types . . . 110
6.9.2 Full Results for Experiment from Section 6.7.3 . . . 112

7 Additional Weighting Considerations to Reduce Inefficiencies of the ERIS Scheduler 119
7.1 Overview . . . 119
7.2 Deterministic Queuing Model . . . 119
7.2.1 Actuated Control Strategies . . . 120
7.2.2 Model Definition . . . 120
7.2.3 Model Evaluation with $\Delta = 0$ . . . 124
7.2.4 Model Evaluation with $\Delta > 0$ . . . 128
7.2.5 Considering Different Arrival Rates . . . 132
7.2.6 Discussion . . . 137
7.3 Experimental Analysis on Halton Hills Network . . . 138
7.3.1 Network Overview . . . 138
7.3.2 Calculation of Critical Movements . . . 139
7.3.3 Experimental Evaluation . . . 140
7.4 Additional Examination of the Intersection at Steeles Avenue and Trafalgar Road 141
7.4.1 Experimental Setup . . . 142
7.4.2 Parameter Sweep . . . 142
7.4.3 Validation Experiment . . . 143
7.4.4 Discussion . . . 144
7.5 Summary of when to Consider Weighting . . . 145
7.6 Conclusion . . . 146
7.7 Appendix: Additional Model Calculations . . . 147
7.7.1 Additional Formulation, $\Delta = 0$ . . . 147
7.7.2 Additional Formulation, $\Delta > 0$, Fully Actuated Control . . . 148
7.7.3 Additional Formulation, $\Delta > 0$, Critical Movement Actuated Control . . 151

8 Conclusion **154**
8.1 Contribution Overview . . . 154
8.1.1 Part I: Expressive Real-time Intersection Scheduling . . . 155
8.1.2 Part II: Green Wave Coordination . . . 155
8.1.3 Part III: Addressing Inefficiencies of the ERIS Scheduler . . . 156
8.2 Future Work . . . 156
8.2.1 Additional Model Granularity . . . 156
8.2.2 Model Selection . . . 157
8.2.3 Additional Prioritization Strategies . . . 157

Bibliography **158**

List of Figures

1.1 Example Intersection 2

3.1 Example Intersection 20
3.2 Dual Ring Barrier, from [47] 20
3.3 Compatible Green Phase Orderings 20
3.4 Example Intersection for Combining Clusters Discussion 21
3.5 Surtrac2012's Internal Model of the Intersection after Combining Clusters 21
3.6 ERIS' Internal Model of Intersection after 40 Seconds 22
3.7 Surtrac2012's Internal Model of the Intersection after 40 Seconds 22
3.8 ERIS Control Flow 24
3.9 Intersection for Clustering Example 25
3.10 West-Origin Turn Ratios 25
3.11 Arrival Timeline and Clusters 25
3.12 Phase Transitions Example 28
3.13 Heuristic Calculation Pipeline 30
3.14 One Possible Sub-Problem Split 31
3.15 St. Albert Network 38

4.1 Example Network where Decentralized Solvers Fail to Make Globally Optimal Decisions 49
4.2 Network To Illustrate Green Waves 50
4.3 CARIC Planning Pipeline 54
4.4 Network with Vehicles To Illustrate Green Wave Coordination Plan Generation . 55
4.5 Portland-Franklin Network 66
4.6 Portland-Franklin Network with Green Wave Route Indicated 67

5.1 Short-Synthetic-Portland-Franklin Network 74
5.2 Portland-Franklin Network 74
5.3 Long-Synthetic-Portland-Franklin Network 74
5.4 Synthetic Network 77
5.5 Support Vector Machine's Predictions of when to Activate Green Wave Coordination Plans (shown in green) 79

6.1 Motivating Example 87
6.2 Modifying the Motivating Example with a Phantom Cluster 88

6.3 Eight-Movement Intersection with Movement-Specific Demands 99
6.4 Bit to Movement Mapping for Weighting Extension 103

7.1 Cycle Duration and Average Delay for All Arrival Rates 133
7.2 Cycle Duration and Average Delay for Selected Arrival Rates 133
7.3 Relationship between Cycle Duration and Average Delay 134
7.4 Change in Delay for Critical Movement Actuated Control at Various Equilibrium Cycle Durations when $\Delta = 2.0$ 135
7.5 Change in Delay for Critical Movement Actuated Control at Various Equilibrium Cycle Durations when $\Delta = 4.0$ 136
7.6 Halton Hills Network 139
7.7 Intersection at Steeles Avenue and Trafalgar Road with Provided Demand 140

List of Tables

1.1 Sample Fixed Timing Plan . . . 3

3.1 Average Vehicle Delay (s) for a Single Intersection . . . 35
3.2 Average Vehicle Delay (s) for a 3 x 3 Grid . . . 36
3.3 Runtime Comparisons (ms), Single Intersection . . . 37
3.4 Average Vehicle Delay (s) on St. Albert Network - Hour-long Experiments . . . 39
3.5 Average Number of Stops on St. Albert Network - Hour-long Experiments . . . 40
3.6 Average Runtimes (ms) . . . 41
3.7 95th Percentile Runtimes (ms) . . . 41
3.8 Percentage of Runs over 1 Second . . . 41
3.9 Average Vehicle Delay (s) on St. Albert Network - Three Hour-Long Experiment 42
3.10 Clustering Strategy Comparison . . . 43
3.11 Left Turn Scheduling Strategy Comparison . . . 43
3.12 Dropping Left Turn Clusters Comparison . . . 44
3.13 Cluster Splitting Comparison . . . 44
3.14 Average Vehicle Delay (s) from Different Connectivity Penetrations . . . 45

4.1 Evaluation for Different Re-Planning Frequencies . . . 67
4.2 Evaluation at Provided Demand for Portland-Franklin Network . . . 68
4.3 Average Delay (s) at Scaled Down Demands for Portland-Franklin Network . . . 69
4.4 Average Fuel Usage (mL) at Scaled Down Demands for Portland-Franklin Network 69
4.5 Average Number of Stops at Scaled Down Demands for Portland-Franklin Network 69
4.6 Average Delay (s) at Scaled Down Wave Demands for Portland-Franklin Network 71
4.7 Average Fuel Usage (mL) at Scaled Down Wave Demands for Portland-Franklin Network . . . 71
4.8 Average Number of Stops at Scaled Down Wave Demands for Portland-Franklin Network . . . 71

5.1 Delay (s) at Scaled Down Demands for the Short-Synthetic-Portland-Franklin Network . . . 75
5.2 Delay (s) at Scaled Down Demands for the Portland-Franklin Network . . . 75
5.3 Delay (s) at Scaled Down Demands for the Long-Synthetic-Portland-Franklin . . 75
5.4 Evaluation of Scheduling Strategies . . . 81

6.1 Average Vehicle Delay (s) at 1x1 Intersection . . . 94

6.2 Average Vehicle Delay (s) at 2x2 Intersection 94
6.3 Average Vehicle Delay (s) at 2x1 Intersection 95
6.4 Average Vehicle Delay (s) for Phantom Clusters Parameter Sweep 101
6.5 Average Vehicle Delay (s) for Makespan Parameter Sweep 102
6.6 Average Vehicle Delay (s) for Weighting Parameter Sweep I - Top 20 Strategies . 104
6.7 Average Vehicle Delay (s) for Weighting Parameter Sweep II 105
6.8 Average Vehicle Delay (s) for Validation Experiment 107
6.9 Average Vehicle Delay (s) for Weighting Validation Experiment 108
6.10 Average Vehicle Delay (s) for Different Weighting Strategies 111
6.11 Average Vehicle Delay (s) for Weighting Parameter Sweep I - All Strategies . . . 118

7.1 Model Example #1, Partial Calculation #1 125
7.2 Model Example #1, Partial Calculation #2 126
7.3 Model Example #1, Partial Calculation #3 126
7.4 Model Example #1, Partial Calculation #4 126
7.5 Model Example #1, Partial Calculation #5 127
7.6 Model Example #1, Full Model 127
7.7 Model Example #2 127
7.8 Model Example when $\Delta = 2$, Fully Actuated Control, Partial Calculation #1 . . . 129
7.9 Model Example when $\Delta = 2$, Fully Actuated Control, Partial Calculation #2 . . . 130
7.10 Model Example when $\Delta = 2$, Fully Actuated Control, Full Model 130
7.11 Model Example when $\Delta = 2$, Critical Movement Actuated Control, Partial Calculation 131
7.12 Model Example when $\Delta = 2$, Critical Movement Actuated Control, Full Model . 132
7.13 Average Vehicle Delay (s) across Different Weighting Strategies on the Halton Hills Network - Demands 3300 veh/hr and below 140
7.14 Average Vehicle Delay (s) across Different Weighting Strategies on the Halton Hills Network - Demands 3520 veh/hr and above 141
7.15 Average Number of Stops across Different Weighting Strategies on the Halton Hills Network - Demands 3520 veh/hr and above 141
7.16 Average Vehicle Delay (s) across Different Weighting Strategies for Intersection at Steeles Avenue and Trafalgar Road 144

Chapter 1

Introduction

1.1 Motivation

Traffic congestion is a widespread problem throughout global metropolitan areas. Traffic congestion causes increases in travel time, increases in emissions, inefficient usage of gasoline, and driver frustration. Estimates of the economic impact of congestion in United States urban areas vary, but tend to be on the order of several hundred billion dollars per year. For example, Schrank et al. (2019) estimates the economic costs of congestion in United States urban areas in 2017 to be \$166B while Cookson and Pishue (2017) estimates these costs to be roughly \$305B.

There is much political and commercial interest in reducing traffic congestion and its negative consequences. The Intergovernmental Panel on Climate Change warns of "climate-related risks to health, livelihoods, food security, water supply, human security, and economic growth" if temperatures rise $1.5°$C or more [51]. Similarly, major technology companies as well as many startups are investigating methods to improve vehicle routing, make parking more efficient, implement smart intersection scheduling strategies, and build autonomous vehicles [7, 68].

While there are many research areas considering how to reduce traffic congestion, this thesis focuses on one specific area - improving traffic signal timings at intersections. Traffic signals are inefficiently timed and often fail to react to real-time traffic conditions. Intersection scheduling strategies that make real-time decisions to extend or end a green signal based on real-time traffic data offer one opportunity to reduce congestion and its negative impacts.

1.2 The Intersection Control Problem

1.2.1 Overview

In the intersection control problem, we consider an intersection, such as the one presented in Figure 1.1 where traffic signals have been installed to improve mobility or increase safety. A traffic light controller, located at each intersection, is responsible for controlling the individual traffic signals. In the intersection control problem, we dictate to the traffic light controller how to act at every time step.

An intersection has inbound lanes from various directions. An intersection may associate

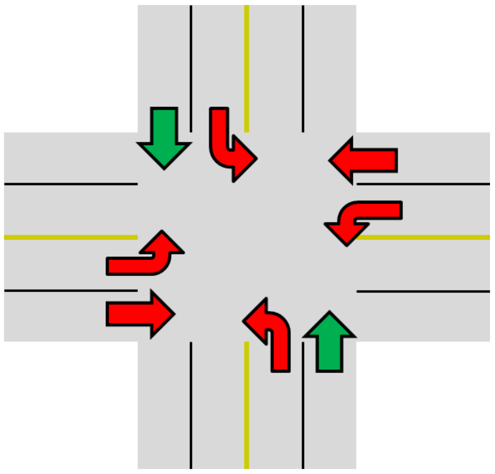


Figure 1.1: Example Intersection

a subset of lanes, referred to as a movement, with a green signal to indicate to vehicles in the respective lanes that it is safe to traverse the intersection. The intersection in Figure 1.1 is currently serving two green movements, namely the north straight movement and the south straight movement. Green movements must satisfy prespecified minimum and maximum timing limits. Similarly, we require yellow and red clearances with fixed lengths to occur between successive green movements for safety.

In the intersection control problem, we are interested in generating a feasible plan for allocating green time to different movements that will minimize a measure of disutility. One common objective that we primarily focus on in this thesis is minimizing the average vehicle delay, which is how long a vehicle must wait for its turn to pass through the intersection. A vehicle's delay is the difference between its expected arrival time at the intersection (assuming no congestion) and when it actually passes through the intersection. Other potential objectives include minimizing travel time, minimizing the number of vehicle stops, minimizing fuel usage, or minimizing emissions. Furthermore, some systems attempt to minimize a weighted average of several of these objectives while other systems attempt to minimize one of these objectives for a subset of vehicles (e.g. minimizing travel time for vehicles travelling along the busiest direction).

1.2.2 Fixed Timing Plans

The majority of intersections in the United States implement some sort of fixed timing plan (also known as a pre-timed plan). In a fixed timing plan, the traffic light controller at the intersection provides the same fixed time schedule regardless of how many vehicles are currently at the intersection. For example, Table 1.1 presents a 45-second-long fixed timing plan. Every 45 seconds, the intersection provides, in this specific order, 15 seconds of green time to the main street, 5 seconds of yellow time to the main street, 5 seconds of all red time, 10 seconds of green time to the side street, 5 seconds of yellow time to the side street, and 5 more seconds of all red time. After the 45 seconds have elapsed, this schedule repeats.

An intersection will often have a separate fixed timing plan for different portions of the day, referred to as time-of-day plans. For example, the intersection might implement an AM rush hour

Time (Seconds)	Main Street	Side Street
0 - 5	Green	Red
5 - 10	Green	Red
10 - 15	Green	Red
15 - 20	Yellow	Red
20 - 25	Red	Red
25 - 30	Red	Green
30 - 35	Red	Green
35 - 40	Red	Yellow
40 - 45	Red	Red

Table 1.1: Sample Fixed Timing Plan

plan, a middle of the day plan, a PM rush hour plan, and an evening plan. Intersections may implement the same plan every weekday and then a distinct plan for weekends. Some intersections may implement special event plans, though this is less common. When several intersections are nearby, fixed timing plans may be generated to improve flow through the network in a particular direction.

Fixed timing plans are often generated based on a historical count of vehicles, observed speeds of vehicles, intersection geometry, and other factors. The Highway Capacity Manual details many considerations traffic professionals should consider when generating a fixed timing plan [44]. Additionally, commercially available software, such as Synchro, is commonly used to generate such fixed timing plans [63]. Much work has considered how to adjust fixed timing plans in real-time based on recently observed traffic flow [42, 43, 50, 71].

While these dynamic methods offer an improvement in performance when compared to static plans, there remains room for improvement. Fixed timing plans run the same schedule regardless of the current concentrations of vehicles; green time is often allocated inefficiently. Fixed timing plans may provide green signals while no vehicles are present on the corresponding movement or terminate a green signal while many vehicles still remain. Additionally, fixed timing plans may lead to too many or too few signal transitions between movements leading to too much red time or wasted green time.

1.3 Real-time Intersection Control

Using real-time data offers an opportunity to improve upon fixed timing plans. The real-time intersection control problem assumes the use of real-time information about vehicle locations to make decisions. Real-time data can be gathered from a variety of sources, including cameras, radar, underground induction loop detectors, connected vehicles transmitting information, and intersections communicating expected outflows to neighbors.

A variety of real-time intersection control methods have been proposed [2, 10, 15, 24, 25, 28, 46, 56, 66, 69]. While we defer a more detailed discussion of such methods to Chapter 2, for now it will suffice to briefly highlight one such method - schedule-driven control.

This section presents an overview of vehicle detection methods used by real-time methods and then introduces schedule-driven control.

1.3.1 Vehicle Detection

Real-time methods require information about nearby vehicles before they can generate a schedule. Presently, two common methods to detect vehicles are underground induction loop detectors and cameras. Induction loop detectors are loops of wire installed into the pavement of the road. When a vehicle passes over the induction loop, the loop's inductance (current) changes [13, 29]. Additionally, cameras can be installed on the poles or wires that hold traffic signals to detect vehicles.

These systems have several limitations including their cost and the quality and quantity of data they provide. Detection systems can cost over \$10,000 per intersection [21]. Additionally, detectors are imperfect and often miscount vehicles. Furthermore, detectors only provide an estimate of where a vehicle is located at a certain point in time. Speed limits and empirically observed speeds are then used to estimate arrival times at an intersection.

Connectivity offers the opportunity to provide improved detection for schedule-driven control methods. Over 80% of American adults have smartphones and this percentage continues to increase [11]. Many smartphone users use mobile applications such as Google Maps and Waze to obtain real-time traffic data to improve their commutes. It seems plausible that some individuals would be willing to share information with these applications or others if it meant improved travel experiences. Similarly, several car manufactures are beginning to incorporate connected technologies into their vehicles. Audi is incorporating on-board cellular into some models to provide drivers with countdowns to green lights [4]. Cadillac has begun incorporating DSRC into their CTS Sedans [9]. The United States Department of Transportation is also highly interested in the potential of connected vehicles for safety and increased mobility, funding three connected vehicle pilots in New York City, Tampa, and Wyoming [64].

Connected devices allow vehicles to easily share information, including their position and speed, with nearby intersections. If sufficiently many vehicles share information using connected devices, we may be able to avoid the large upfront sensor costs. Additionally, connected devices offer the potential to provide more accurate information than the current generation of sensors. This may lead to better performance and make investment into smart intersection scheduling strategies more cost-effective.

1.3.2 Schedule-Driven Control

In schedule-driven control, a computer (often located at the intersection) receives information from sensors to construct an estimate of nearby vehicle locations [24, 28, 46, 56, 69]. After constructing this estimate of nearby vehicle locations, the computer calculates a schedule that allows vehicles to pass through the intersection while minimizing a selected cost function. (Schedules must obey intersection feasibility rules, which we explain in more detail in Chapter 3). Once a schedule has been calculated, the first step of the schedule is passed to the traffic light controller to implement. This process usually repeats every time step.

Schedule-driven methods make a variety of assumptions to ensure real-time solvability. For example, several methods calculate a schedule based on a fixed resolution (for example, every phase duration must be a multiple of 5 seconds) [24, 28, 56]. Methods that make this assumption will not examine all possible schedules and will often fail to identify the best schedule. Other methods do not require a fixed resolution but make other assumptions to limit the problem complexity to ensure that it can be solved in real-time [46, 69]. For example, the original version of Surtrac combined clusters that utilize compatible movements into one larger cluster (e.g. northbound and southbound vehicle clusters are sometimes combined into one larger cluster since both receive a green signal simultaneously) [69]. However, vehicles will not always be able to pass simultaneously, leading to inaccuracies in the predicted quality of particular schedules.

Although such models are imperfect, there have been successful deployments of such schedule-driven control methods. For example, Surtrac's Pittsburgh, Pennsylvania deployment has successfully improved traffic when compared to fixed timing plans generated by Synchro [59]. Average travel times were reduced by over 25%, average wait time was reduced by over 40% and emissions were reduced by 21%. Surtrac has since been deployed to many other cities.

1.4 Challenges of Real-time Schedule-Driven Control

Several challenges make real-time schedule-driven control difficult. In this section, we enumerate these challenges and briefly discuss their interactions.

1. **Requirement of Real-time Solvability** - Schedule-driven methods typically make decisions every time step, which may be as often as every second. The time a model has to run its calculations is limited by the time step. Thus, schedule-driven methods must make modelling and optimization compromises to ensure that they can successfully return a solution within this time limit.
2. **Complexity of Traffic Dynamics** - If search states maintain information about all vehicles separately, search state updates will be too slow. Instead, schedule-driven methods, including the original Surtrac system, tend to group vehicles into clusters and usually do not allow splitting clusters [69]. As a result, there may be good schedules that split clusters that are not considered.
3. **Exponentially Large Search Space** - Comparing all possible signal timing strategies is exponentially large in the length of the scheduling horizon. Some methods require all steps to be exactly 5 seconds in duration, reducing the number of branching points of the search [24, 28, 56]. There may be good schedules with steps of different durations that are not considered.
4. **Global Coordination** - Scheduling more than one intersection at once tends to increase runtime exponentially. (For example, McCluskey and Vallati (2017) proposes a search method that requires roughly 30 seconds to generate a joint schedule for 7 intersections). As a result, many schedule-driven methods are decentralized, meaning the scheduler at each intersection optimizes a local objective. These local objectives do not always align with global objectives, and as a result, schedulers make sub-optimal decisions.

5. **Finite Sensing Horizon** - A scheduler only schedules the vehicles it has sensed through its detection system, which has limited range, or from shared information between intersections. We refer to this as the finite sensing horizon. Vehicles outside of the finite sensing horizon will continue to arrive at the intersection and most methods fail to consider the impact of their decisions on these vehicles.
6. **Inadequate Objective Function** - A scheduler that minimizes delay every time step will not necessarily minimize delay during the entire evaluation period (even if the evaluation period is as short as 5 minutes). A decision that minimizes delay over the scheduling horizon may cause queues to form along some movements, ultimately increasing delay during future time steps.

These challenges are interlinked. Better modelling often comes at the expense of an increased runtime. For example, increasing the complexity of traffic dynamics will make the search space larger and increase the runtime. Similarly, considering a more complex objective function increases runtime.

1.5 Thesis Statement

Given the increasing concern surrounding climate change and the reduction in the cost of vehicle detection due to connectivity, we believe the importance of real-time intersection control to be increasing while the costs of implementing successful strategies to be decreasing. While several real-time algorithms have been successfully deployed, there is additional algorithmic work to be performed to further improve intersection scheduling. In this thesis, we investigate the following areas to address the above scheduling challenges.

A **More Expressive Scheduling Model** By developing a more detailed traffic model that maintains a better estimate of traffic dynamics than current state of the art models, we can reduce congestion and delay (Challenge #2). When developing this model, we implement a faster scheduling algorithm (using A* search) to ensure that schedules can be generated in real-time (Challenges #1 and #3).

B **Green Wave Coordination** By analyzing green wave coordination strategies that allow intersections to jointly optimize their schedules, intersections can coordinate their schedules to allow vehicles to travel through multiple intersections without stopping, which is difficult when intersections only consider local objectives (Challenge #4).

C **Methods Addressing the Finite Sensing Horizon and Inadequate Objective Function** By considering three extensions to our core scheduler (modifying the objective function, adding phantom vehicle clusters, and optimizing weighting parameters), we can calculate schedules that do not solely seek to minimize short-term delay, ultimately leading to less congestion (Challenges #5 and #6).

Thesis Statement: We investigate schedule-driven traffic signal control methods to demonstrate their ability to improve traffic flow and reduce the negative impacts of traffic congestion.

1.6 Reading Guide to the Thesis

The remainder of this thesis is organized as follows. In Chapter 2, we present a more detailed overview of signal control strategies, including but not limited to summarizing multiple schedule-driven control methods. Following the related work, the remainder of the thesis is divided into three parts. In the first part, in Chapter 3, we present Expressive Real-time Intersection Scheduling (ERIS), our core schedule-driven control strategy.

The second part presents CARIC, which is a layer that sits on top of ERIS and calculates green waves that allow vehicles to travel across a network more efficiently. Chapter 4 explains CARIC and green waves, and Chapter 5 discusses a method that adaptively applies green waves.

The third part of this thesis considers additional inefficiencies in ERIS' decision making. Chapter 6 discusses how ERIS' performance can be improved by considering three extensions. Chapter 7 carefully examines the most promising of these extensions (weighting vehicles).

We conclude in Chapter 8.

A more detailed chapter overview is presented below:

Chapter 2 - Background provides an overview of several other intersection control strategies. We discuss parametric methods, machine learning methods, schedule-driven control methods, and methods for autonomous vehicles. We provide a detailed overview of Surtrac, which is the method most similar our method, ERIS.

Chapter 3 - Expressive Real-time Intersection Scheduling presents ERIS, our core schedule-driven control strategy. This chapter details the specifics of our underlying traffic model and details how these are captured by the ERIS framework. We explain the A* search employed to efficiently calculate schedules and present an example of our underlying heuristic calculations. We present a variety of experiments across different networks comparing ERIS to baseline methods, including a method based on the original version of Surtrac.

Chapter 4 - Integrating Green Wave Coordination with ERIS considers adding green wave coordination constraints to allow vehicles travelling along the busiest route in a network to travel with limited stops. We present CARIC, which calculates the ideal green wave coordination constraints and shares them with intersections in the network. We discuss implementation specifics and necessary modifications to ERIS. We present results demonstrating improvement on several networks. We then examine various demand profiles to characterize which demand profiles benefit from green waves.

Chapter 5 - Considerations for Applying Green Wave Coordination considers the question of when one should run green waves (as opposed to solely running ERIS). We present high level network characteristics describing what makes a network a good candidate for green waves. We also train a support vector machine to characterize when it is beneficial to use green waves. We present experiments demonstrating that this model that informs when to adaptively turn on and off green waves outperforms simpler strategies.

Chapter 6 - Three Proposed Extensions to Reduce Inefficiencies of the ERIS Scheduler presents several situations where ERIS makes decisions that appear optimal in the short run but ultimately increase congestion. We propose three candidate extensions that aim to help ERIS calculate better schedules. We examine generating phantom clusters that will arrive in the near future, penalizing schedules with longer makespans, and weighting movements. We run experiments examining each of these methods and argue that weighting movements is the strongest of these three methods.

Chapter 7 - Additional Weighting Considerations to Reduce Inefficiencies of the ERIS Scheduler examines weighting movements in additional detail. We propose a deterministic queuing model supporting the experimental results from Chapter 6. We then present an experiment demonstrating the importance of weighting on a simulated network based on a real-world network. We carefully examine the busiest intersection from this network to better understand the importance of weighting the critical movements for some demand profiles while weighting based on the relative number of lanes for other demand profiles. We conclude this chapter by presenting a library of rules for when to apply weighting strategies.

Chapter 8 - Conclusion summarizes our contributions while noting how they address the challenges presented in Section 1.4. We then propose several areas for future work.

Chapter 2

Background

In Chapter 1, we introduced the intersection control problem and briefly described fixed timing plans, the real-time intersection control problem, and schedule-driven control. In this chapter, we provide more detail about a variety of intersection control strategies. We begin by describing parametric methods. We then explain schedule-driven methods and present a detailed overview of the original version of Surtrac, which is the method most similar our method, ERIS, that we will present in Chapter 3. We then briefly describe several other real-time methods including machine learning techniques, Max Pressure, methods with global and local planning, and actuated control. We then discuss several planning methods specifically for autonomous vehicles. We conclude this chapter by providing an overview of the traffic simulator that will be used in experiments throughout this thesis.

A general treatment of all past intersection scheduling work is beyond the scope of this thesis. We refer the reader to the works by Shelby (2001) and Stevanovic (2010) for additional information about other intersection control strategies not covered in this document.

2.1 Parametric Methods

We introduced fixed timing plans in Chapter 1. Recall that a fixed timing plan runs the same schedule of phases (with fixed times) for a given duration. We refer to methods that allow us to calculate these fixed timing plans as parametric methods.

These strategies set timing parameters, typically *cycle time*, *splits*, and *offsets*. *Cycle time* refers to the period of the pattern and generally applies to an entire network of intersections. *Splits* dictate how to divide the cycle time across different movements at each traffic signal within the network. *Offsets* specify how to stagger the start times of green phases across the network.

TRANSYT was one of the first non-adaptive intersection control strategies to calculate these timing parameters by running an off-line hill-climbing algorithm to minimize an objective, such as delay or number of stops, on historical data [49]. Presently, Synchro has become a common software tool used by traffic professionals to generate these timing plans, though details about the algorithms it implements are not explicitly mentioned on their website [63].

Several more recent procedures, such as SCOOT, SCATS, and ACS Lite perform optimizations similar to TRANSYT that adapt timing parameters during execution based on observed real-time traffic data [42, 43, 50]. SCOOT, SCATS, and ACS Lite adapt gradually to changing traffic conditions. These methods have been deployed on a small fraction of intersections in the United States. For example, Siemens, a major provider of traffic signals, has deployed SCOOT across roughly 15 cities in the United States [57]. The number of intersections in each deployment varies, and is not always easily accessible. As one example, Siemens has deployed SCOOT across 32 intersections in downtown Seattle [53]. Similarly, Econolite, another provider of traffic signals, offers their Centracs Edaptive platform, which adapts offsets based on a 'link-pivot' algorithm [19]. The link-pivot algorithm moves one direction along a corridor, greedily calculating the optimal offset between adjacent intersections, assuming that all previously calculated offsets are fixed [16].

Some fixed time deployments have event plans for sporting events or accidents. Yao and Qian (2019) use real-time traffic data and train an LTSM to predict accidents and send alerts to the township to implement a contingency fixed timing plan. This method currently requires a human in the loop to ultimately make the decision to activate the contingency plan. Qualitative results are positive, but no published quantitative values on estimated time savings exist.

2.2 Schedule-Driven Methods

The above methods do not react in real-time but rather integrate real-time information across one or more cycles and make adjustments over a timescale of several minutes. When realized flows on any side vary more rapidly than expected, these methods provide unnecessary green time that could be better allocated to busier sides.

These next two sections discuss methods that react to real-time traffic congestion and make changes based on real-time traffic conditions. This section specifically describes several schedule-driven methods while the next section discusses other real-time methods. Schedule-driven methods examine multiple feasible schedules of green phase timings and select the schedule that optimizes an objective, such as minimizing delay, based on a prediction of when vehicles will cross the intersection. The first step of this schedule is then executed and then the process repeats. A variety of schedule-driven methods have been developed over the years, and the assumptions have become less restrictive.

2.2.1 Overview of Schedule-Driven Methods

We present an overview of many of these methods and then a detailed description of Surtrac.

Shelby (2001) proposes a branch and bound method that uses linear and neural networks to rank states to dictate which states are most promising to expand. This ranking function can both overestimate and underestimate remaining cost to goal, so there are no guarantees on the optimality of the returned schedule without incorporating additional pruning logic. Additionally, this model requires every phase duration to be a multiple of 5 seconds and looks at a fixed planning horizon of 15 time steps.

There are several real-time dynamic programming schedule-driven methods. Gartner (1983) and Henry et al. (1984) each propose dynamic programs that assume a fixed step size of 5 seconds to calculate a schedule. Similarly, Sen and Head (1997) proposes a real-time dynamic programming method, referred to as Controlled Optimization of Phases, with fixed step sizes. Such course granularity can distort actual traffic conditions rather significantly. More recent methods, such as Surtrac, do not require a fixed step size though make other assumptions to limit the problem difficulty to ensure that calculations can execute in real-time [69].

Al Islam and Hajbabaie (2017) propose a mixed-integer linear programming method to solve for optimal schedules within a connected vehicle framework. They assume a relaxed underlying traffic model that drops several practical constraints. They have no fixed phase ordering, no required yellow or red clearances, and no phase minimum or maximum requirements. As such, they are able to instantaneously jump between phases. In our work, we respect common intersection dynamics such as the fixed phase ordering, dual ring barrier controller logic, and fixed yellow and red clearances. The underlying traffic model presented in their work is similar to the relaxation step of our heuristic calculation.

McCluskey and Vallati (2017) proposes a centralized PDDL+ model to calculate an optimal schedule for a network of seven intersections. While it calculates a schedule several minutes in duration, solving this model requires on the order of 30 seconds to obtain a solution. This is impractical for our work as we wish to be able to run in real-time and re-optimize our solution every second.

2.2.2 Detailed Discussion of Surtrac

Our work most closely resembles the original Surtrac system [58, 69]. Surtrac is a decentralized schedule-driven method that runs a forward dynamic program to calculate an optimal schedule for known vehicle clusters in real-time (as often as every second) by treating each intersection as a single-machine scheduling problem. Surtrac has been modified and improved in several ways since these original publications, not all of which have been publicly documented. Many of the improvements that have been published (e.g. the sharing of additional information with neighboring intersections [33]) are actually complementary and could be equally applied to the approach described in this thesis, so it is reasonable to consider this version as a good benchmark against which to measure the current research. For the remainder of this thesis, we will refer to this earlier version of Surtrac as ‘Surtrac2012.’ We present a more detailed description of Surtrac2012 so that we may explain similarities with ERIS.

During each time step, Surtrac2012 begins by generating an estimate of vehicles near the intersection using detector data and information about predicted downstream arrivals received from neighboring intersections. Vehicles travelling along the same movement and within a provided distance cutoff parameter are grouped into the same cluster. In addition to the specific movement, clusters are defined by the number of vehicles in the cluster, their arrival time at the intersection, and an expected departure time if the cluster were to begin passing through the intersection once it arrives. These pieces of information can be used to deduce several other useful parameters, including the cluster’s expected flow rate through the intersection once it receives a green signal.

To simplify the search problem, Surtrac2012 combines the clusters from compatible movements into a larger set of clusters, which we refer to as a ‘movement group.’ For example, a cluster travelling along the north straight movement beginning at time 20 and a cluster travelling along the south straight movement also beginning at time 20 will be combined into a single, larger cluster. When Surtrac2012 calculates a schedule, it schedules movement groups, assuming that vehicles from each movement in the movement group will depart the intersection simultaneously. This assumption limits the complexity of the search, but is not always accurate.

Once the clusters have been calculated and bucketed into their appropriate movement group, Surtrac2012 runs a dynamic programming search to calculate the optimal schedule to serve these clusters. The dynamic programming scheduler assumes that clusters may not be divided and also includes constraints relating to traffic dynamics, including phase minimum and maximum times and yellow and red clearance times. Specifically, the scheduler proceeds by generating states in the form $(\vec{X}, s, pd, t, \text{delay})$.

- $\vec{X}$ denotes the number of clusters along each movement group that have been served by the schedule up to this point in time. This is initialized to a vector of all zeros. The search ultimately looks to calculate a goal state that serves all vehicle clusters (namely, each element in $\vec{X}$ corresponds to the total number of clusters along the respective movement group). Clusters in a given movement group are served in a first-come first-served fashion. (Knowing that a particular number of clusters have been served along a particular movement group uniquely determines which clusters have been served).
- s represents the current traffic signal phase at this point in time in the schedule. The phase is required to calculate how much transition time (if any) is required to serve a specific cluster next.
- pd represents the current amount of time that has been spent in the phase. This ensures that the schedule satisfies minimum and maximum phase time limits. If a cluster would cause Surtrac2012 to exceed a phase maximum time, Surtrac2012 includes logic to divide the cluster into two smaller clusters. However, this is the only reason that Surtrac2012 would split a cluster.
- t denotes the current time within the search state. This is required to know which clusters have arrived and calculate the delay (if any) on each candidate cluster that may be served next.
- Delay specifies the cumulative delay of the clusters served ($\vec{X}$) by the search state. Surtrac2012's goal is to calculate the search state that serves all vehicle clusters with the minimum delay.

When the forward dynamic programming search examines a search state, it will create one child state for each movement group with at least one unserved vehicle cluster. If the movement matches s, the search calculates the maximum of the current time and the cluster arrival time as the actual start time of the cluster. The search calculates how long is required to serve the cluster and assuming the maximum green phase time is not violated, uses this information to update t and pd accordingly. (When the maximum green time is violated, the cluster is split into two smaller clusters and the search is restarted). The search increases delay by the incurred delay of this cluster based on the arrival time, the actual start time, and the number of vehicles in the cluster. The appropriate element in $\vec{X}$ is also updated. s is unchanged.

If the controller's movement does not match s, the time to switch from the current movement (s) to the cluster's movement is calculated using the minimum remaining time in the current phase based on the phase minimum (if any) and the minimum yellow and red clearances required to start the desired movement. Additionally, if any phases are required to occur between s and the desired movement, their minimum green times and yellow and red clearances are also added. This provides a temporary start time that the signal may begin the next movement. If the vehicles have arrived prior to this time, then they experience a startup loss of several seconds (which is added to the temporary start time) to model the fact that these vehicles must react to the green signal and accelerate from a stop. This provides an actual start time for the cluster. If the vehicles arrive at or after the temporary start time, the cluster's actual start time is its arrival time. As above, this information is used to update the value of delay. t, pd, $\vec{X}$, and s are also updated accordingly.

To ensure that the problem can be solved in real-time, for each $\vec{X}$, only the search state with the lowest delay is maintained. (If the search were to maintain all search states and expand all of them, problem runtime would grow exponentially).

The dynamic programming search begins with the state that has served 0 clusters (meaning all elements in $\vec{X}$ are 0). Once this state has been expanded, all states with one served cluster are expanded. Then all states with two served clusters are expanded. This process repeats until the search reaches the goal state that has served all clusters. The best schedule of phases can be determined from the choices made to reach the goal state. The first step of this schedule is then formatted as a command and sent to a traffic light controller to execute. This process repeats every time step.

2.2.3 Surtrac2012 Limitations

Surtrac2012 makes several simplifications to ensure that the dynamic program can be solved efficiently. As a result of these simplifications, Surtrac2012 does not consider all possible schedules and sometimes miscalculates the delay for a particular schedule, occasionally leading to selecting a sub-optimal schedule. We will discuss how our methods address these limitations in Chapter 3.

As mentioned earlier, Surtrac2012 combines vehicle clusters from paired movements (e.g. the north straight and south straight movements) into the same cluster to ensure that the problem size remains tractable. This reduces the dimensionality of the search space, but sometimes leads to solutions that mistakenly allocate too much time to clusters that could be served simultaneously. As a result, the delay for some schedules may be overestimated and thus, these schedules may not be selected. (Please refer to Section 3.3 for an example).

A second simplification is that Surtrac2012 requires left turns to run semi-actuated, meaning left turn movements cannot be extended if no vehicles are present. While this simplifies the search space, if vehicles will soon arrive on the left turn movement, Surtrac2012 will not extend the green signal for the movement even though this may reduce delay.

A third simplification is that Surtrac2012 does not split vehicle clusters except to satisfy maximum timing limits. There may exist better schedules that involve serving a portion of a cluster, switching the active movement to serve another cluster, and then completing the first cluster. Surtrac2012 will not examine such schedules.

2.2.4 Recent Improvements to Surtrac2012

As mentioned earlier, Surtrac2012 has been modified and improved in several ways since the original publications. Here, we detail several of the published modifications. Hu and Smith (2017a, 2017b) propose calculating dynamic weights for each movement based on their relative congestion to prioritize certain movements. Hu and Smith (2019) proposes additional information sharing between neighboring intersections; specifically, downstream intersections share their congestion with upstream neighbors which allow upstream neighbors to de-prioritize sending vehicles to downstream intersections with relatively high congestion. Hu and Smith (2020) discusses using reinforcement learning to learn improved phase maximum times to reduce overall delay.

2.3 Other Real-time Methods

This section discusses several other real-time methods that are not schedule-driven methods.

2.3.1 Machine Learning Techniques

Several machine learning techniques that learn to make real-time decisions without an explicit traffic model have been proposed. Abdoos et al. (2011) and El-Tantawy and Abdulhai (2012) both propose multi-agent Q-learning strategies where intersections learn to make decisions based on queue lengths around the intersection. The model proposed in El-Tantawy and Abdulhai (2012) contains additional logic where intersections also incorporate an estimate of the decisions of neighboring intersections into their optimization. Bazzan (2005) models the interaction between intersections using evolutionary game theory to encourage intersections to consider the global impact of their decisions in their optimization.

Genders and Razavi (2016) introduces a deep Q-network traffic signal control agent to reduce queue lengths and delay. Casas (2017) trains a deep deterministic actor-critic policy gradient model to improve traffic flow speeds when compared to a random scheduling baseline.

Auction-based agents where bid weights are trained using stochastic hill-climbing have also been shown to perform well with adaptive signal control [5, 15]. In an auction based model, each sensor places a ‘bid’ on each phase representing how strongly it wants this phase to occur next. These bids are based on predefined weights (which were trained using stochastic hill-climbing) as well as local sensor data. These works illustrate that these micro-auctions can reduce travel times in the simulated networks. However, their traffic models assume that left-turn phases are skippable, meaning that the specific delay of these vehicles may vastly exceed the delays of other vehicles in the network.

2.3.2 Max Pressure

Another real-time method is Max Pressure [66]. In the original work, the Max Pressure algorithm argues that serving the movements with the largest combined queues can maximize network throughput assuming there are no change-over times [66]. This work proved that if, for a given demand profile, it is possible to stabilize the queue lengths, then Max Pressure is one such algorithm that can stabilize the queue lengths. We note that while this work is compelling and presents motivation that an algorithm needs to consider throughput, the setting without change-over time is unrealistic. It does not make practical sense to switch phases every few seconds. Follow-on work has considered allowing for change-overs, yet there is no proof that this more realistic algorithm is guaranteed to stabilize any system that is possible to stabilize [41]. The notion of maximizing throughput is an important motivation for the third part of this thesis.

2.3.3 Shared Global Timing Plans with Decentralized Adaptive Management

Several methods combine shared global timing plans with decentralized adaptive management of individual intersections. The RHODES traffic signal control system employs both centralized coordination (to optimize cluster prioritization) and local optimization (dynamic programming) [46]. Another system, In|Sync, generates global timing plans to allow vehicles to traverse along the busiest route(s) in a network without stopping; individual intersections plan their other movements locally [12]. We present a more detailed overview of In|Sync in Chapter 4, where we compare and contrast it to our contribution.

2.3.4 Actuated Control

One real-time method that is deployed across a variety of intersections in the United States is actuated control. In actuated control, detectors (such as underground induction loop detectors) estimate the current locations of vehicles and use this information to decide on the current phase. For example, a phase might remain green until it has been several seconds (e.g. 3 seconds) since the most recent vehicle passed over a detector (and appropriate phase minimum times have been satisfied) [36, 62]. In fully actuated control, all movements have detectors, which are used to determine which movements receive green signals.

In semi-actuated control, only some movements have such detectors. It is usually the case that the busiest movements do not have detectors, and green signals tend to default to these movements. Detector firings alert the intersection that there are vehicles on a less busy movement, which will (though not necessarily immediately) cause the intersection to provide a green signal to these movements.

2.4 Methods for Autonomous Vehicles

Several methods consider how traffic control may change if autonomous vehicles are commonplace.

2.4.1 Adapting Speed

Several recent works examine sending traffic signal information to connected and autonomous vehicles so that these vehicles may adapt their speed to reach a signal before it turns red. Hu et al. (2019) and Liang et al. (2019) demonstrate that sharing such information can lead to a reduction in delay across 100% autonomous vehicle penetration as well as partial penetrations.

2.4.2 Elimination of Traffic Lights

Several methods propose the elimination of traffic lights. Autonomous Intersection Management is a framework suggesting that in a world with fully autonomous vehicles, vehicles can make first-come first-serve (FCFS) reservations and weave through an intersection whenever their space-time path does not conflict with those of other vehicles [18]. Tachet et al. (2016) proposes an alternative batch scheduling algorithm that may outperform FCFS reservations during high traffic volumes. Yang and Monterola (2016) and Ferreira et al. (2010) analyze cases where partially autonomous or connected vehicles can discern when it is safe to pass through an intersection without a traditional overhead signal. The majority of these methods cannot handle disconnected users (e.g. non-connected human drivers and pedestrians); however, some recent work has begun to consider how to integrate traditional detection techniques so that intersections can serve both autonomous vehicles as well as disconnected users [55].

2.5 Simulator Overview

Throughout the thesis, we run evaluations of our methods using the Simulation of Urban MObility (SUMO) [37]. SUMO is a microscopic traffic simulator that simulates the dynamics of vehicles and traffic light controllers. We interface with SUMO through Python, using the Traffic Control Interface (TraCI) [67]. TraCI provides real-time data about vehicle speeds, locations, headings, and routes and allows for external control of traffic signals.

Part I

Expressive Real-time Intersection Scheduling

Chapter 3

Expressive Real-time Intersection Scheduling

3.1 Overview

This chapter presents Expressive Real-time Intersection Scheduling (ERIS), a schedule-driven control strategy for adaptive intersection control to reduce traffic congestion.

ERIS maintains detailed estimates of the vehicles travelling along each lane approaching a traffic intersection allowing it to more accurately estimate the effects of scheduling decisions than previous schedule-driven approaches. We present a detailed description of the search space and A* search heuristic employed by ERIS to make scheduling decisions in real-time (every second). As a result of its increased expressiveness, ERIS outperforms a less expressive schedule-driven approach and a fully actuated control method in a variety of simulated traffic connected vehicle environments.

ERIS addresses the first three challenges presented in Chapter 1 that make real-time intersection scheduling difficult; the remaining challenges will be addressed by additional methods introduced in later chapters. The three challenges addressed by ERIS are reproduced below:

1. **Requirement of Real-time Solvability** - Schedule-driven methods typically make decisions every time step, which may be as often as every second. The time a model has to run its calculations is limited by the time step. Thus, schedule-driven methods must make modelling and optimization compromises to ensure that they can successfully return a solution within this time limit.
2. **Complexity of Traffic Dynamics** - If search states maintain information about all vehicles separately, search state updates will be too slow. Instead, schedule-driven methods, including the original Surtrac system (which we will refer to as 'Surtrac2012' for the remainder of this thesis), tend to group vehicles into clusters and usually do not allow splitting clusters [69]. As a result, there may be good schedules that split clusters that are not considered.
3. **Exponentially Large Search Space** - Comparing all possible signal timing strategies is exponentially large in the length of the scheduling horizon. Some methods require all steps to be exactly 5 seconds in duration, reducing the number of branching points of the search

[24, 28, 56]. There may be good schedules with steps of different durations that are not considered.

In response to Challenge #2 of modelling traffic, ERIS adopts a lane-based framework; ERIS maintains separate estimates of the vehicles present in every lane approaching an intersection. Given the above information and learned vehicle dynamics, ERIS estimates when a vehicle will reach the intersection stop line, the probability of changing lanes, and when it will pass through the intersection.

This lane-based framework is more expressive than prior work in schedule-driven intersection control that combines compatible movements into a single movement group (as Surtrac2012 does). (Please refer to Section 3.3 for an example as to why combining compatible movements can lead to sub-optimal delay calculations and scheduling decisions). However, the increased size of our traffic state representation ultimately makes Challenge #3 of calculating an optimal schedule of signal timings (relative to the aggregate representation of the problem that is being solved) more difficult. ERIS exploits problem structure to quickly calculate lower bounds on the delay associated with potential signal patterns. ERIS runs an A* search, using this lower bound estimate as an admissible heuristic, to efficiently calculate optimal schedules every second (addressing Challenges #1 and #3). We show that Expressive Real-time Intersection Scheduling (ERIS) can reduce delay by up to 20% when compared to other real-time intersection control strategies.

The remainder of this chapter is organized as follows. We first discuss dual ring barrier traffic light controllers, which are a common traffic pattern setup for traffic light controllers and will be assumed by ERIS. We highlight several related works and discuss similarities and differences compared to ERIS. Next, we describe ERIS, our strategy for the real-time intersection control problem, and discuss the application of A* search to solve single instances of the problem. We then present several experiments demonstrating that ERIS outperforms other real-time approaches. We conclude and discuss several limitations to ERIS.

3.2 Traffic Light Controller Assumptions

While describing ERIS and in our first set of experiments, we will assume a 4-sided intersection with eight distinct movement groups as presented in Figure 3.1. Straight movements are grouped with right turn movements into a single movement group; left turn movements are separate. Additionally, we will assume that all intersections are controlled by a dual ring barrier controller. The dual ring barrier controller may provide two compatible movements with a green signal. A graphic illustrating the dual ring barrier controller is presented below in Figure 3.2. (ERIS does not require that an intersection have eight distinct movements and it is not necessary for left turns and straight movements to be served by separate lanes; in Section 3.7.1, when discussing evaluation on a network based on a real-world network, we detail modifications to the controller that allow for these modifications).

At all times, a dual ring barrier controller allows one phase from each ring (a row in Figure 3.2) to be active. The active phases must be on the same side of the bold middle line marked "Barrier." The dual ring barrier requires that both straight phases must end simultaneously. Compatible movements and legal transitions between compatible pairs based on the dual ring barrier

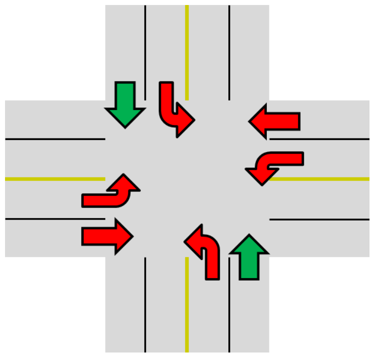


Figure 3.1: Example Intersection

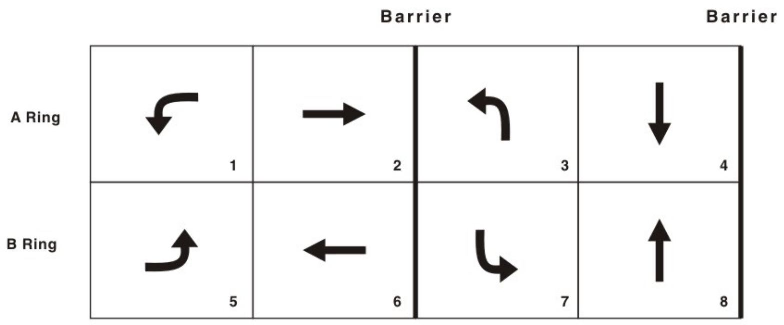


Figure 3.2: Dual Ring Barrier, from [47]

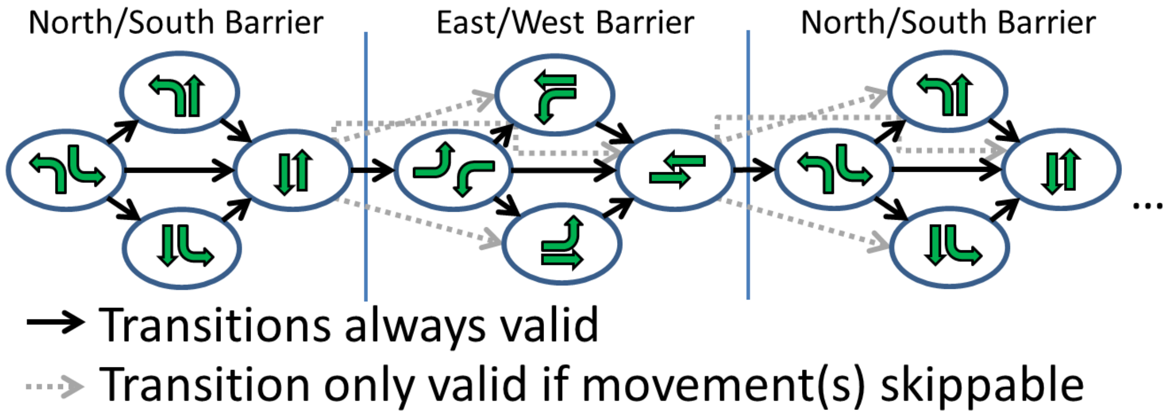


Figure 3.3: Compatible Green Phase Orderings

controller are presented in Figure 3.3. (Yellow and red clearances between compatible pairs are assumed). The dual ring barrier controller allows left turn movements to end at separate times; this increases scheduling flexibility but requires keeping track of green movements and their respective start times separately.

In our dual ring controller, left turn movements precede straight movements and left turn movements can be skipped if no vehicles are present. In our modelling, left turn movements are assumed to be protected movements. (We provide details about how we handle permissive left turn movements in Section 3.7.1).

3.3 Related Work

Our work most closely resembles the Surtrac2012 system [58, 69].

Recall from Chapter 2 that Surtrac2012 is a decentralized schedule-driven method that runs a forward dynamic program to calculate an optimal schedule for known vehicle clusters in real-time. To ensure that the forward dynamic program can run in real-time, Surtrac2012 makes simplifications to the underlying scheduling problem. Surtrac2012 schedules a simpler controller that requires left turn movements to end simultaneously, which reduces the branching factor of the dynamic programming search.

Additionally, Surtrac2012 combines vehicle clusters travelling on compatible movements (e.g. east straight and west straight) when conducting its dynamic programming search. This reduces the dimensionality of the search space, but sometimes leads to solutions that mistakenly allocate too much time to clusters that could be served simultaneously. Consider the following example:

There is a cluster of ten vehicles along the south straight movement that is currently waiting at the intersection, a cluster of ten vehicles arriving along the east straight movement at time 20, and a cluster of ten vehicles arriving along the west straight movement at time 40. Each cluster will require 20 seconds to pass through the intersection. This example is depicted in Figure 3.4.

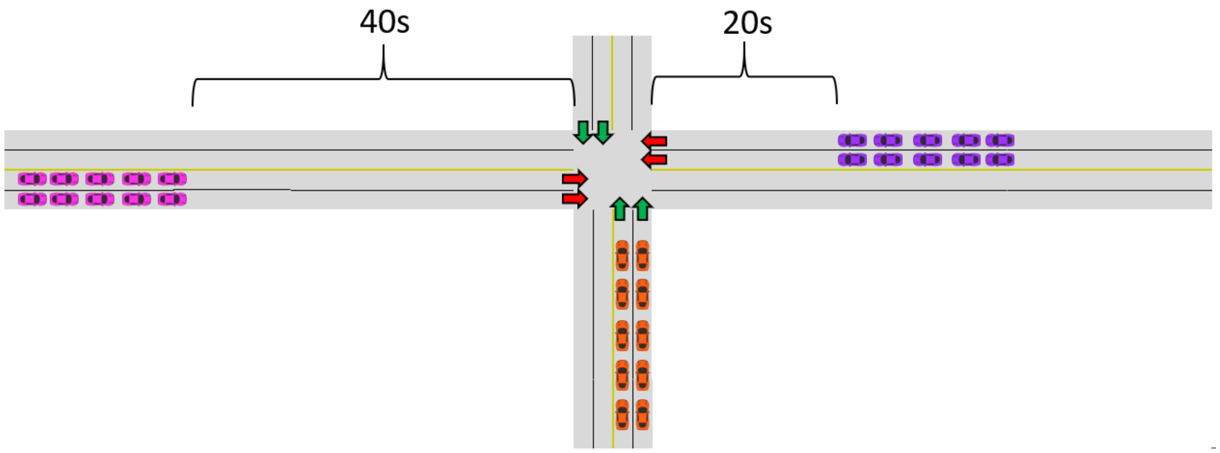


Figure 3.4: Example Intersection for Combining Clusters Discussion

Since Surtrac2012 combines clusters on compatible movements, it will assume that the east and west clusters must be scheduled sequentially. Surtrac2012's internal model of the intersection after combining clusters is depicted in Figure 3.5.

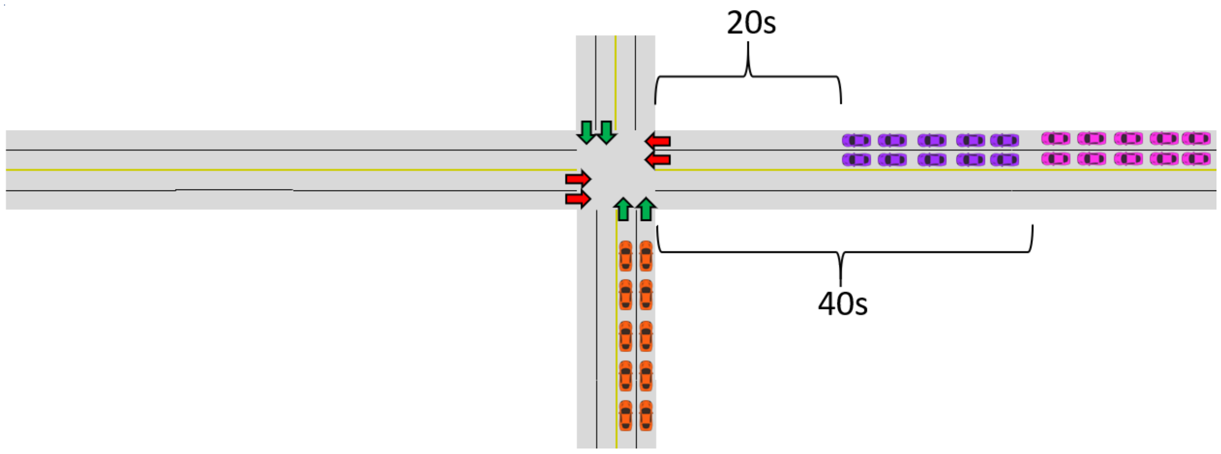


Figure 3.5: Surtrac2012's Internal Model of the Intersection after Combining Clusters

In some cases, this will not be an issue (for example, if the purple cluster is scheduled to pass through the intersection at time 20 and then the pink cluster at time 40). However, there are situations where these cluster will be able to pass through the intersection simultaneously even though they are scheduled to pass through sequentially.

For example, suppose that there is a search state with an associated time of 40 that has not yet served any vehicles along the east or west movements. Vehicles from both the east and the west movements will have approached the intersection and be able to traverse through the intersection simultaneously. This situation is depicted in Figure 3.6.

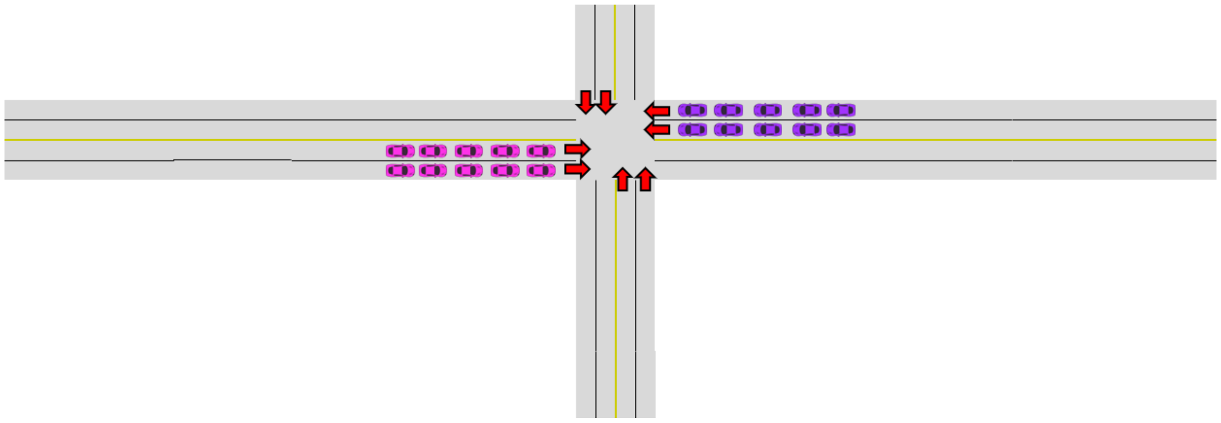


Figure 3.6: ERIS' Internal Model of Intersection after 40 Seconds

However, because of its decision to combine clusters, Surtrac2012 continues to assume that these clusters must be served sequentially. Surtrac2012's internal model of the intersection is depicted in Figure 3.7.

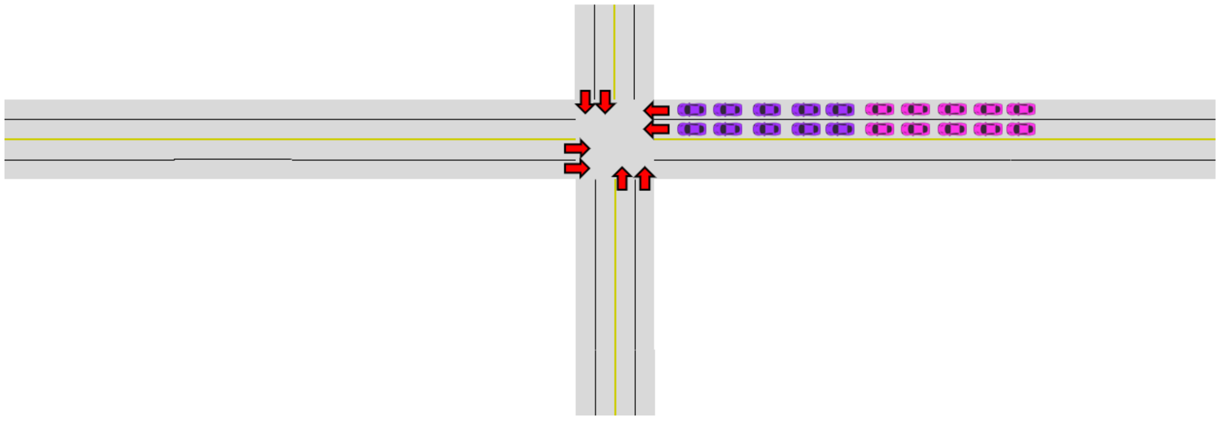


Figure 3.7: Surtrac2012's Internal Model of the Intersection after 40 Seconds

If Surtrac2012 schedules the purple cluster to pass through the intersection at time 40, it incorrectly assumes that the combined movement cannot schedule the pink cluster until the purple cluster finishes passing through the intersection at time 60. Thus, the pink cluster will be scheduled at time 60 and Surtrac2012 will overestimate the delay for this cluster (and possibly future clusters as well). This may cause Surtrac2012 to overestimate the delay for the best schedule and not select it.

By scheduling a dual ring controller and maintaining separate counts of clusters in each lane, ERIS is able to maintain a more accurate representation of the underlying traffic conditions. ERIS must solve a more challenging problem; the state space is much larger, so instead of using forward dynamic programming to calculate schedules, ERIS applies an A* search. While the A* search does not guarantee less complexity than the dynamic program, the heuristic function allows the A* search to prune many sub-optimal states and calculate a solution efficiently (in under one second).

In addition to being a decentralized schedule-driven method, ERIS adopts several underlying design choices of Surtrac2012, namely the parameterization of clusters as jobs with arrival times, flow rates, and sizes and the neighbor communication protocol.

Also recall that Shelby (2001) proposes a branch and bound method that uses linear and neural networks to rank states to dictate which states are most promising to expand. Unlike ERIS, which runs an A* search with an admissible heuristic, this ranking function can both overestimate and underestimate remaining cost to goal, so there are no guarantees on the optimality of the returned schedule without incorporating additional pruning logic. Additionally, their model requires fixed timing steps of 5 seconds and looks at a fixed planning horizon of 15 time steps. Such course granularity can distort actual traffic conditions rather significantly. ERIS, like Surtrac2012, is able to efficiently calculate schedules that are several minutes in duration without compromising granularity.

Finally, recall that Al Islam and Hajbabaie (2017) propose a mixed-integer linear programming method assuming a relaxed underlying traffic model that drops several practical constraints. Our A* search heuristic function uses a relaxed traffic model similar to the simplified model proposed in their work.

3.4 Expressive Real-time Intersection Scheduling

3.4.1 Overview

Expressive Real-time Intersection Scheduling (ERIS) is a decentralized, schedule-driven real-time intersection control method that makes a decision every second based on current traffic conditions and passes this information to the traffic light controller to execute. This section summarizes the overall control flow of an ERIS controlled intersection. The control flow is depicted in Figure 3.8.

Every second, the ERIS Executor runs an outer loop that executes the necessary modules to make a scheduling decision. The Executor first calculates a prediction of vehicle clusters in each lane using nearby vehicle broadcasts of position, speed, and heading as well as information shared from neighboring intersections about predicted future outflows. Historical turning proportions and a model of expected travel times are learned from past vehicles flows. They are used to estimate travel times and applied in cluster construction to estimate desired lane. An example

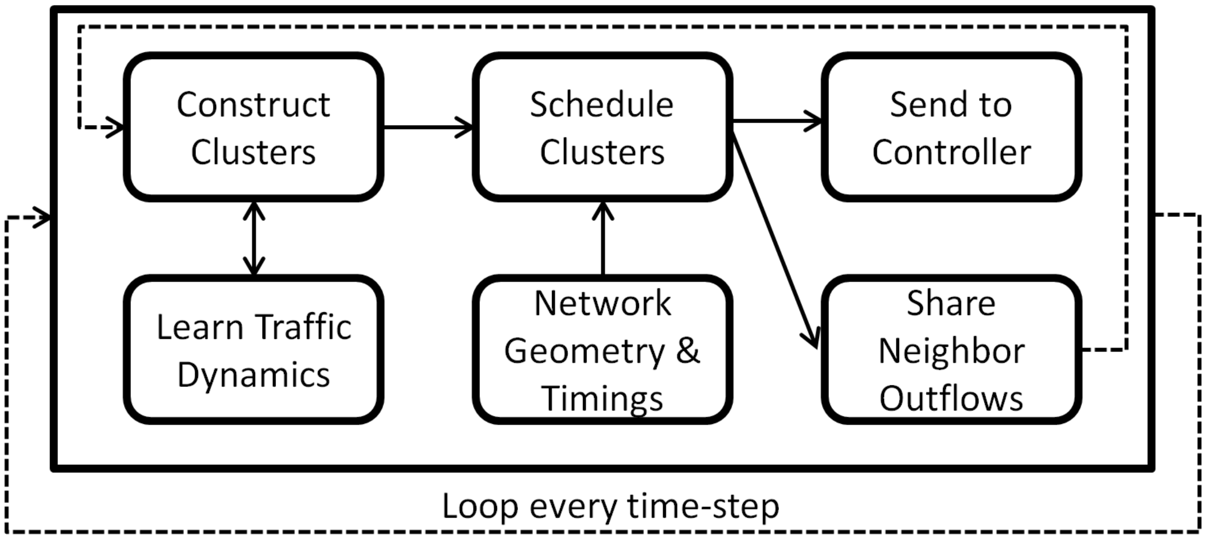


Figure 3.8: ERIS Control Flow

of cluster construction is presented in the following sub-section.

Once clusters are obtained, they are passed to the Scheduling Module to calculate an optimal schedule of phases (given our problem formulation) that minimizes delay for all vehicle clusters to pass through the intersection. Clusters are scheduled in accordance with programmed traffic dynamics including the fixed phase ordering, phase minimum and maximum times, and start-up lost time (the delay vehicles experience when accelerating from a full stop). The scheduling procedure is explained in more detail in the following section.

Once a schedule is obtained, the first step of the schedule is formatted into a command to either stay in the same phase or switch to another phase. This command is sent to the traffic light controller to execute. Similarly, scheduled cluster outflows are broadcast to and received from neighboring intersections for use during the next time step when constructing vehicle clusters.

3.4.2 Cluster Construction Example

During each iteration of the Executor, ERIS constructs clusters based on current traffic conditions. Clusters contain information about a group of nearby vehicles travelling along the same lane. Each cluster has an expected arrival time at the intersection, a size, and a flow rate. The arrival time of a cluster is the earliest time that any vehicle in the cluster would reach the intersection's corresponding stop bar. The size of the cluster is the total number of vehicles in the cluster. The flow rate is the rate at which we expect the vehicles in the cluster to pass through the intersection. Additionally, each cluster will be assigned a delay. This delay is the sum of individual delays of each vehicle in the cluster. A single vehicle's delay is the difference between its departure time at the intersection (defined as the time at which is enters the intersection) and the vehicle's expected arrival time at the intersection.

An example intersection and the resulting clusters are presented in Figure 3.9 and Figure 3.11, respectively. Our cluster calculation is presented below:

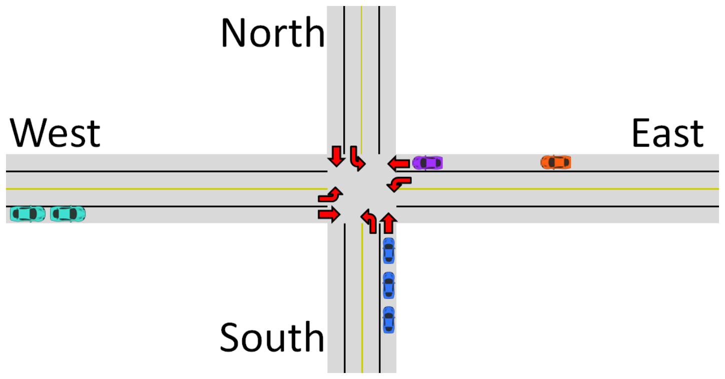


Figure 3.9: Intersection for Clustering Example

🚗	Count	%
↰	100	25%
➡	300	75%

Figure 3.10: West-Origin Turn Ratios

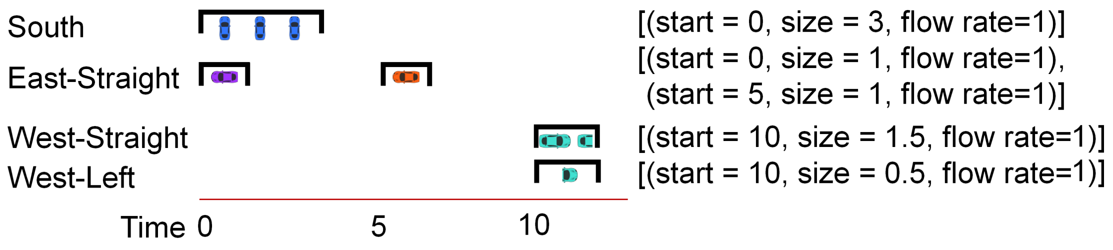

South
East-Straight
West-Straight
West-Left
Time 0
5
10
[(start = 0, size = 3, flow rate=1)]
[(start = 0, size = 1, flow rate=1),
(start = 5, size = 1, flow rate=1)]
[(start = 10, size = 1.5, flow rate=1)]
[(start = 10, size = 0.5, flow rate=1)]

Figure 3.11: Arrival Timeline and Clusters

The intersection presented in Figure 3.9 has three nearby blue vehicles travelling along the south straight lane. We group them into one cluster for the south straight lane. This cluster is currently waiting at the intersection, so it is assigned a start time of 0. The cluster consists of three vehicles, so it is assigned a size of 3. Assuming an observed average flow rate of 1 vehicle per second, this cluster is assigned an equivalent flow rate of 1 vehicle per second.

The east straight lane has two vehicles separated by roughly 5 seconds. Since the vehicles are farther apart than a fixed cut-off of 3 seconds, we place them into two separate clusters. Each cluster has a start time corresponding to the cluster's predicted arrival at the intersection (0 and 5, respectively), a size of 1, and a flow rate of 1.

The west side of the intersection has two turquoise vehicles. Since the vehicles just entered the lane, they might change lanes. Thus we divide these vehicles according to historically observed turning proportions (as shown in Figure 3.10). We obtain two turquoise clusters, one for the west straight lane and one for the west left lane. Both clusters have an arrival time of 10. Each cluster has a size corresponding to the expected number of vehicles travelling along the lane (1.5 and 0.5, respectively). Vehicles are assigned the observed flow rate of 1 vehicle per second. (A previous version of ERIS' cluster construction used weighted flow rates based on turning proportions, but this caused ERIS to assign too much time to serve larger clusters).

3.5 Scheduling Module

The Scheduling Module runs an A* search to calculate an optimal schedule with respect to the underlying model, the provided vehicle clusters, traffic light controller dynamics, and the current controller state. Search states keep track of vehicle clusters that have been scheduled on each lane as well as other important information. This section explains our overall search, individual search states, the start state, and the ending criterion. We then describe state transitions. Next, we describe our heuristic function that allows us to efficiently calculate this optimal schedule. We conclude this section with a brief discussion of our dominance check algorithm.

3.5.1 Search Overview

The search aims to calculate an optimal, feasible schedule of phases that provides all vehicle clusters within each of the L lanes adequate time to pass through the intersection, while minimizing total delay. Each of the L lanes of the intersection is provided a unique index $\in [1, L]$.

The search requires knowledge of where vehicle clusters are currently located in order to schedule them. As mentioned above, each cluster has a corresponding lane, an expected arrival time, a size, and a flow rate. We define A_i as the sequence of ordered clusters on the i-th lane and N_i as the cardinality of this sequence. $\vec{A} = (A_1, A_2, ..., A_L)$, denoting the sequences of ordered clusters across all lanes.

Our search also requires the actual state of the traffic signal controller. The search requires the current phase of the controller, which consists of two compatible green movements, which we define as $\vec{m}^a = (m_1^a, m_2^a)$. Additionally, the search requires the respective start times of each of these movements, which we define as $\vec{t^a} = (t_1^a, t_2^a)$.

3.5.2 Search States

We define a search state as a tuple: $(\vec{c}, \vec{m}, \vec{t}, T, g, h, f, \mathit{PS})$. $\vec{c} = (c_1, c_2, ..., c_L)$, where c_i represents the number of clusters that have been served on lane i. Because vehicle clusters cannot pass through each other on a lane, cluster order is maintained and the number of clusters that have been served on a given lane uniquely determines which clusters have been served.

A dual ring barrier controller requires that we keep track of two compatible green movements and their respective start times. $\vec{m} = (m_1, m_2)$ represents the currently active or upcoming green movements for the controller and $\vec{t} = (t_1, t_2)$ defines the respective start times of these movements. During an active green movement, m_i indicates the movement and t_i indicates when the movement began (to ensure we remain within minimum and maximum timing limits). During a yellow or red clearance interval, m_i indicates the next green movement that will occur after the clearance interval, and t_i indicates the time that this movement will begin. This is based on the end time of the previous green movement and the fixed yellow and red clearances.

T denotes the end time of the most recent scheduling decision (equivalently, T is also the duration, in seconds, of the underlying partial schedule). We note that during a green movement, $t_i \leq T$; during a yellow or red clearance, $t_i > T$ to represent that the green movement has not yet started. Each state also has a g value, which is the total delay incurred thus far, an h value which is an underestimate of the remaining delay, and an f value which is the sum the g value and h value. Furthermore, search states maintain the underlying partial schedule (*PS*), which is a set of phases and their respective start times. Once we determine the optimal goal state, we can use the underlying schedule to calculate projected downstream arrivals.

3.5.3 Start State

In our start state, no clusters in any lane have passed through the intersection, and by definition, $T = 0$. We initialize delay to 0 to minimize cumulative delay incurred from now into the future. The actual controller state (consisting of the movements, $\vec{m}^a$, and their respective start times, $\vec{t^a}$) is communicated by the controller. Our start state is thus:

$$(\vec{0}, \vec{m}^a, \vec{t^a}, 0, 0, \text{calcHeuristic(...)}, \text{calcHeuristic(...)}, \{\ \})$$

where calcHeuristic(...) is our function which calculates the heuristic value (explained in detail in section 3.5.6).

3.5.4 Goal State

A goal state requires that all clusters have been assigned a time to pass through the intersection. Formally, we require that $\vec{c} = (N_1, N_2, ..., N_L)$.

3.5.5 State Transitions

Our state transition algorithm is presented in Algorithm 1.

Algorithm 1 Calculate State Transitions

```
1: procedure CALCULATE STATE TRANSITIONS($\vec{c}_p$, $\vec{m}_p$, $\vec{t}_p$, $T_p$, $g_p$, $PS_p$, $\vec{A}$)
2:     children = { }
3:     possiblePhaseTransitions = getPhaseTransitions($\vec{c}_p, \vec{m}_p, \vec{t}_p, T_p, \vec{A}$)
4:     for each $(\vec{m}, \vec{t}, \Delta T) \in$ possiblePhaseTransitions do
5:         $\vec{c} = \vec{c_p}, T = T_p, g = g_p, PS = PS_p$
6:         $PS = PS \cup \{(\vec{m}, \vec{t})\}$
7:         $\Delta\vec{c}, \Delta$Delay = serveClusters($\vec{c}, \vec{m}, \vec{t}, T, \Delta T, \vec{A}$)
8:         $\vec{c} = \vec{c} + \Delta\vec{c}$
9:         $g = g + \Delta$Delay
10:        $T = T + \Delta T$
11:        $h$ = calcHeuristic($\vec{c}, \vec{m}, \vec{t}, T, \vec{A}$)
12:        $f = g + h$
13:        children = children $\cup \{(\vec{c}, \vec{m}, \vec{t}, T, g, h, f, PS)\}$
14:    return(children)
```

Our algorithm expands a parent state (whose elements are denoted with the subscript 'p'), calculating all possible child states to ultimately add to our search's open list. In addition to the parent state, our algorithm also requires arrival information, namely $\vec{A}$, to generate child states.

Our algorithm begins by initializing an empty set of child states (line 2). To calculate child states, our algorithm first must calculate possible traffic phase transitions from the parent state based on the traffic light controller logic (line 3). Each phase transition contains information regarding a possible future configuration of the controller, namely the active green movements, the respective green movement start times, and how much time to advance the search.

We present an example of such transitions in Figure 3.12. If both movements are left turn green movements (Figure 3.12, state A), we can transition to stay in the phase and step forward in time the duration necessary to serve the next vehicle cluster (3 seconds for state B) or instantaneously end one (states C and D) or both of the green movements (state E). When ending a green movement, we replace the current green movement with the next green movement according to the compatible phase ordering (as presented in Figure 3.3). We calculate the future start time of this green movement by adding the fixed duration of yellow and red clearances (5 seconds in our example) to the current time. Vehicles may not pass through the intersection in this direction during the transition.

When in a state with two green movements that are waiting to begin (state E), we step forward to the earlier start time (state G). However, if in a state where one green movement is active and the other is waiting to begin (state C), we cannot automatically jump forward. We must allow the current green movement to serve the next vehicle cluster (state F) and then branch on whether to maintain the current movement and advance the search forward in time (state H) or to end the active green movement (state I).

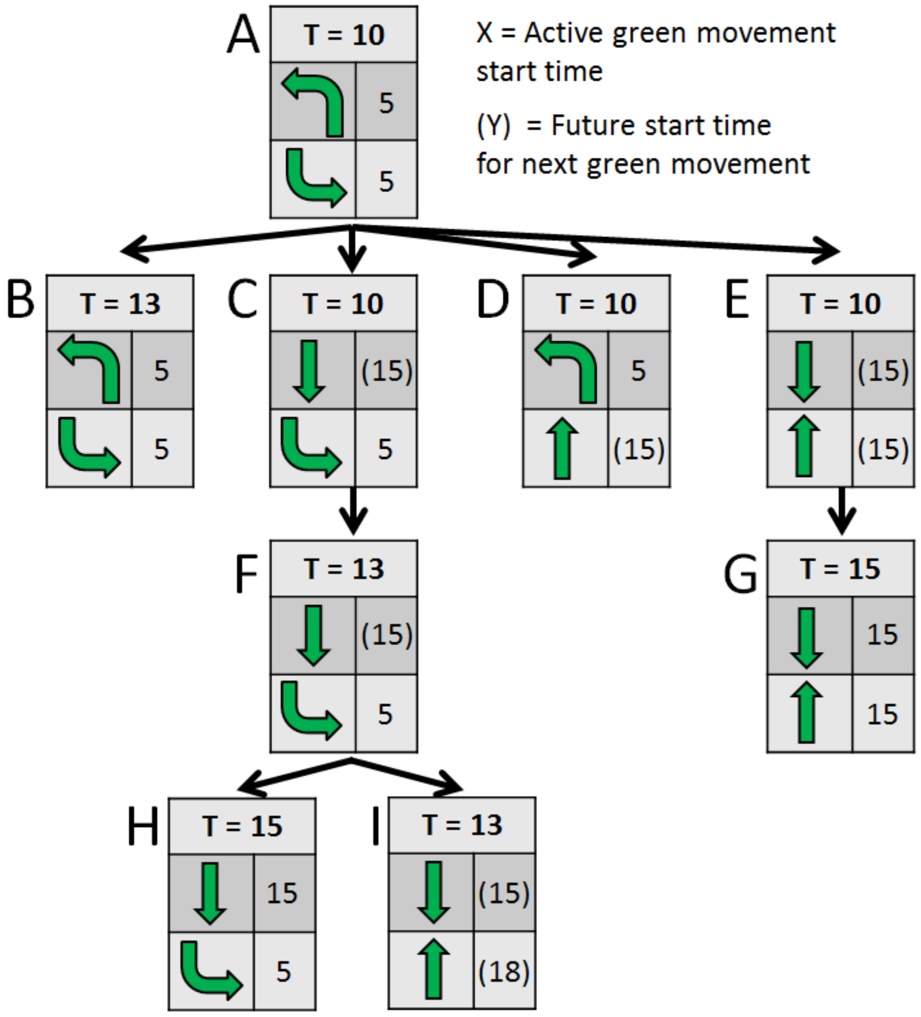


Figure 3.12: Phase Transitions Example

For each of these valid phase transitions (which consists of the active green movements, the respective start times, and how much time to advance the search), we will calculate one child state (Algorithm 1, line 4). To calculate a child state, we first recall parent state information as a starting point (line 5) and then perform several steps to calculate the child state. We add the new controller state to our overall schedule of phases (line 6). Given the new controller state and amount of time to advance, we calculate how many clusters (or partial clusters) will advance through the intersection along the active green movements of the intersection and the delay incurred by these clusters (line 7). This information is then used to update the cumulative number of clusters served and cumulative search state delay (lines 8 - 9). We next update the time step according to how much time we have advanced (line 10). Given the child state's new count of clusters served, controller state, and current search time, we calculate a lower bound heuristic estimate on the child state's remaining delay (line 11). We then add cumulative delay to the heuristic to obtain the child state's f value (line 12) and add the child state to our set of child states (line 13). This process then repeats for each possible controller phase transition. After all child states have been calculated, they are returned by the algorithm (line 14).

3.5.6 Heuristic Function

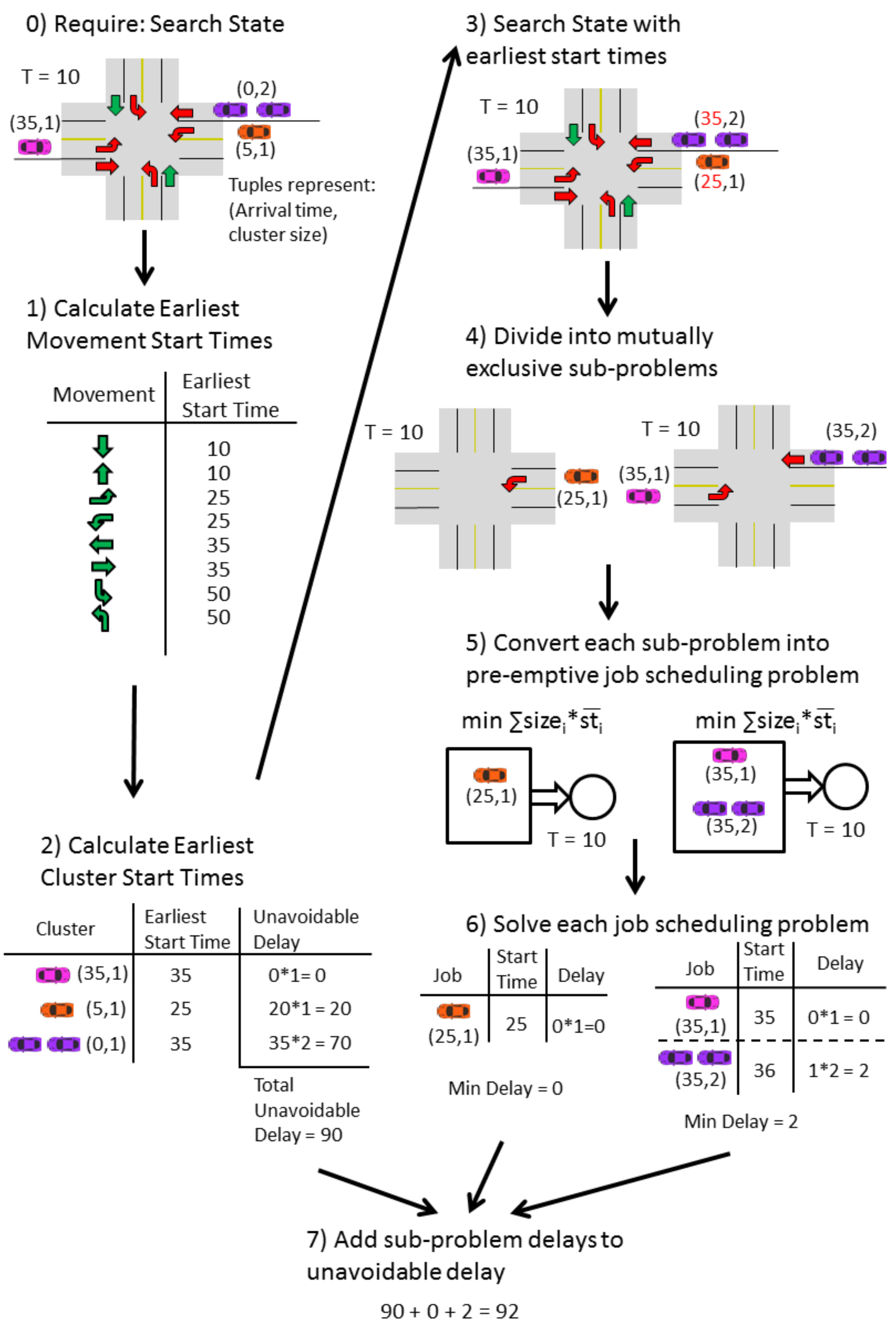
Crucial to any A* search is the effectiveness of the heuristic function. We apply a heuristic function that exploits problem structure to efficiently calculate lower bounds on the remaining delay that will be encountered by the unserved vehicle clusters. Our heuristic function begins by converting the problem into a tighter problem after calculating earliest cluster start times. We decompose this tighter problem into two independent, additive sub-problems and quickly solve a relaxation of each sub-problem as a pre-emptive job scheduling problem. This section describes the pipeline for our heuristic calculation in more detail. We present a graphic of the pipeline in Figure 3.13.

Our heuristic function operates on the current search state. In the example presented in Figure 3.13, we have three unserved vehicle clusters: a pink cluster with one vehicle, a purple cluster with two vehicles, and an orange cluster with one vehicle. Each cluster is travelling along a separate direction and has an expected arrival time at the intersection (e.g. 35 for the pink cluster, 0 for the purple cluster). In this example, we assume all flow rates are 1 vehicle per second and omit them for brevity.

Given information about which clusters are remaining, the current controller state, and the current world time, we first calculate the earliest possible start time for each movement. This involves using the fixed phase ordering, phase minimum times, and required yellow and red clearances. Additionally, we increase the earliest possible start time of the currently active movements to the current world time because clusters that have not yet been served cannot be served before the current world time.

For example, the north straight and south straight movements are currently active and began at time 10. They are assigned an earliest possible start time equal to the current time, which is also 10. The north straight and south straight directions have a minimum green time requirement of 10 seconds. Yellow and red clearances are a combined 5 seconds. Thus the next phases in sequence, namely the west left and east left phases, cannot begin before time 25.

In step 2, we calculate the earliest start time of each cluster. A cluster's earliest start time is the maximum of its arrival time at the intersection, the movement's earliest start time, and the end time of any preceding clusters along the same lane. Earliest cluster start times are also used to calculate unavoidable delays for each cluster. For example, the pink cluster arrives at time 35,


0) Require: Search State
T = 10
(35,1)
(0,2)
(5,1)
Tuples represent: (Arrival time, cluster size)
1) Calculate Earliest Movement Start Times
2) Calculate Earliest Cluster Start Times
3) Search State with earliest start times
T = 10
(35,1)
(35,2)
(25,1)
4) Divide into mutually exclusive sub-problems
T = 10
(25,1)
T = 10
(35,1)
(35,2)
5) Convert each sub-problem into pre-emptive job scheduling problem
min ∑size$_i$*$\overline{st_i}$
(25,1)
T = 10
min ∑size$_i$*$\overline{st_i}$
(35,1)
(35,2)
T = 10
6) Solve each job scheduling problem
7) Add sub-problem delays to unavoidable delay
90 + 0 + 2 = 92

Movement	Earliest Start Time
⬇	10
⬆	10
↱	25
↰	25
⬅	35
➡	35
↳	50
↲	50

Cluster	Earliest Start Time	Unavoidable Delay
(35,1)	35	0*1= 0
(5,1)	25	20*1 = 20
(0,1)	35	35*2 = 70
		Total Unavoidable Delay = 90

Job	Start Time	Delay
(25,1)	25	0*1=0

Min Delay = 0

Job	Start Time	Delay
(35,1)	35	0*1 = 0
(35,2)	36	1*2 = 2

Min Delay = 2

Figure 3.13: Heuristic Calculation Pipeline

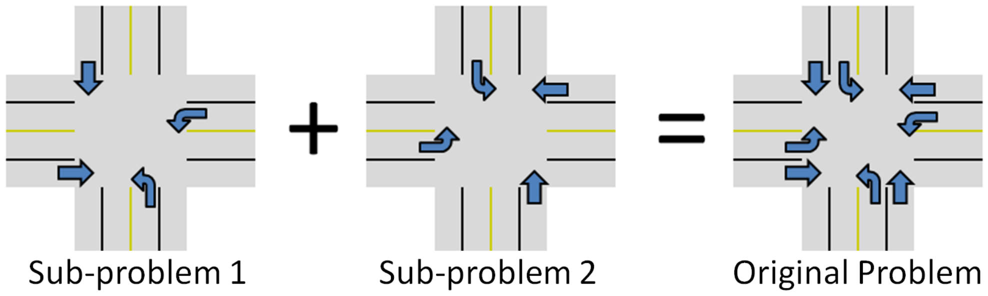


Figure 3.14: One Possible Sub-Problem Split

after the west left turn movement's earliest possible start time (25). The pink cluster maintains its earliest possible start time of 35 and experiences no delay. The purple cluster arrives at time 0, but cannot receive a green signal until time 35, meaning the earliest possible start time of each vehicle in the cluster is pushed back by 35 seconds. Thus, both vehicles in the purple cluster experience 35 seconds of delay for a total of 70 seconds of delay.

Once cluster start times are obtained, we replace the arrival times with the earliest start times in our original problem. As noted earlier, the pink cluster is not delayed so it maintains its prior start time. The purple cluster is delayed and is assigned the later start time of 35. This step makes the calculations from future steps tighter yet still admissible because no feasible start times are eliminated.

In step 4, we divide this new problem into two additive sub-problems based on non-compatible movements. Each sub-problem is composed of the clusters corresponding to green movements from exactly four of the eight directions. A graphic of one such split is presented in Figure 3.14. Observe that the four movements in sub-problem 1 are identical to the movements on the top ring (i.e. the top row) of the dual ring barrier presented earlier in Figure 3.2. Similarly, the four movements in sub-problem 2 are identical to the movements in the bottom ring (i.e. the bottom row) in Figure 3.2. While each phase in Figure 3.2 must be grouped with the adjacent one on the same ring on the same side of the barrier (phase 1 with phase 2, phase 3 with phase 4, etc.), there is nothing requiring phases 1 and 2 to be grouped on the same ring as phases 3 and 4. Phases 1 and 2 could instead be grouped with phases 7 and 8. This leads to a second possible sub-problem split where phases 1, 2, 7, and 8 make up one sub-problem while phases 3, 4, 5, and 6 make up the other sub-problem. The ERIS pipeline examines both combinations and takes the one with the maximum delay. In this example, we only consider the sub-problem split presented in Figure 3.14. We note that our weighting strategy proposed in Chapter 7 will also be influenced by the dual ring barrier structure.

Sub-problems are additive because no cluster appears more than once across the sub-problems. The movements within a sub-problem all conflict with each other; this provides the important property that within each sub-problem, at most, one movement can receive a green signal at any given time. For example, observe that in the right sub-problem, the pink vehicle turning left and the purple vehicles travelling straight cannot simultaneously receive a green signal.

Next, we relax each sub-problem by dropping several constraints: we ignore forced yellow and red clearances, phase orderings, and start-up lost time. This relaxation ensures fast calculation, and is not necessary to establish admissibility. Once these constraints are relaxed, our problem is equivalent to a single pre-emptive machine scheduling problem with no switch-over costs. Vehicle clusters become pre-emptible jobs with arrival times, sizes, and deterministic service rates.

In step 6, we solve each sub-problem with a polynomial time pre-emptive job scheduling algorithm, minimizing weighted average job start time (denoted $\overline{st}$), to calculate a lower bound on the delay of each original sub-problem. In the right sub-problem, the pink and purple clusters cannot both receive service at time 35. One possible optimal solution is to serve the pink vehicle first, followed by the purple vehicles. Each vehicle in the purple cluster is delayed by 1 second, leading to an overall sub-problem delay of 2 seconds.

In our final step, we combine these sub-problem delays with the unavoidable delay (from step 2) to obtain a lower bound on remaining delay for the original search state.

3.5.7 Closed List & Dominance Checks

A* search typically maintains a closed list to ensure that search states are not expanded more than once. Recall that when an A* search encounters a state already on the closed list, it does not expand the state and then progresses to the next state on the open list.

Our search states are defined by two discrete features, namely, the two movements of the controller, and several continuous features, including the fractional clusters (resulting from the algorithm's willingness to split clusters) completed on each lane, the world time, and the movement start times. Visiting the exact same state twice is less likely than in typical graph search as world time and clusters completed are monotonically increasing state features. Checking if a state is on the closed list is hence unlikely to return an exact match. For this reason, we do not run a closed list check as is typical in A* search.

ERIS performs dominance checks on states prior to expansion. It is inefficient to compare a state to expand against all previously expanded search states to check for dominance. Thus, we instead save expanded states into 'buckets.' Buckets are indexed by a combination of controller state ($\vec{m}$) and a non-negative integer corresponding to schedule length (T). Each expanded state is saved into the appropriate bucket matching the state's $\vec{m}$ and matching the state's truncated (to the nearest second) value for T.

When deciding whether or not to expand a new state, ERIS only compares this state to states in the same bucket and states in nearby buckets, specifically those with identical values of $\vec{m}$ and slightly lower (up to 10 seconds lower) values of T. (We only examine buckets with equal or slightly lower values of T because we consider it unlikely that a search state with a much lower value of T will have served more vehicle clusters and thus be dominating; we do not examine states with higher values of T because they cannot be dominating).

If the search identifies a state having served at least as many clusters in each direction, having an equal or lower cost, and weakly dominating in terms of timing flexibility (defined below), the candidate state is not expanded. If the state is expanded, it is then saved in the appropriate bucket to be checked against future states to expand.

Examining timing flexibility domination requires evaluating several logical clauses. State A weakly dominates State B in terms of timing flexibility when four of the below clauses are true (only one of 3a or 3b can be true; only one of 4a or 4b can be true). The clauses are as follows:

1 The states have equal $\vec{m}$.

2 State A has a lower or equal value of T.

3a State A's first active movement began at the same time as State B's first active movement.

3b State A's first active movement is not below the minimum phase duration and began at a later time than State B's first active movement. (Observe that if State A's first active movement is under the minimum phase duration, it could be optimal to end the phase as soon as possible and thus State B has flexibility that State A does not).

4a State A's second active movement began at the same time as State B's second active movement.

4b State A's second active movement is not below the minimum phase duration and began at a later time than State B's second active movement.

3.6 Experimental Evaluation on Synthetic Networks

This section and the following section present our experimental evaluation of ERIS. This section focuses on evaluation of ERIS on synthetic traffic networks. The following section will present experimental evaluation of ERIS on a simulated real-world traffic network.

In this section, we begin by explaining our assumptions about vehicle data. Next, we introduce our baselines. We then explain several implementation details specific to our simulation. Finally, we present our results on the synthetic traffic networks.

3.6.1 Perfect Information Connected Vehicle Environment

Our experimental evaluation assumes a perfect information connected vehicle environment. Namely, we assume that all vehicles use a connected device to share their position, speed, and heading every time step. The Cluster Construction module uses this information to learn models that estimate when each vehicle will reach the intersection and which lane the vehicle will be in prior to entering the intersection. Using this information, nearby vehicles are then grouped into clusters along each lane of the intersection for use by the Scheduling Module.

ERIS and other scheduling approaches can also be applied to non-connected vehicle frameworks where cameras or other detectors estimate vehicle locations. Different cluster construction algorithms are then applied to generate an estimate of vehicle clusters along each side of the intersection. Using probe vehicles or noisy data to generate clusters is a difficult problem that has been addressed in a variety of other works and is beyond the scope of this work [3, 22, 72]. These cluster construction algorithms lead to less accurate vehicle cluster estimations, and as a result, schedules may provide green signals for non-existent vehicles.

Here, we present comparisons only in the fully connected vehicle framework to reduce the impacts of noisy cluster estimation to allow us to focus on comparing the quality of the scheduling algorithms. All algorithms employ the same detection and cluster construction models.

Additionally, we note that the vehicles in our simulations use SUMO's default car following model, the Krauss Model [39]. When a vehicle follows another vehicle, it updates its speed according to the distance between itself and the leading vehicle, the speed of the leading vehicle, and an individual reaction time parameter.

3.6.2 Baselines

In this section, we compare ERIS to two other real-time scheduling methods in a perfect information connected vehicle environment. First, we compare ERIS to a real-time scheduling method based on Surtrac2012 [69]. We do not run the original Surtrac2012 algorithm, but rather, run a simplified version of ERIS that makes the same assumptions as Surtrac2012 and returns the same schedule that Surtrac2012 would return. We refer to this method as SynSurtrac2012 (synthetic Surtrac2012). Second, we compare our methods to a variant of fully actuated control, which we refer to as connected fully actuated control.

In a connected vehicle environment, comparing to traditional fully actuated control is unfair as the phase must wait several seconds to end, but the controller could have used information about future arrivals (i.e. knowing no vehicles would arrive in the next 3 seconds) to end earlier. In connected fully actuated control, we assume that vehicles only send a message to an intersection when they are within a fixed number of seconds (e.g. 3 seconds) of the intersection. If a message is received, the movement continues; otherwise, the movement ends. Using these models as comparisons will allow us to present a more competitive comparison than using simpler methods such as traditional fully actuated control or fixed timing plans that have previously been shown to significantly underperform Surtrac2012 [69].

Simulations in this section run for an hour of simulated time. For each simulation, we report average delay per vehicle. Note that these calculated values of delay also include other factors that we do not aim to improve, such as the delay resulting from driving behind a vehicle with a lower cruising speed. To eliminate the effects of simulation start up and termination, we only report the delay of vehicles arriving within the middle 40 minutes. Results for a given experiment are averaged across ten simulation runs with different random seeds. Identical random seeds are used when comparing the three methods to generate identical traffic arrivals. Vehicle arrivals are modelled as a Poisson process where the average arrival rate is set according to the desired level of congestion.

We present a comparison of these methods on two separate networks, a discussion of statistical significance, and a timing analysis on components of the Scheduling Module.

3.6.3 Simulation Implementation Details

In this subsection, we detail two implementation details that apply to all evaluations conducted in this thesis.

When deploying ERIS to a network of intersections, each intersection in the network would run its own localized process on a different machine in real-time (under 1 second). Every time step, each machine would asynchronously calculate a schedule, communicate results to downstream neighbors, and then execute this schedule.

For our experimental evaluations, a single machine conducts all calculations, including the calculation of each schedule. We do not require that the total duration required to conduct all of these calculations during a single time step is under one second (though in many cases, this duration is under one second). Rather, immediately before calculating schedules, we suspend time. The simulation then proceeds through the calculations of schedules at each intersection in a fixed sequence. Each intersection is provided up to 1 second to conduct its evaluation. Once all schedules are obtained, time is resumed, and all machines execute the first step of their schedule.

Additionally, in our implementation, we assume that all communication is instantaneous. Specifically, projected downstream vehicle arrivals are instantaneously shared with neighboring intersections so that they can be incorporated in the cluster construction module for the following time step.

Instantaneous communication likely improves the quality of schedules that are calculated compared to what we would expect when ERIS would be deployed in the field. (When deploying ERIS in the field, when the projected downstream arrivals are not received in time, projected downstream arrivals from an earlier time step could be used instead). However, we do not believe that in our evaluations instantaneous communication provides a significant advantage to ERIS because SynSurtrac2012 also benefits from instantaneous communication.

3.6.4 Single Intersection

We first examine ERIS on a single intersection with one joint straight and right turn lane and one dedicated left turn lane on each of its four sides (identical to the intersection presented in Figure 3.1). The left turn lanes are only serviced by a protected left turn signal (we do not allow permissive left turns).

Average vehicle delay (in seconds) and standard errors across a range of vehicle volumes are presented in Table 3.1. We also present ERIS' improvement when measured against the better of SynSurtrac2012 and connected fully actuated control at each level of congestion.

Vehicles/Hour	160	400	800	1200	1600
ERIS	18.4 ± 0.4	25.5 ± 0.4	33.0 ± 0.3	39.9 ± 0.3	49.6 ± 0.4
SynSurtrac2012	22.9 ± 0.6	28.7 ± 0.4	35.1 ± 0.3	41.9 ± 0.4	51.3 ± 0.7
Actuated	26.8 ± 0.6	29.6 ± 0.3	35.1 ± 0.3	41.5 ± 0.3	51.1 ± 0.5
% Improvement	20.0%	11.2%	5.8%	3.8%	3.0%

Table 3.1: Average Vehicle Delay (s) for a Single Intersection

ERIS outperforms SynSurtrac2012 and connected fully actuated control for all tested levels of congestion. ERIS' performance improvement is largest, at 20.0%, at the lowest level of congestion. With light and medium congestion, more exact vehicle information is useful to exactly place clusters of vehicles into a schedule of green phases. It is important to know if two vehicles are arriving on the same lane or different lanes. If they are arriving on different lanes, it is possible to schedule them simultaneously once both have arrived, but, as discuss above, Surtrac2012's simplified clustering method groups these clusters to be served in sequence.

With increased congestion, the value of exact cluster estimation diminishes and ERIS outperforms the baselines by 3.0% and 3.8% in the two most congested scenarios. It is more likely that there are vehicles waiting in queues at the stop line on different lanes of the intersection.

SynSurtrac2012's clustering simplifications are more accurate for queued vehicles. Similarly, when vehicles form queues, fully actuated control holds green movements until queues empty, which is often the best strategy in high congestion situations.

3.6.5 3 x 3 Grid of Intersections

In this experiment, we model a 3 x 3 grid of intersections. As before, each intersection has one joint straight and right turn lane and one dedicated left turn lane on each of its four sides. Additionally, intersections share their projected vehicles outflows with neighboring intersections. ERIS and SynSurtrac2012 utilize an identical strategy for sharing information about projected outflows. Specifically, after calculating a schedule, ERIS and SynSurtrac2012 calculate the actual departure time of each cluster to share with neighboring intersections. For situations where SynSurtrac2012's scheduler incorrectly assumes a later start time for a specific cluster (due to the combining compatible movements simplification described in Section 3.3), the corrected (earlier) start time is instead calculated and shared with neighboring intersections.

Average vehicle delay (in seconds) and standard errors are presented in Table 3.2.

Vehicles/Hour	500	1300	2700	4000	5300
ERIS	39.4 ± 0.3	55.6 ± 0.6	78.3 ± 0.5	103.8 ± 0.6	146.0 ± 1.4
SynSurtrac2012	48.5 ± 0.5	63.8 ± 0.7	85.1 ± 0.4	109.4 ± 0.6	146.7 ± 1.3
Actuated	57.8 ± 0.3	67.5 ± 0.5	85.3 ± 0.6	110.1 ± 0.5	149.0 ± 1.6
% Improvement	18.9%	12.8%	7.9%	5.2%	0.5%

Table 3.2: Average Vehicle Delay (s) for a 3 x 3 Grid

We observe similar results to the above experiment. ERIS outperforms the two comparison algorithms on all levels of congestion. This improvement is largest, at 18.9%, for the least congested setting. This experiment demonstrates that the advantages of ERIS extend to network settings.

3.6.6 Statistical Significance

We perform 20 two-sided paired t-tests with a 95% confidence level comparing ERIS' performance with each of the other methods. Each test compares ERIS' average vehicle delay to a competing method's average vehicle delay at a single congestion level on a single network. (We did not compare SynSurtrac2012 to connected fully actuated control and we did not compare across different congestion levels). We applied the Holm-Bonferroni method to maintain an overall error rate of 5% [30]. In 19 of the 20 comparisons (all except ERIS vs SynSurtrac2012 for 5300 vehicles/hour in the 3 x 3 grid setting), ERIS' average vehicle delay is statistically significantly different than the comparison method's average vehicle delay.

3.6.7 Timing Analysis

This subsection presents a brief demonstration of the importance of the heuristic function and the minor importance of the dominance checks. Table 3.3 presents runtime performance percentiles in the most congested (and hardest) case for the single intersection model. All timings are presented in milliseconds. These experiments were conducted on a Dell E7450 laptop with a 2.2-GHz Intel Core i5-5200U CPU, and 8GB of RAM.

Method	Mean	25%ile	50%ile	75%ile	90%ile	95%ile
Dominance Checks & Heuristic	3.6	2.0	2.8	4.4	6.7	8.5
Heuristic Only	4.1	2.3	3.2	5.1	7.7	9.5
Dominance Checks Only	105.0	7.6	17.1	43.8	114.1	237.2

Table 3.3: Runtime Comparisons (ms), Single Intersection

Running our scheduling method with both dominance checks and the heuristic is fairly efficient, requiring under 9 milliseconds 95% of the time for the congested single intersection model. Applying dominance checks provides a small speed-up over only using the heuristic function. The heuristic function is quite important. Without using the heuristic function, ERIS requires an order of magnitude increase in runtime to find the optimal schedule. This increase in runtime is larger for more complicated problems.

3.7 St. Albert Network

In this section, we present several additional experiments where we examine ERIS on a simulated traffic network consisting of 11 intersections from St. Albert, a city in Alberta, Canada. We describe the network then present four experiments.

3.7.1 Network Overview

The St. Albert traffic network was provided by Rapid Flow Technologies who have deployed an updated version of Surtrac along the 11 intersections of the network. This is a corridor network with 10 intersections along the corridor (Boudreau Road) and an additional intersection just off of the corridor. A sketch of this network, with select roads labelled, is presented in Figure 3.15 below.

We have based our model of this network on the provided model from Rapid Flow Technologies. Similarly, we obtained measured demand information for the PM rush hour from Rapid Flow Technologies. Unfortunately, the demand data was somewhat inaccurate; we also obtained videos matching the timestamps of our demand data and manually counted vehicles, one movement at a time. We observed that our manual counts were at times as much as 50% higher as the obtained data. Additionally, we observed long queues at the southern approach (both the left turn movement and the straight movement) of the intersection of Boudreau and Campbell (indicated with a red circle on Figure 3.15). We also observed the intersection of Campbell and Carswell

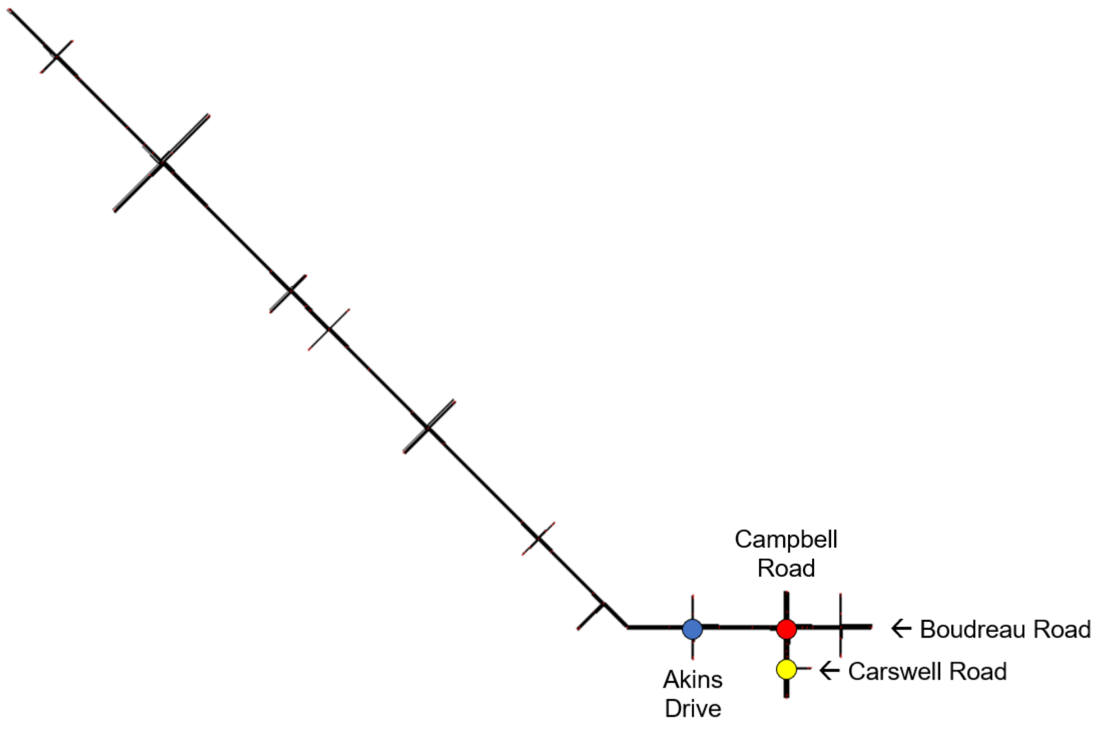


Figure 3.15: St. Albert Network

(indicated with a yellow circle). Here, we noticed that vehicles travelling along the southern approach were occasionally unable to proceed through this intersection because the the southern approach to Boudreau and Campbell was at capacity. In our simulation, we noticed qualitatively similar behavior when scaling the provided demand up by factors of 1.43 and 1.5. As such, we chose to compare the algorithms at these levels of demand. We also selected two lower levels of demand, namely scaling factors of 1.25 and 1.1.

In addition to comparing ERIS against SynSurtrac2012, we obtained the network's fixed timing plan to also compare against. We implemented the fixed timing plan as described in the provided Econolite documents for the PM rush hour (4PM - 6PM), but are suspicious of the efficacy of the fixed timing plan. Specifically, most of the intersections had a common cycle time of 120 seconds, and thus when implemented with the provided offsets, coordination between these intersections was observed in our simulations. However, three intersections along the western portion of the network had a different cycle time. Travelling from west to east along Boudreau Road, cycle times were respectively: 115 seconds, 120 seconds, 135 seconds, 130 seconds, 120 seconds, 120 seconds, 120 seconds, 120 seconds, and 120 seconds. (The provided plan did not provide a cycle time for the tenth intersection, and thus, the cycle time is not listed above).

While we compare to this fixed timing plan, we are doubtful that it is the ideal fixed timing plan (for example, we expect that a fixed timing plan where all intersections have a fixed cycle time of 120 seconds would work better).

We note that several intersections in this network are not 8-phase intersections. For example, the intersection of Boudreau Road and Akins Drive (indicated with a blue circle) does not have dedicated left turn lanes or dedicated left turn phases for the side streets. We utilize our

Scaling Factor	1.1	1.25	1.43	1.5
Approximate Vehicles/Hour	4500	5200	5900	6200
ERIS	57.2 ± 0.3	62.9 ± 0.6	74.6 ± 1.4	78.3 ± 0.9
Fixed Timing Plan	93.9 ± 0.6	101.1 ± 1.1	117.6 ± 2.5	129.9 ± 2.4
SynSurtrac2012	59.2 ± 0.4	66.1 ± 0.9	81.8 ± 1.3	87.9 ± 1.5
ERIS Improvement vs SynSurtrac2012	3.3%	4.8%	8.7%	10.9%
ERIS Improvement vs Fixed Timing Plan	39.1%	37.8%	36.5%	39.7%
SynSurtrac2012 Improvement vs Fixed Timing Plan	37.0%	34.7%	30.5%	32.3%

Table 3.4: Average Vehicle Delay (s) on St. Albert Network - Hour-long Experiments

underlying dual ring barrier controller (discussed in Section 3.2) but make several modifications:

1. We set the minimum and maximum phase times to 0 for the unused phases.
2. We modify the straight phases to also have permissive green signals for left turning vehicles.
3. We assume a slightly lower flow rate for the vehicles travelling on shared left turn and straight lanes.

In the remainder of this section, we present several experiments. First, we compare ERIS, SynSurtrac2012, and the provided fixed timing plan across hour long runs at a variety of different levels of demand. We then present an experiment of three-hour long runs in which we allow demand to gradually increase at the beginning of the simulation and then decrease at the end of the simulation. Third, we consider four differences between ERIS and SynSurtrac2012 and analyze how runtime and solution quality is impacted by each difference. Finally, we compare ERIS and SynSurtrac2012 when we assume some fraction of the vehicles are not connected, and thus, will not be included in the vehicle clusters.

3.7.2 Hour-long Experiments

We begin by presenting hour-long experiments run on the St. Albert network. As in experiments from the previous section, we run 10 trials for each demand and scheduling method combination and report the average delay for vehicles arriving to the network in the middle 40 minutes of the simulation. Average vehicle delay (in seconds) and standard errors for each demand and method pair are presented in Table 3.4.

We observe that both ERIS and SynSurtrac2012 reduce delay, when compared to the fixed timing plan, at all levels of congestion by between 30% and 40%. This is not surprising given our earlier concerns about the fixed timing plan not using a common cycle time. We also notice that ERIS reduces delay, when compared to SynSurtrac2012, at all levels of congestion. This improvement is largest (10.9%) at the highest level of congestion and smallest (3.3%) at the lowest level of congestion. Recall that in the prior experiments, we noticed that ERIS' improvement over SynSurtrac2012 was largest at the lowest, not the highest, levels of congestion. We propose two reasons for this reversal. First, in this network, left turns begin protected then switch to being permissive while in the previous networks, left turns were protected and then forbidden. In this network, ERIS' ability to hold a phase a little longer to squeeze in a left turning vehicle is less important because these vehicles will likely be able to complete their turn during the permissive

Scaling Factor	1.1	1.25	1.43	1.5
Approximate Vehicles/Hour	4500	5200	5900	6200
ERIS	1.24 ± 0.02	1.33 ± 0.03	1.51 ± 0.06	1.56 ± 0.05
Fixed Timing Plan	1.92 ± 0.03	2.04 ± 0.07	2.30 ± 0.14	2.52 ± 0.12
SynSurtrac2012	1.23 ± 0.02	1.30 ± 0.04	1.51 ± 0.06	1.58 ± 0.06
ERIS Improvement vs SynSurtrac2012	-1.4%	-2.8%	-0.2%	1.6%
ERIS Improvement vs Fixed Timing Plan	35.4%	34.8%	34.3%	38.1%
SynSurtrac Improvement vs Fixed Timing Plan	36.3%	36.6%	34.4%	37.0%

Table 3.5: Average Number of Stops on St. Albert Network - Hour-long Experiments

phase (and not wait a full cycle for their next protected phase).

Second, in the experiments presented in the previous section, demands across the different approaches were roughly equivalent and we argued that during high congestion, subtlety in scheduling wasn't too important - simply providing a green signal when a vehicle was present was a reasonable strategy. Here, we observed that the demand was unequal across different sides of many intersections. For example, at the busiest intersection (Boudreau Road and Campbell Road), the southern and western approaches were much busier than the northern and eastern approaches. Because it uses a lane-based framework, ERIS is able to differentiate between vehicles along each approach while SynSurtrac2012 does not (due to its cluster folding procedure). Similarly, the intersection has two west straight lanes and one west right lane. ERIS is able to keep these clusters separate while SynSurtrac2012 does not keep these clusters separate. SynSurtrac2012 keeps green phases active for too long because it does not know that when there is a single active (green) movement with vehicles, if it waits to serve these vehicles until the next cycle, it will often be able to serve vehicles along other movements concurrently. As discussed in Section 3.3, SynSurtrac2012 overestimates the delay for some optimal schedules, does not select these schedules, and ultimately, this leads to queue buildup and higher delay.

When optimizing one criteria, one should consider whether another criteria may be worsened. For this reason, we also examine the average number of times that each vehicle stops. The average number of stops and standard errors for each demand and method pair are presented in Table 3.5.

First, we note that ERIS and SynSurtrac2012 both reduce the average number of stops when compared to the fixed timing plan, at all levels of congestion, by between 30% and 40%. This reduction in the average number of stops corresponds closely to the reduction in delay that we discussed earlier.

Next, we compare ERIS and SynSurtrac2012. We observe a relatively minor, yet statistically significant (when conducting a two-sided paired t-test with a 95% confidence level), increase in the average number of stops when comparing ERIS against SynSurtrac2012 at one level of demand (5200 vehicles/hour). However, we note that this increase is relatively minor (0.03 additional stops) and only occurs at this single level of demand. There are no statistically significant differences between the average number of stops at other levels of demand (when conducting two-sided paired t-tests with a 95% confidence level). Due to the low magnitude of all of these differences, we conclude that optimizing delay did not materially increase the average number of times that each vehicle stops.

We also present the runtimes for ERIS and SynSurtrac2012 at the different levels of congestion; these measurements are based on runs conducted on AWS EC2 t2.micro Amazon Linux 2 instances. We present average runtimes (Table 3.6), 95th percentile runtimes (Table 3.7) and the percentage of solves that require over one second (Table 3.8). Note that these runtimes are higher than the single intersection runtimes presented in Table 3.3 primarily because the intersections in the St. Albert network share projected outflows, leading to a longer planning horizon. (We also conduct these measurements on different hardware which is also a factor).

Approximate Vehicles/Hour	4500	5200	5900	6200
ERIS	15.1	19.9	39.0	51.1
SynSurtrac2012	2.0	2.2	2.5	2.6

Table 3.6: Average Runtimes (ms)

Approximate Vehicles/Hour	4500	5200	5900	6200
ERIS	50.6	71.6	156.9	210.5
SynSurtrac2012	4.8	5.4	6.7	7.1

Table 3.7: 95th Percentile Runtimes (ms)

Approximate Vehicles/Hour	4500	5200	5900	6200
ERIS	0.0003%	0.004%	0.07%	0.18%
SynSurtrac2012	0.00%	0.00%	0.00%	0.00%

Table 3.8: Percentage of Runs over 1 Second

We notice that at the busiest level of congestion, ERIS requires roughly 20 times as long as SynSurtrac2012, 51 ms compared to 3 ms (Table 3.6). We observe a larger difference when comparing 95th percentile runtimes at the highest level of congestion (Table 3.7). 95% of the time, SynSurtrac2012 can calculate an optimal schedule in 7.1 ms or less. ERIS calculates a schedule in under 211 ms 95% of the time. This is still within our real-time requirement of 1 second, but much higher than SynSurtrac2012's 95th percentile runtimes. Finally, we never observe that SynSurtrac2012 requires over 1 second to calculate an optimal schedule, but at the busiest level of congestion, ERIS requires over 1 second roughly 1 out of every 500 runs (Table 3.8). For a real-time system deployed in the field, ERIS has several options that could allow it to return a solution in under 1 second. First, one could turn off ERIS' cluster splitting funtionality; we discuss this later in this section. Alternatively, if one notices ERIS often requiring over 1 second, one could run a weighted A* search instead. Weighted A* search has no guarantees about reducing solve time, but tends to work well in practice with minimal reduction to solution quality.

Scaling Factor	1.43
Approximate Vehicles/Hour	5900
ERIS	75.1 ± 0.8
SynSurtrac2012	80.5 ± 1.0
ERIS Improvement vs SynSurtrac2012	6.8%

Table 3.9: Average Vehicle Delay (s) on St. Albert Network - Three Hour-Long Experiment

3.7.3 Three Hour-Long Experiment

We also ran a three hour-long experiment to determine whether using additional burn-in time altered our results. This experiment was conducted at the second highest level of demand (5900 vehicles per hour) because it is our best estimate of the realistic traffic for this network. We scaled up demand linearly, beginning at 50% of the maximum at time 0, reaching the maximum demand at time 30 minutes (0.5 hours), remaining at this maximum demand until time 150 minutes (2.5 hours) and then scaling down demand linearly to 50% of the maximum demand by the end of the simulation (3 hours). Delay was measured for vehicles arriving between the 45th minute and the 135th minute. We noticed similar results to the above experiment, demonstrating that the one-hour long runs seem to provide adequate burn-in. Average vehicle delay and standard errors are presented in Table 3.9.

3.7.4 ERIS Setting Examination

In this subsection, we highlight four differences between ERIS and Surtrac2012 and measure how each of these differences impacts average delay and runtime. Three of these choices are binary, while the fourth has three possible options. We run $2^3 * 3 = 24$ total runs, one for each combination, and report average delay and runtime for each feature across the 8 or 12 associated runs. The experiment set-up matches that from the Hour-long Experiments Section with the exception that this experiment was conducted solely at the second highest level of demand (5900 vehicles per hour).

Lane-Based Clustering

As mentioned throughout this chapter (an example is presented in Section 3.3), ERIS keeps track of the clusters on each lane separately while Surtrac2012 folds clusters on compatible movements into larger cluster sets to reduce the search space. We present the delay and runtime of these two options in Table 3.10 below. We notice a substantial decrease in delay of 8.9% from keeping clusters separate with a very minimal increase in average runtime (an under 1 millisecond increase). Of the four modifications we present in this section, this provides the majority of the reduction in delay while also providing the smallest increase in runtime. Thus, we consider ERIS' decision to use lane-based clustering (and to not engage in the folding of clusters) to be the most important difference between ERIS' model and Surtrac2012's model.

	Delay (in s)	Runtime (in ms)
Lane-Based Clustering	77.6	14.9
Fold Compatible Clusters	85.2	14.4
Percent Change	-8.9%	3.3%

Table 3.10: Clustering Strategy Comparison

Scheduling Left Turns and Dropping Left-turning Clusters

We now consider the second and third modifications - whether to schedule left turns or run them actuated and whether to schedule all left turn clusters.

Surtrac2012 runs left turns as actuated. If there is at least one vehicle in the left turn queue, then the phase will activate until there are no more vehicles at the intersection (or the maximum time is reached). However, ERIS has the ability to be less restrictive with how it handles left turns. It could extend left turn phases even if there are no vehicles currently present because it knows that vehicles will arrives in the next few (5 - 10) seconds. Alternatively, it could end left turn phases earlier - after the minimum time has been satisfied, even if there are still vehicles present. ERIS has three settings for this options - the Surtrac2012 setting of running left turns as actuated, the ‘partial scheduling’ setting where ERIS must serve vehicles if they are present, but also may extend the phase if vehicles will be arriving later, and finally, the ‘full scheduling’ setting where ERIS both may extend the phase to serve vehicles arriving in the future while also cutting the phase if vehicles are still present (in this setting, the scheduler does not differentiate between left turns and other movements).

The third modification involves whether or not to schedule all clusters turning left. While not discussed in the Surtrac2012 papers, a review of the Surtrac2012 code (conducted in 2017) showed that Surtrac2012 drops all but the first left turning cluster along each left turn movement. The code schedules the first cluster as a regular cluster but does not schedule additional clusters. We examine the importance of keeping and scheduling all left turn clusters.

Average delay and average runtime from these settings are presented in the two tables below (Table 3.11 for how to schedule left turns, Table 3.12 for dropping additional left turn clusters). We notice a small, but not statistically significant (when conducting a two-sided paired t-test with a 95% confidence level), reduction in delay for scheduling left turns (1.4%) and also from not dropping additional left turn clusters (1.1%). These methods lead to relatively large increases in runtime of 57% and 79% but practically this is under a 10 ms increase in runtime. Thus, we recommend running ERIS with these settings.

	Delay (in s)	Runtime (in ms)
Full Scheduling for Left Turns	81.0	18.7
Partial Scheduling for Left Turns	81.1	13.2
Actuating Left Turns	82.1	11.9
Percent Change (Full Scheduling vs Actuating Left Turns)	-1.4%	57.1%

Table 3.11: Left Turn Scheduling Strategy Comparison

	Delay (in s)	Runtime (in ms)
Schedule all Left Turn Clusters	80.9	18.8
Only Schedule First Left Turn Cluster	81.9	10.5
Percent Change	-1.1%	78.6%

Table 3.12: Dropping Left Turn Clusters Comparison

Aggressive Clusters Splitting

The final modification considers a more aggressive cluster splitting procedure. We remind the reader that our ERIS solver splits clusters when two compatible movements have clusters that are different length (which is something Surtrac2012 does not do). For example, suppose there is a cluster of 10 vehicles that will require 20 seconds to serve along the east straight movement and a cluster of 5 vehicles that will require 10 seconds to serve along the west straight movement. If these movements receive a green signal at the same time, a single ERIS step may advance forward in time the minimum duration required to serve either of these clusters, namely 10 seconds. The west straight cluster will be marked as complete and the east straight cluster will be marked as half-complete.

ERIS can also engage in more aggressive cluster splitting by limiting the maximum step size (in time) between search states to 3 seconds. In our example from above (assuming no minimum time constraints), the clusters can be cut after serving 3 seconds, after serving 6 seconds, after serving 9 seconds (assuming there is no minimum phase time that needs to be satisfied), and as before, after serving 10 seconds. This setting allows ERIS to examine a larger set of possible schedules, but at the expense of additional branching. (Aggressive cluster splitting is required for green waves, which we discuss in more detail in Chapter 4).

Table 3.13 presents the impact of the different cluster splitting strategies. Aggressive cluster splitting leads to a small decrease in delay of 1.4% (not statistically significant) but at the expense of an approximately 400% increase in runtime. If ERIS is not running in real-time, one may wish to first consider deactivating aggressive cluster splitting.

	Delay (in s)	Runtime (in ms)
Aggressive Cluster Splitting	80.8	24.4
Regular Cluster Splitting	82.0	4.9
Percent Change	-1.4%	401.3%

Table 3.13: Cluster Splitting Comparison

3.7.5 Relaxing the 100% Connectivity Assumption

Throughout this chapter, we have assumed that all vehicles run a connected device which shares information with nearby intersections so that intersections can use this information in scheduling clusters. However, we note that vehicle connectivity is likely to increase gradually, and thus, we are interested in understanding what penetration of connectivity is necessary for ERIS to function satisfactorily without employing additional detection.

We conduct an experiment to examine the impact on average vehicle delay when only a fraction of the vehicles share their information with the network; our experiment does not utilize any other detection, meaning that some vehicles are never detected. We run 25 trials for each connectivity level for ERIS on the St. Albert Network. Average vehicle delay (in seconds) and standard errors are presented in Table 3.14.

Fraction Connected	100%	90%	75%	50%	25%
ERIS	74.5 ± 0.7	73.9 ± 0.8	73.7 ± 0.7	76.3 ± 0.8	89.6 ± 1.0
Delay Change vs 100% Connectivity	0.0%	-0.9%	-1.2%	2.3%	20.2%

Table 3.14: Average Vehicle Delay (s) from Different Connectivity Penetrations

First, we notice that delay is approximately unchanged as the percentage of connected vehicles decreases from 100% to 75%. The delay from 50% connectivity is slightly (2.3%) higher than the delay from 100% connectivity; this difference is statistically significant when performing a two-sided paired t-test with a 95% confidence level. The delay at 25% connectivity is 20% higher than than 100% connectivity and this difference is also statistically significant.

Thus, this experiment suggests that ERIS could be successfully deployed with 50% or higher connected vehicle penetration without the need for additional detection. At 50% penetration, we would expect a small decrease in performance compared to 75% (or higher) penetration.

3.8 Conclusion

This chapter presented ERIS, a real-time schedule-driven adaptive traffic scheduling framework. ERIS' increased flexibility and powerful A* search heuristic allow it to quickly make intelligent traffic scheduling decisions.

We began by describing the overall control flow of ERIS and explained our clustering methods. We detailed the underlying scheduling model which has a more detailed search space than past models. We explained our usage of A* search and the A* heuristic function that allow schedules to be calculated in real-time. We then presented experiments on simple networks that demonstrated that ERIS can reduce vehicle delay by as much as 20% when compared to other methods. We conducted additional experiments on a simulated model of a real-world network and observed that ERIS offers an improvement when compared against the baselines. We compared and contrasted ERIS to SynSurtrac2012 and determined that the most beneficial difference (in terms of reducing delay) was keeping clusters separate. Finally, we showed that ERIS, even without additional detection, can still reduce delay when the penetration of connected vehicles is 50% or higher.

This chapter made several modelling assumptions. Excluding the final experiment, we assumed a fully connected world where all vehicles noiselessly communicate their position, speed, and heading every second that can then instantaneously be used by ERIS to calculate an optimal decision for the next time step. While this was a useful assumption for demonstrating the improvement of ERIS over other scheduling methods, it is not realistic. On-board GPS data can be imprecise; according to the US Department of Defense (2001), 95% of the time, GPS readings will be within a 7.8 meter radius. The average lane is 3 meters wide; this margin of error means that we might estimate the wrong lane for a vehicle. Additionally, with realistic communication protocols, messages will be dropped and ERIS will need to make scheduling decisions on stale data. Though it is not discussed in this thesis, we acknowledge that one may wish to relax the perfect information assumption and examine performance on stale, noisy data, possibly by incorporating Bayesian state estimation techniques (e.g. Kalman filtering).

Additionally, we note that ERIS' decisions are not optimal. While it performs better than the comparison methods, there is further room for improvement. Even though intersections share information, they act independently, meaning that intersections make decisions that appear optimal given the known cluster information, but may be globally sub-optimal. Chapters 4 and 5 address this issue by demonstrating that incorporating global constraints with ERIS can lead to performance improvements. Furthermore, ERIS occasionally makes inefficient decisions that increase the delay for some vehicles in the network. Chapters 6 and 7 discuss these inefficiencies and demonstrate how modifications to ERIS' optimization problem can reduce congestion and delay.

Part II

Green Wave Coordination

Chapter 4

Integrating Green Wave Coordination with ERIS

4.1 Overview

Chapter 3 presented ERIS, a real-time, decentralized scheduling algorithm to reduce traffic congestion at intersections. While ERIS offers an advantage over other scheduling methods, there remains further room for improvement. In the second part of this thesis, we consider one such improvement, namely introducing green wave coordination constraints that can be assessed in real-time to reduce overall network delay. This proposed modification is motivated by a challenge presented in Chapter 1:

4. **Global Coordination** - Scheduling more than one intersection at once tends to increase runtime exponentially. (For example, McCluskey and Vallati (2017) proposes a search method that requires roughly 30 seconds to generate a joint schedule for 7 intersections). As a result, many schedule-driven methods are decentralized, meaning the scheduler at each intersection optimizes a local objective. These local objectives do not always align with global objectives, and as a result, schedulers make sub-optimal decisions.

4.1.1 Motivating Example

We begin by considering the following example, presented below in Figure 4.1, that demonstrates that a decentralized approach may select a sub-optimal global plan. To ease the presentation of our example, we make several simplifying assumptions: vehicles may travel through the intersection instantaneously and each green signal has a 5 second minimum time, a 5 second change-over time (yellow and red time), and no start-up lost time.

In a decentralized environment, the left intersection will calculate that it is better to first serve the three pink vehicles than the two blue vehicles. Based on the 5 second green minimum time and 5 second change-over time, each blue vehicle experiences a delay of 10 seconds for a total delay of 20 seconds. The left intersection will inform the right intersection of the soon to arrive pink vehicles. The right intersection repeats a similar calculation and also concludes that it is better to serve the three pink vehicles before the two blue vehicles. Again, this causes 20

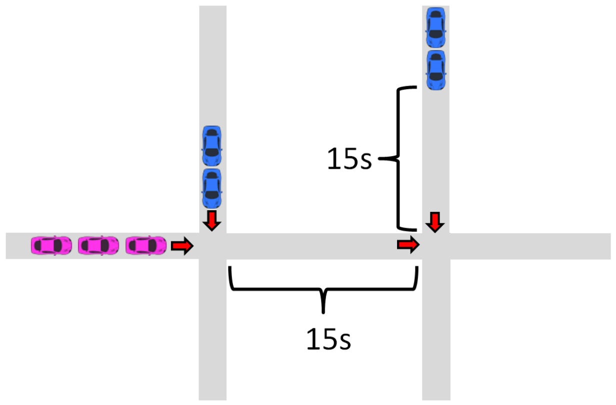


Figure 4.1: Example Network where Decentralized Solvers Fail to Make Globally Optimal Decisions

seconds of delay at this intersection for a total delay of 40 seconds across both intersections.

However, a global optimizer would have observed that the three pink vehicles slow down a total of four blue vehicles. If we were to serve the blue vehicles first, the pink vehicles would each experience 10 seconds of delay at the left intersection. The pink vehicles will not be delayed at the right intersection because the blue vehicles will have already passed through the intersection by the time the pink vehicles arrive. Overall, this is a total network delay of 30 seconds, which is lower than the enacted strategy's delay of 40 seconds.

4.1.2 Chapter Outline

This chapter proposes CARIC (Centralized Agent Reducing Intersection Congestion), an intersection scheduling method that considers both local and global objectives while maintaining real-time responsiveness. While centralized control schemes are generally difficult to embed into real-time traffic signal control decision processes without sacrificing scalability, CARIC provides a limited amount of centralization that can run in real-time. CARIC proposes potential green wave coordination plans for several (or all) intersections in the network, asks selected intersections to assess each plan from its local perspective, selects the plan that provides the lowest global delay, and informs each intersection of the selected plan.

The remainder of this chapter is organized as follows. We define the term "green waves," which is an important feature of our coordination strategy. We then discuss several related scheduling methods that consider coordination. We describe our implementation of CARIC to generate green waves. We discuss modifications to ERIS so that it can be integrated with CARIC. We present several experiments comparing CARIC against ERIS. We then conclude.

4.2 Background

This section first defines the term "green wave." Then, we discuss several other intersection control strategies that consider real-time coordination between multiple intersections.

4.2.1 Green Waves

A green wave is commonly defined as pattern of traffic signals programmed to allow vehicles travelling a specified route to travel across two or more intersections without stopping after crossing the first intersection. (Vehicles may be required to wait at the first intersection in the green wave). When the green wave begins, this first intersection will provide a green signal to vehicles travelling along the specified route. These vehicles pass through the intersection and travel along the specified route to the second intersection in the green wave. This intersection will provide a green signal to these vehicles when they arrive at the intersection. These vehicles then pass through the second intersection. If there is a third intersection in the wave, vehicles travel along the specified route to the third intersection, which also provides a green signal to these vehicles when they arrive at the intersection. If there are additional intersections along the wave, the above pattern continues at these intersections.

Consider the network presented in Figure 4.2 that has three intersections which are roughly 15 seconds apart when travelling eastbound (towards the right).

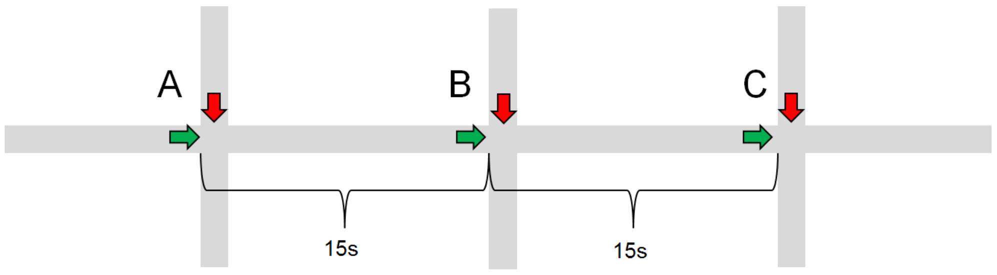


Figure 4.2: Network To Illustrate Green Waves

Let us assume that our intersections have been programmed such that Intersection A provides a green signal at time 30 to the eastbound movement, Intersection B provides a green signal at time 45 to the eastbound movement, and Intersection C provides a green signal at time 60 to the eastbound movement.

A vehicle that arrives at Intersection A along the eastbound movement at time 30 will pass through the intersection upon its arrival. The vehicle requires 15 seconds to travel to Intersection B. Since Intersection B has been programmed to provide a green signal to the eastbound movement at time 45, this vehicle can pass through the intersection when it arrives at time 45. Similarly, since Intersection C has been programmed to provide a green signal to the eastbound movement at time 60, this vehicle can pass through the intersection when it arrives at time 60.

This pattern is an example of a green wave as it allowed the vehicle to travel along a certain route (eastbound) without stopping at Intersection B or Intersection C. (The vehicle also did not stop at Intersection A, but this is not required. Had the vehicle instead arrived at time 15, it would wait for 15 seconds and then starting at time 30, proceed through the green wave as described above).

Our method, CARIC, will generate green waves that allow vehicles travelling along a certain route to travel through a network without stopping.

4.2.2 Related Work

Chapter 2 provided an overview of parametric methods and schedule-driven methods, both of which share similarities with our model, CARIC. We review several of these that are most similar. We then present a detailed description of In|Sync which is a commercially available system which utilizes green waves.

Recall that Surtrac2012 (like ERIS) is a decentralized schedule-driven control strategy [69]. While the schedules that Surtrac2012 calculates consider predicted downstream arrivals from neighboring intersections (in addition to vehicle clusters at the intersection), Surtrac2012's decisions are myopic since they do not consider their impact on overall network performance.

Recent work discusses how to incorporate additional downstream information to improve Surtrac2012 [33]. By modifying the objective function to incorporate a term relating to congestion at downstream neighbors, this method is able to reduce delay by roughly 35% in high demand scenarios. However, decisions are still made by individual intersections and thus intersections make globally sub-optimal decisions. CARIC, with help from ERIS, will instead optimize a global objective.

Parametric control methods were also discussed in Chapter 2. Parametric methods attempt to implement a fixed timing plan that efficiently serves all vehicles across a network. SCOOT, SCATS, and ACS Lite use recent traffic observations to update the cycle, split, and offsets [42, 43, 50]. In some instances, they might allow vehicles to travel across several intersections at once without stopping. However, as mentioned earlier, these methods do not react to real-time traffic conditions. Our method, CARIC, generates schedules that allow vehicles to travel across several intersections at once without stopping while also reacting to real-time traffic conditions.

Another method, known as REALBAND, considers real-time signal coordination to minimize delay [17]. Using estimated cluster arrival times at intersections, REALBAND runs a centralized search to generate a tree where each decision point/branch corresponds to prioritizing one cluster (meaning it will not need to stop at the intersection) over another cluster (meaning this cluster will need to stop). They then select the leaf with the minimum cost. Their experiments demonstrate a 10% reduction in delay compared to a fixed timing plan when running this algorithm across two intersections.

Similar to our method, the RHODES traffic signal control system employs both centralized coordination and local optimization [46]. RHODES utilizes the REALBAND method to optimize cluster prioritization while individual intersections each run a dynamic programming algorithm, Controlled Optimization of Phases [54].

Most relevant to our work is Rhythm Engineering's In|Sync system [12]. This system generates "green tunnels" (which we interpret to be a synonym of "green waves"). In some instances, their green tunnels are applied to two movements (e.g. both eastbound and westbound traffic) simultaneously.

Their video description mentions five key aspects to their methods: green tunnel durations, adaptive periods, dynamic phasing, dynamic sequences, and green time allocation [48]. With In|Sync, the busiest intersection is selected as the "anchor" intersection meaning that important parameters are based on this intersection's congestion - the green tunnel duration (how much green time to provide the busiest movement(s)) and the adaptive period (how much time should occur between successive starts of the green tunnel at this anchor intersection). Given these constraints, other intersections individually calculate their local green tunnel start and end times (based on travel time between them and the anchor intersection).

In|Sync allows traffic signal controllers to jump between phases in any order, which they refer to using the terms "dynamic phasing" and "dynamic sequencing." A phase may be served twice (if there is sufficient time) between successive green tunnels. This specification provides additional flexibility to the range of strategies that they may implement on their controller. We note that the increased flexibility (meaning more efficient schedules may be found) may contribute to the potential for increased driver confusion (meaning accidents may be more likely). In our work, we do not allow for dynamic phasing and sequencing primarily due to the fact that it is less common, yet note that this is an important question beyond the scope of this thesis.

Once the green tunnel start and end times are known, each intersection in the system, individually, performs a green time allocation calculation. At a given intersection, between the green tunnel start and end times, a green signal is provided to the movement(s) corresponding to the green tunnel. Otherwise, the intersection greedily selects the busiest compatible movements to provide a green signal, prioritizing vehicles that have been waiting longer. The controller serves movements until completion and then switches to another phase if time remains before the beginning of the green tunnel movement(s).

CARIC differs from In|Sync in several ways, two of which we highlight. In|Sync solely considers the congestion at the anchor intersection when selecting the green wave pattern to execute. CARIC queries each intersection along the green wave to determine the local impact and combines this information to determine the best green wave pattern. This method prevents CARIC from selecting green waves that greatly increase the delay for vehicles travelling along side streets. Second, the scheduling rules and algorithms at individual intersections are quite different. With In|Sync, when an intersection is not serving the green tunnel movement(s), the intersection uses a greedy scheduling method to select which phase to serve; additionally, the intersection does not require a fixed phase ordering. CARIC, when not serving the green wave movements, runs a schedule-driven method (namely ERIS) to calculate the schedule to enact while obeying a fixed phase ordering. (ERIS optimizes a more restrictive local problem, yet considers more options than In|Sync; we cannot easily compare the quality of local schedules between the two methods).

4.3 CARIC

In this section, we present Centralized Agent Reducing Intersection Congestion (CARIC). CARIC is a centralized agent that calculates green wave coordination plans for a network of intersections that each run ERIS. CARIC makes high level decisions about which green wave coordination plans to consider and ultimately select while relying on the power of ERIS' A* search to efficiently assess potential green wave coordination plans.

Before discussing CARIC in detail, we present several definitions that will be useful when describing CARIC. We then provide an overview of CARIC by discussing the CARIC pipeline. Finally, we present CARIC's pseudocode and explain several components of CARIC in additional detail.

4.3.1 Green Wave Definitions

This subsection presents several definitions that will be useful when explaining CARIC and green waves.

The **green wave pattern** is a sequence of tuples $(i, \vec{m})$ that will be included in our green wave. In each tuple, i denotes the intersection and $\vec{m}$ denotes the movements at that intersection that should receive a green signal during the green wave (these movements will be referred to as the **green wave movements**). For example, for the network presented in Figure 4.2, the green wave pattern is the following sequence:

$$\begin{aligned}&[(\text{Intersection A}, [\text{Eastbound Green}]),\\&\ (\text{Intersection B}, [\text{Eastbound Green}]),\\&\ (\text{Intersection C}, [\text{Eastbound Green}])]\end{aligned}$$

Vehicles that travel through the intersections and movements of the green wave pattern are said to travel along the **green wave route**.

Next, we define the term **local green wave constraint** as the timing requirements that a single intersection must follow to implement a green wave. A local green wave constraint is specified by a tuple $(i, t_s, t_e, \vec{m})$ where i represents the intersection to which this constraint applies, t_s represents the start time of the constraint, t_e represents the end time of the constraint, and $\vec{m}$ represents the required green movements of the intersection during this time range.

For the example, the constraint $(\text{Intersection A}, 0, 8, [\text{Eastbound Green}])$ specifies that Intersection A must provide a green signal in the eastbound direction from time 0 through time 8. (The constraint does not prevent the green signal at Intersection A from beginning before time 0 or ending after time 8).

A **green wave coordination plan** is a sequence of **local green wave constraints**, matching the sequence from the **green wave pattern**. For example, the following sequence is a green wave coordination plan:

$$\begin{aligned}&[(\text{Intersection A}, 0, 8, [\text{Eastbound Green}]),\\&(\text{Intersection B}, 15, 23, [\text{Eastbound Green}]),\\&(\text{Intersection C}, 30, 38, [\text{Eastbound Green}])]\end{aligned}$$

The sequence of intersections and movements matches that of the example green wave pattern. Additionally, each intersection and movement pair is associated with a matching local green wave constraint with a start time and end time.

For the remainder of this thesis, we will use the phrase **green wave coordination plan** to refer to the green waves that CARIC generates while reserving the shorter term **green wave** to refer to the general concept of green waves.

The **start time** of a green wave coordination plan refers to the time that the green wave coordination plan begins at the first intersection of the green wave pattern. For the above example, the start time of our green wave coordination plan is 0, which is the start time of the green movement (t_s) at Intersection A.

When discussing a specific intersection (intersection i), a **corresponding** local green wave constraint refers to the local green wave constraint from the green wave coordination plan that applies to intersection i.

We will use three adjectives to describe green wave coordination plans and local green wave constraints. First, we will use the term **candidate** to describe a green wave coordination plan that CARIC is in the process of evaluating. CARIC will often evaluate multiple candidate green wave coordination plans simultaneously. Similarly, at times, CARIC will inform individual intersections of candidate local green wave constraints that correspond to the candidate green wave coordination plans.

After completing its evaluation of multiple candidate green wave coordination plans, CARIC will choose one of these plans, which we refer to as the **selected** green wave coordination plan. The selected green wave coordination plan is saved and during later time steps will be referred to as an **existing** green wave coordination plan.

It may be the case that there are multiple existing green wave coordination plans. (For example, one green wave coordination plan might start at time 0, another might start at time 60). Intersections must jointly satisfy the corresponding local green wave constraints from all existing green wave coordination plans. A **set of local green wave constraints** contains any number of local green wave constraints that an intersection must jointly satisfy.

4.3.2 CARIC Architecture & Planning Pipeline

To implement CARIC, we require a network where each intersection has a separate processor running ERIS. There is an additional processor, which we refer to as the central agent, located at one of the intersections which runs CARIC. This central agent is responsible for calculating green wave coordination plans, storing the existing green wave coordination plans, and informing each intersection of the corresponding local green wave constraints. Individual intersections run a modified version of ERIS to enact schedules obeying local green wave constraints. When all intersections enact schedules that satisfy their local green wave constraints, the green wave coordination plan allows vehicles to travel along the green wave route without stopping.

We begin by explaining how CARIC calculates green wave coordination plans through its *Planning* activity. A graphic of CARIC's *Planning* pipeline is presented in Figure 4.3.

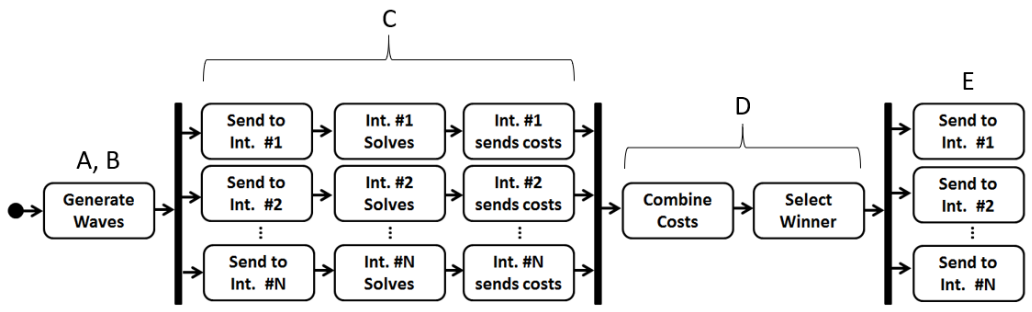


Figure 4.3: CARIC Planning Pipeline

When *Planning* a green wave coordination plan, CARIC begins by generating a variety of candidate green wave coordination plans. The corresponding local green wave constraints for each candidate green wave coordination plan are sent to the appropriate intersection.

At each intersection, for each candidate local green wave constraint it receives, ERIS will calculate a minimal cost schedule that satisfies the local green wave constraint. The costs corresponding to each these schedules are then sent back to CARIC. After receiving all of the costs, for each candidate green wave coordination plan, CARIC sums the costs of all of the component local green wave constraints that were calculated by ERIS. This provides a total cost for each candidate green wave coordination plan. CARIC selects the candidate with the lowest total cost to become an existing green wave coordination plan. CARIC then informs each intersection along the green wave pattern of the corresponding local green wave constraint that they must now enforce.

We now present CARIC's pseudocode to provide additional detail on how CARIC generates candidate green wave coordination plans during *Planning*.

4.3.3 Planning Pseudo-code

The pseudo-code for our implementation is presented in Algorithm 2. The letters A-E in the Planning Pipeline (Figure 4.3) point to the relevant portions of the pseudo-code. Additionally, we will reference the intersection in Figure 4.4 as an example when detailing the algorithm.

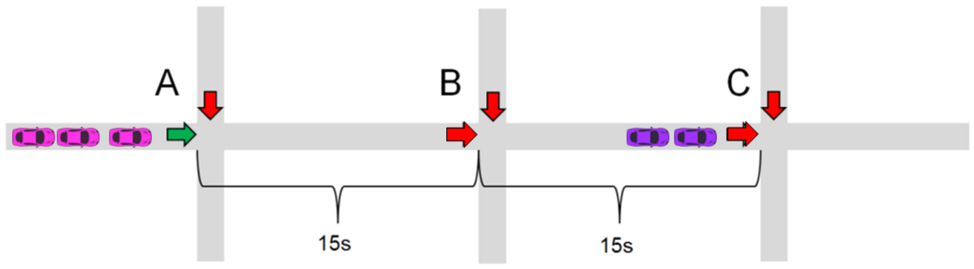


Figure 4.4: Network with Vehicles To Illustrate Green Wave Coordination Plan Generation

Step A) Generate Relative Intersection Start Times

To generate the candidate green wave coordination plans, CARIC must first calculate the relative start time for every green wave movement of the green wave pattern. These relative start times are relative to the start time at the first intersection of the green wave pattern, so we set the relative start time at the first intersection to 0 (line 7). To calculate the relative start time for other intersections, CARIC requires the average travel time between each pair of adjacent intersections in the green wave pattern as well as the estimated time to clear the existing queued vehicles that is based on the provided vehicle clusters. The relative start time for these intersections is the sum of the previous intersection's relative start time and the expected travel time between the two intersections, minus the time to clear the queue along the green wave movement at this intersection (lines 11 - 13). (By subtracting the time required to clear the existing queue from the expected travel time, vehicles travelling along the green wave route do not need to wait for existing queues to clear).

Let us consider the network presented in Figure 4.4 and use the same example green wave pattern as earlier (repeated below):

Algorithm 2 *Planning* a Green Wave

```
procedure PLAN(GreenWavePattern, VehicleClusters, ExistingGreenWaveCoordinationPlans)
    // A) Generate Relative Intersection Start Times
    Initialize Empty DictionaryOfRelativeStartTimes
    Initialize Empty DictionaryOfTimeToClearQueues
    for each Intersection in GreenWavePattern do
        if Intersection is First in GreenWavePattern then
            IntersectionStartTime = 0
            Calculate TimeToClear Queue at Intersection
            Duration = TimeToClear
        else
            Calculate ExpectedTravelTime between Intersections
            Calculate TimeToClear Queue at Intersection
            IntersectionStartTime += ExpectedTravelTime - TimeToClear
        Save IntersectionStartTime to DictionaryOfRelativeStartTimes
        Save TimeToClear to DictionaryOfTimeToClearQueues

    // B) Generate Candidate Green Wave Coordination Plans
    Initialize empty ListOfCandidateGreenWaveCoordinationPlans
    StartTimeRange = [duration_to_next_start_min, duration_to_next_start_min + step, ... duration_to_next_start_max]
    for GreenWaveStartTime ∈ StartTimeRange do
        Initialize empty ListOfConstraints for a Single Green Wave Coordination Plan
        for each (Intersection,GreenWaveMovements) in GreenWavePattern do
            IntersectionStartTime = GreenWaveStartTime + DictionaryOfRelativeStartTimes[Intersection]
            if Intersection Not First in GreenWavePattern then
                Duration += DictionaryOfTimeToClearQueues[Intersection]
            IntersectionEndTime = IntersectionStartTime + Duration
            SingleConstraint = (Intersection, IntersectionStartTime, IntersectionEndTime, GreenWaveMovements)
            Save SingleConstraint to ListOfConstraints
        Save ListOfConstraints into ListOfCandidateGreenWaveCoordinationPlans

    // C) Send Local Green Wave Constraints to Intersections to Evaluate
    for each Intersection in GreenWavePattern do
        if Intersection is First in GreenWavePattern then
            Send Flag to Modify Calculation
        else
            Alert Intersection of VehicleClusters to Drop (if any)
        Send Corresponding Local Green Wave Constraint(s) for ExistingGreenWaveCoordinationPlans
        for each CandidateGreenWaveCoordinationPlan in ListOfCandidateGreenWaveCoordinationPlans do
            Send Corresponding Local Green Wave Constraint
    // Wait for Intersections to Calculate Delays
    Receive Delays from each Intersection

    // D) Score each Candidate Green Wave Coordination Plan and Select Best
    Initialize Empty IncumbentBestGreenWaveCoordinationPlan with Infinite Delay
    for Each CandidateGreenWaveCoordinationPlan do
        Calculate TotalDelay of CandidateGreenWaveCoordinationPlan by Adding Delays at all Intersections
        if TotalDelay is less than IncumbentBestGreenWaveCoordinationPlan's Delay then
            Update IncumbentBestGreenWaveCoordinationPlan
    SelectedGreenWaveCoordinationPlan = IncumbentBestGreenWaveCoordinationPlan

    //E) Share Selected Green Wave Coordination Plan with Intersections
    for each Intersection in GreenWavePattern do
        Send Corresponding Local Green Wave Constraint for SelectedGreenWaveCoordinationPlan
```

[(Intersection A, [Eastbound Green]),
(Intersection B, [Eastbound Green]),
(Intersection C, [Eastbound Green])]

The relative start time for Intersection A, the first intersection in the green wave pattern, is set to 0. There is a 15 second travel time between Intersection A and Intersection B (for simplicity, we will assume that this includes possible startup lost time at Intersection A). Additionally, there are no vehicles queued along the eastbound straight movement at Intersection B. Thus, the relative start time for Intersection B is 15 seconds. There is a 15 second travel time between Intersection B and Intersection C. Additionally, there are two vehicles queued along the eastbound movement at Intersection B. Assuming it requires a total of 5 seconds to clear these vehicles (including startup lost time), the relative start time at Intersection C will be 25 (15 + 15 - 5 = 25) seconds.

Additionally, CARIC must calculate the minimum green phase duration to serve the vehicles (travelling along the green wave route) at each intersection along the green wave pattern. At the first intersection, the minimum green phase duration is equal to the amount of time required to serve all vehicles along the green wave movement (lines 8 - 9).

Returning to our example presented in Figure 4.4, if we assume that it will require 8 seconds to serve the three pink vehicles at Intersection A, CARIC will set the initial green phase duration to be 8 seconds long.

Step B) Generate Candidate Waves

Once the relative start times have been calculated, CARIC uses this information to generate a list of candidate green wave coordination plans.

CARIC generates a range of possible start times, one for each candidate green wave coordination plan (line 19). This is parameterized as an inclusive range between two parameters, *duration_to_next_start_min* and *duration_to_next_start_max*. Additionally, CARIC does not consider every possible start time in this range, but rather start times that are *step* (another parameter) seconds apart. For example, if *duration_to_next_start_min* = 20, *duration_to_next_start_max* = 50, and *step* = 10, CARIC will generate one candidate green wave coordination plan beginning at each of the following times: 20, 30, 40, and 50.

This parameterization allows CARIC to explore a wide range of possible start times when *Planning* the green wave coordination plan. As the start time of the wave approaches, this parameterization also allows CARIC to zoom in and explore minor modifications to the previously calculated green wave coordination plan's start time.

CARIC loops through these start times and calculates a candidate green wave coordination plan for each start time (lines 20 - 29). For each start time, CARIC initializes an empty sequence (ordered list) that will be populated with the corresponding local green wave constraints for each intersection to ultimately become a candidate green wave coordination plan (line 21). For each intersection in the green wave pattern, CARIC calculates one such local green wave constraint (line 22). t_s is the sum of green wave start time and the relative offset calculated in step A (line 23). (For the first intersection, the relative offset is 0, and thus the green wave start time equals t_s). At the first intersection, t_e is set as the sum of t_s and previously calculated green phase duration (line 26). However, for intersections after the first intersection, the green phase duration for green wave movements must be increased by the time required to clear the queue (calculated in part A) so that the local green wave constraint provides sufficient time to both serve the vehicles travelling along the green wave route as well as the vehicles queued at the intersection (lines 24 - 25). This information, along with the intersection id (i) and relevant movement(s) ($\vec{m}$) are saved as a constraint tuple (line 27). This tuple is then appended to the ListOfConstraints for a single candidate green wave coordination plan (line 28).

We now present an example of the calculation when WaveStartTime = 20 for the network presented in Figure 4.4. The tuple for the first intersection in the green wave pattern, Intersection A, is (A, 20, 28, [Eastbound Green]). $t_s = 20$ because the offset is 0 for the first intersection and the WaveStartTime $= 20$. $t_e = 28$ because the green phase duration (d) is set to 8 and $t_e = t_s + d = 20 + 8$.

The tuple for the second intersection in the green wave pattern, Intersection B, is (B, 35, 43, [Eastbound Green]). $t_s = 35$ because the relative start time is 15 for the second intersection and the WaveStartTime $= 20$. Since there are no vehicles queued to increase the green phase duration, it remains at 8. It follows that $t_e = 43$ because $t_e = t_s + d = 35 + 8$.

The tuple for the third and final intersection in the green wave pattern, Intersection C, is (C, 45, 58, [Eastbound Green]). $t_s = 45$ because the offset is 25 for the third intersection and the WaveStartTime $= 20$. The required green phase duration will be increased at Intersection C to 13 because there is a queue of two vehicles (requiring 5 seconds to serve) at the intersection. It follows that $t_e = 58$ because $t_e = t_s + d = 45 + 13$.

Each of these local green wave constraints is added to a list and the full green wave coordination plan is:

$$\begin{aligned}&[(A, 20, 28, [\text{Eastbound Green}]),\\ &(B, 35, 43, [\text{Eastbound Green}]),\\ &(C, 45, 58, [\text{Eastbound Green}])]\end{aligned}$$

Step C) Send Waves to Intersections to Solve

Once the candidate green wave coordination plans have been generated, CARIC loops through each intersection in the green wave pattern and shares the corresponding local green wave constraints as well as other information (Step C).

Other than at the first intersection along the green wave pattern, the green wave coordination plan makes it unlikely that the vehicles travelling along the green wave route will be delayed. CARIC instructs the intersections along green wave pattern (except for the first intersection) to not consider downstream arrivals that originated from the first intersection so that they do not influence the delay calculations (lines 35 - 36).

Because CARIC does not calculate the delay for these downstream arrivals at intersections after the first intersection, it must ensure that it accurately calculates the delay for these vehicles at the first intersection along the green wave pattern. If the first intersection's schedule provides a green signal along the green wave pattern that starts at the green wave coordination plan's start time, the delay does not need to be modified. However, if the intersection's schedule provides a green signal prior to the green wave coordination plan's start time, the green wave coordination plan's start time will override the schedule's start time. Additional delay for vehicles at the first intersection of the green wave pattern must then be increased to account for this difference. CARIC sends a flag to the first intersection indicating that ERIS must modify its delay calculation accordingly (lines 33 - 34).

CARIC also reminds each intersection of the existing green wave coordination plans that were previously calculated by sending the corresponding local green wave constraints (line 37). (For example, if CARIC selects a new green wave coordination plan approximately every 60 seconds, it is possible that the previous green wave coordination plan has not completed by the time this new green wave coordination plan is being considered). After informing each intersection of the existing green wave coordination plans (if any), CARIC sends the corresponding local green wave constraint for each candidate green wave coordination plan (lines 38 - 39).

Returning to our earlier example, CARIC will send the following four candidate local green wave constraints to Intersection A, one for each candidate green wave coordination plan: (A, 20, 28, [Eastbound Green]), (A, 30, 38, [Eastbound Green]), (A, 40, 48, [Eastbound Green]), (A, 50, 58, [Eastbound Green]).

After receiving the candidate local green wave constraints, a modified ERIS solver running at each intersection calculates one locally optimal schedule for each candidate local green wave constraint (line 40). Intersections perform these calculations in parallel.

After an intersection calculates a schedule for each candidate local green wave constraint, it sends the corresponding delays back to CARIC (line 41). For example, at the first intersection, ERIS might send the following four delays back to CARIC: 200, 300, 400, 500. The first delay corresponds to the first candidate local green wave constraint and so on, thus, CARIC knows how to compile this information. CARIC hangs on this step until it has received information from all intersections along the green wave pattern.

Step D) Score each Wave Candidate and Select Best

Once CARIC has received the delay information from each intersection, it sums this information to calculate the total delay for each candidate green wave coordination plan (lines 45 - 46). It compares these delays to determine the candidate with the lowest total delay which then becomes the selected green wave coordination plan (lines 47 - 49).

Step E) Share Best Wave with Intersections

After selecting the best candidate, CARIC shares the cooresponding local green wave constraint with each intersection along the GreenWavePattern (lines 52 - 53).

4.3.4 CARIC Activities

We have detailed how CARIC initially calculates green wave coordination plans, which we refer to as *Planning*. CARIC also *Re-Plans* and *Re-Calibrates* these green wave coordination plans. In this section, we describe *Re-Planning* and *Re-Calibration* and detail when each activity occurs. Finally, we discuss how CARIC handles situations when no feasible green wave coordination plans are calculated.

Re-Planning

After *Planning* a green wave coordination plan, vehicles will continue to arrive throughout the network and the previously selected green wave coordination plan may no longer be the ideal plan. CARIC will engage in *Re-Planning* to considers slight modifications to the green wave coordination plan's start time. CARIC examines whether shifting the green wave coordination plan's start time by up to 5 seconds (earlier or later) of the incumbent start time can lead to a reduction in delay.

We do not want CARIC to *Re-Plan* a green wave coordination plan but not have time to share the results with all intersections in the network. For this reason, once a green wave coordination plan is within 3 seconds of beginning, the start time is locked and the green wave coordination plan is not *Re-Planned* again.

We only need to modify one line of our pseudo-code to allow for *Re-Planning*. When generating the StartTimeRange (line 19), we instead initialize this range to explore all integral start times within 5 seconds of the incumbent start time. Formally, we specify $\textit{duration\_to\_next\_start\_min} = (\text{IncumbantStartTime} - 5)$, $\textit{duration\_to\_next\_start\_max} = (\text{IncumbantStartTime} + 5)$, and $\textit{step} = 1$. The remainder of the algorithm is unchanged.

Re-Calibration

Even though *Re-Planning* runs in real-time, we may wish to avoid running it every time step as real-world communication may become overwhelmed. When not running *Planning* or *Re-Planning*, CARIC will still make minor updates to the green wave coordination plan by running *Re-Calibration*.

During *Re-Calibration*, CARIC assesses whether additional vehicles have arrived or departed along the green wave movements. CARIC keeps the start time of the green wave coordination plan unchanged but updates the relative start times at intersections (excluding the first) as these queues change. CARIC also updates the duration of individual local green wave constraints according to the new queue lengths.

Re-Calibration executes Step A of the above pseudo-code and then executes Step B with one modification: StartTimeRange is a list with a single element, the previously calculated green wave coordination plan's start time. This proposed green wave coordination plan is then evaluated to see if each intersection can feasibly implement the corresponding local green wave constraints. If every intersection can satisfy these constraints, the re-calibrated green wave coordination plan replaces the existing green wave coordination plan. Otherwise, this proposed green wave coordination plan is dropped.

Activity Frequency

CARIC runs *Planning* when the traffic controller at the first intersection along the green wave pattern terminates the green signal for the vehicles travelling along the green wave route. Thus, after planning a green wave coordination plan, the green wave coordination plan does not immediately begin. As mentioned above, traffic conditions will change and the selected green wave coordination plan may no longer be the ideal candidate. Every *freq* seconds, CARIC will engage in *Re-Planning* to determine if a better green wave coordination plan exists. *Re-Planning* continues until the green wave coordination plan's start time is within 3 seconds. At this point, the green wave coordination plan's start time is fixed.

freq will often be set above 1 second. When this is the case, CARIC will engage in *Re-Calibration* every time step that neither *Planning* nor *Re-Planning* are executed.

Activity Failure

If many vehicles arrive between two successive *Re-Planning* steps, it may be the case that the examined candidate green wave coordination plans' start times are not feasible. Specifically, if many vehicles join the queue at an interior intersection of the green wave pattern, CARIC will desire to push the start time of this movement earlier. However, due to limitations from the fixed phase ordering and phase minimum times, such a move may not be feasible. Similarly, since CARIC limits how much it is willing to push back the start time of the green wave coordination plan during *Re-Planning* (e.g. to 5 seconds), none of the examined candidate green wave coordination plans may be feasible. When this is the case, CARIC maintains the previously calculated green wave coordination plan and does not make modifications. Vehicles travelling along the green wave route will still receive a benefit but may need to slow down at this intersection.

An analogous issue may arise during *Re-Calibration*. At an intersection with queue buildup, CARIC may wish to push the local green wave constraint's start time earlier, but due to limitations from the fixed phase ordering and phase minimum times, such a move may not be feasible. As above, when this is the case, CARIC does not modify the existing green wave coordination plan.

4.4 Modifications to ERIS

The previous section formalized CARIC and explained how CARIC calculates green wave coordination plans. In this section, we discuss how ERIS assesses and enacts the local green wave constraints.

During most time steps, when CARIC is neither *Planning* nor *Re-Planning*, each intersection runs a modified version of ERIS that returns the best schedule that satisfies the local green wave constraints of the existing green wave coordination plan(s). When CARIC is either *Planning* or *Re-Planning*, each intersection assists CARIC by assessing the corresponding local green wave constraints of each candidate green wave coordination plan.

We make two modifications to ERIS and enable a parameter setting so that ERIS can assist CARIC to enact and to calculate the delay for local green wave constraints. In this section, we discuss these necessary components. First, we detail how ERIS calculates schedules that satisfy the local green wave constraints from CARIC. Next, we recall aggressive cluster splitting from Chapter 3 and explain why it is necessary for satisfying local green wave constraints. Finally, we discuss additional logic at the first intersection of the green wave pattern to hold a red phase beyond the phase minimum so that the green wave coordination plan does not begin early.

4.4.1 Satisfying Local Green Wave Constraints

We modify ERIS to calculate schedules that satisfy the local green wave constraints sent by CARIC. First, we discuss modifications to ERIS so that it can calculate a single schedule that satisfies a single set of local green wave constraints. We then discuss how ERIS simultaneously calculates multiple schedules (each schedule satisfying one unique set of local green wave constraints) during *Planning* and *Re-Planning*.

ERIS Changes for Calculating a Schedule Satisfying a Single Set of Local Green Wave Constraints

During most time steps (when CARIC neither runs *Planning* nor *Re-Planning*), CARIC dictates that each intersection in the wave pattern must follow the existing green wave coordination plan(s). CARIC reminds each intersection of the corresponding local green wave constraint(s). Each intersection must calculate a schedule that jointly satisfies all corresponding local green wave constraint(s). (Recall from Section 4.3.1 that we refer to this as a **set of local green wave constraints**).

For every search state it examines, ERIS considers whether the search state's underlying partial schedule can satisfy every local green wave constraint in the set of local green wave constraints. Given the underlying partial schedule, ERIS assesses whether each local green wave constraint is *Satisfied*, *Feasible*, or *Infeasible*. A local green wave constraint will be marked as *Satisfied* if the underlying partial schedule serves the appropriate phase for the full duration of the local green wave constraint.

When a local green wave constraint cannot be marked *Satisfied*, ERIS will consider whether to mark the local green wave constraint as *Feasible* or *Infeasible*. If a local green wave constraint is marked as *Feasible*, then the search state has at least one descendant with an underlying partial schedule that can satisfy the local green wave constraint. To determine this, ERIS considers the search state's underlying partial schedule and calculates all future possible start times for the phase dictated by the local green wave constraint. For each possible start time, ERIS also calculates the phase end time, based on the phase maximum time. If ERIS finds a start time and end time pair that include the local green wave constraint's times, ERIS marks the local green wave constraint as *Feasible*.

If a local green wave constraint cannot be marked as *Satisfied* or *Feasible*, it will be marked as *Infeasible*, meaning either this search state's underlying partial schedule violates the local green wave constraint or that there exist no search state descendants with an underlying partial schedule that can satisfy the local green wave constraint.

After assessing each local green wave constraint individually, ERIS combines the results to assess the feasibility of the set of local green wave constraints. Trivially, if there are no constraints in this set, the set is marked as *Satisfied*. Otherwise, we combine the evaluation as follows: If all local green wave constraints are determined to be *Satisfied*, the set is marked as *Satisfied*. If any local green wave constraints is marked as *Infeasible*, the set is marked as *Infeasible*. Otherwise the set is marked as *Feasible*.

Based on the results of this calculation, ERIS search states are updated to include an additional categorical variable, $s : s \in \{\textit{Satisfied}, \textit{Feasible}, \textit{Infeasible}\}$, to indicate if the set of local green wave constraints is satisfiable by the underlying partial schedule. Given this additional feature, ERIS returns the first goal state (recall that a goal state has served all clusters) that is marked as either *Satisfied* or *Feasible*. During its search, ERIS only expands search states that are marked as *Satisfied* or *Feasible* since search states marked as *Infeasible* cannot lead to a goal state that satisfies every local green wave constraint in the set of local green wave constraints.

ERIS Changes for Calculating Multiple Schedules each Satisfying a Coordination Plan

When *Planning* or *Re-Planning* a green wave coordination plan, CARIC generates multiple candidate green wave coordination plans and requests that intersections in the green wave pattern assess the local delay from each candidate green wave coordination plan. CARIC shares the corresponding candidate local green wave constraints with each intersection in the green wave pattern.

Note that in addition to the candidate local green wave constraints, there may already be one or more existing local green wave constraints (from existing green wave coordination plans that were calculated during earlier time steps). When an intersection calculates a schedule that satisfies a single candidate local green wave constraint, it must also ensure to satisfy any already existing local green wave constraints. Combining a candidate local green wave constraint with any number of existing local green wave constraints provides us a candidate set of local green wave constraints.

Each intersection runs a single A* search to calculate one schedule that satisfies each set of candidate local green wave constraints. (For example, if CARIC sends one existing local green wave constraint and five candidate local green wave constraints to an intersection, ERIS generates five distinct candidate sets of local green wave constraints, each consisting of the existing local green wave constraint and one of the candidate local green wave constraints; ERIS then calculates five schedules, one satisfying each candidate set of local green wave constraints). ERIS' A* search allows it to share work between the calculation of each schedule and thus, runs in real-time.

ERIS search states keep track of the status for each of these candidate sets of local green wave constraints separately; formally, s now becomes, $\vec{s}$. ERIS must continue searching until it reaches the goal state (or determines that no valid schedule exists) for each candidate set. A search state is expanded only if there is at least one candidate set that is marked as *Satisfied* or *Feasible* for which the search has not yet found a corresponding schedule (goal state).

4.4.2 Aggressive Cluster Splitting

To enable the execution of green wave coordination plans, we require that ERIS run with aggressive cluster splitting (examined in detail in Section 3.7.4). Aggressive cluster splitting allows ERIS to consider a larger set of schedules and ultimately find a reasonable schedule that satisfies a set of local green wave constraints. Recall that aggressive cluster splitting allows ERIS to break clusters into 3 second-sized pieces and serve any number of these pieces.

The importance of aggressive cluster splitting is best demonstrated through example. Suppose the following: The side street is currently receiving a green signal. There is a cluster along this side street that requires 30 seconds to serve. There is a 5 second change-over time between phases. Finally, there is a local green wave constraint that requires that the main street receives a green signal at time 20.

If ERIS is not allowed to divide clusters (aggressive cluster splitting disabled), it will consider two options. First, it could end the side street phase immediately, switch to the main street phase at time 5, and be 15 seconds early for the local green wave constraint, thereby unnecessarily delaying the side street vehicles. Alternatively, if ERIS schedules the full cluster, it will not switch to the main street until time 35, which violates the local green wave constraint.

On the other hand, if aggressive cluster splitting is enabled, ERIS is able to serve the side street cluster until time 15, switch to 5 seconds of change-over time, and then switch to the main street green phase at time 20 to satisfy the local green wave constraint.

When ERIS attempts to satisfy local green wave constraints but does not have aggressive cluster splitting enabled, we observe that many local green wave constraints are very costly to satisfy. This problem is eliminated when aggressive cluster splitting is enabled.

4.4.3 Delaying Green Signals at the First Intersection

The local green wave constraints that CARIC sends to ERIS are requirements on when the signal should be green along a particular movement. The local green wave constraints do not preclude ERIS from starting a green phase earlier (or ending a green phase later). Starting a green phase too early is problematic for the first intersection in the green wave pattern. If this intersection provides a green signal before the green wave coordination plan's start time, vehicles will depart this intersection and will reach the next intersection earlier than the local green wave constraint requires this intersection to provide a green signal. To prevent this from occurring, we add additional functionality to the first intersection in the green wave pattern that allows it to delay providing a green signal.

Our logic is implemented as follows at the first intersection in the green wave pattern: If the corresponding movement is currently red, the controller wants to provide a green signal to this movement, and the green wave coordination plan's start time is not yet reached, we override ERIS' output and maintains a red signal along this movement. (The opposite movement is allowed to provide a green signal at this time). This functionality prevents a vehicle traveling along the green wave route from reaching a downstream intersection before the intersection is able to provide a green signal.

ERIS must also modify its delay calculation at the first intersection of the green wave pattern. At this intersection, after calculating a schedule, ERIS considers whether the schedule plans to send vehicles travelling along the green wave route through the intersection prior to the green wave coordination plan's start time. If so, ERIS increases the delay for each of these vehicles by the difference between their planned start time and the time that they would actually depart if the movement were not started until the green wave coordination plan's start time.

4.5 Experimental Analysis on the Portland-Franklin Network

In this section, we examine running CARIC along the Portland-Franklin network. We begin by providing a brief overview of the network. We then present a parameter sweep where we examine different values for the *Re-Planning* frequency. The remainder of our evaluation will use this learned parameter.

We then present a validation experiment where we compare CARIC to ERIS using actual demand; we observe a statistically significant reduction in delay. We then modify the demand to examine the impacts of CARIC when demand is reduced.

4.5.1 Network Overview

The Portland-Franklin traffic network is in Portland, Maine. Along the northwest end of the network are two exits from Interstate 295. The network travels southeast along Franklin Street, along the northern edge of downtown Portland, and ends just before reaching the waterfront area. This model was provided by Rapid Flow Technologies who are in the midst of deploying an updated version of Surtrac along the network. (Rapid Flow Technologies has deployed other networks in Portland, Maine which is why this network is not simply named the Portland network).

This is a corridor network with 5 intersections along the corridor (Franklin Street). The northwestern-most intersection of the network, Franklin Street and Marginal Way, is the busiest intersection in the network as this intersection is adjacent to two interstate exits. To simplify our modelling, we combine the two interstate exits into one shared movement and combine the demand.

A graphic of our modified network is presented below in Figure 4.5. While the network runs from the northwest to southeast in the real-world, in our figures and further discussion, we rotate the intersection 45° counter-clockwise and model Franklin Street as running from the west (highway exit) to the east (waterfront area).

We have based our model of this network on the provided model from Rapid Flow Technologies. Similarly, we obtained estimated demand information from Rapid Flow Technologies for the AM rush hour. These estimates were provided by an outside party, and as such, we did not modify the demand to correct for imperfect detection as we did with the St. Albert network studied earlier in this thesis.

4.5.2 Experimental Setup

We examine running CARIC during the AM rush hour. The dominant flow during the AM rush hour is eastbound, beginning at the freeway exits and travelling into the city. According to the provided demand file, during the AM rush hour, approximately 40.7% of vehicles enter the network from these interstate exits and a total of 11.7% of all vehicles both enter the network from the interstate exits and also travel straight through all five intersections. CARIC will generate green wave coordination plans originating at the highway exits and travelling straight through all five intersections. Note that while green wave coordination plans primarily benefit the 11.7% of vehicles travelling straight through all five intersections, other vehicles travelling eastbound in the network gain lesser benefits. A graphic of the Portland-Franklin network indicating the green wave route is presented in Figure 4.6.

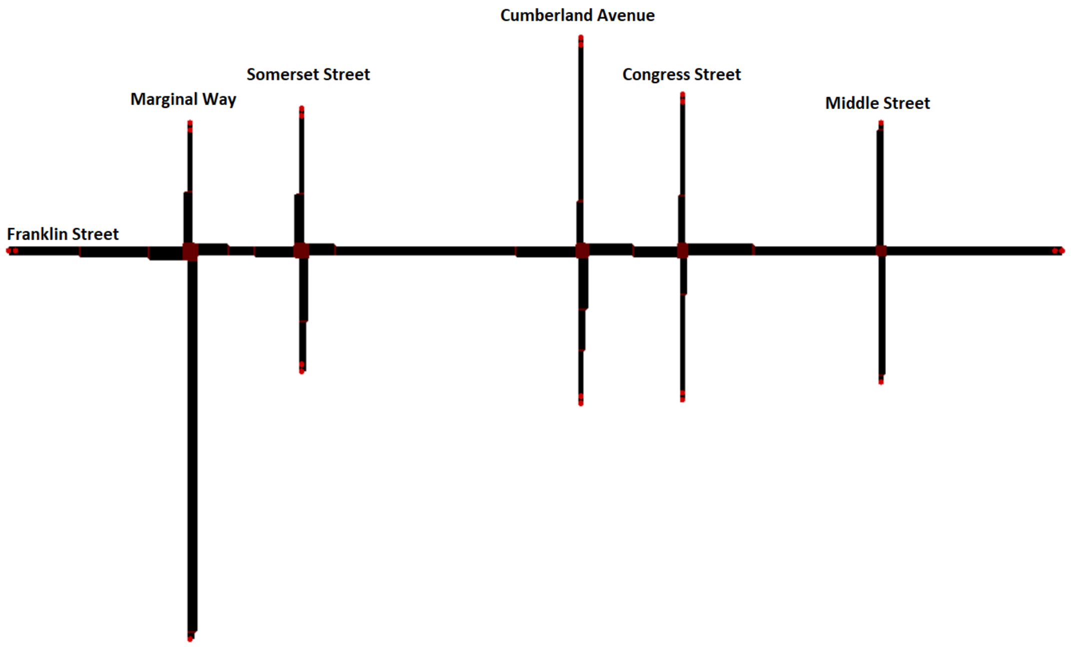


Figure 4.5: Portland-Franklin Network

4.5.3 Examining the Re-Planning Frequency

First, we present a parameter sweep to determine the ideal *Re-Planning* frequency for CARIC. Recall that after *Planning* a green wave coordination plan, traffic conditions will change and the selected start time for the green wave coordination plan may no longer be ideal. CARIC will *Re-Plan* the green wave coordination plan's start time every *freq* seconds. In this evaluation, we examine the ideal value for *freq*.

Specifically, we consider *Re-Planning* the green wave coordination plan every 1, 2, 4, 8, 16, or 32 seconds. We also consider never *Re-Planning* the green wave coordination plan. Additionally, we run ERIS to serve as our baseline. In Table 4.1, we present average delay and average simulation runtime for each strategy. Average simulation runtime is an imperfect yet still useful metric as it also includes set-up time and other unrelated overhead. These runs are conducted on AWS EC2 t2.medium Amazon Linux 2 instances.

This experiment is conducted at the provided level of demand. 50 trials are conducted for each *Re-Planning* frequency.

We first note that all of the wave strategies improve over ERIS. The improvement is largest when *Re-Planning* every 4 or 8 seconds. When *Re-Planning* less frequently than every 8 seconds, we notice a gradual increase in delay. Similarly, when *Re-Planning* more frequently than every 4 seconds, we also notice a gradual increase in delay.

We perform individual two-sided paired t-tests with a 95% confidence level. *Re-Planning* every 8 seconds is statistically significantly better than *Re-Planning* every 1 second, Never *Re-Planning*, and No Waves. *Re-Planning* every 4 seconds is statistically significantly better than *Re-Planning* every 1 second, *Re-Planning* every 32 seconds, Never *Re-Planning*, and No Waves.

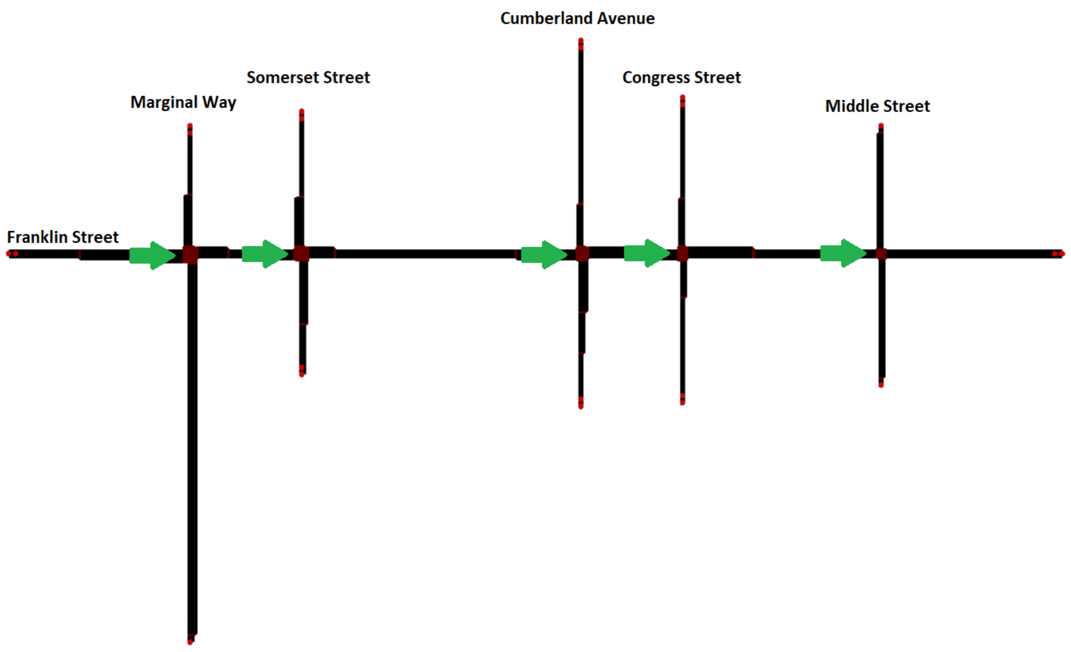


Figure 4.6: Portland-Franklin Network with Green Wave Route Indicated

It is surprising that *Re-Planning* every second is not the best strategy. We propose the following explanation: More frequent *Re-Planning* causes the green wave coordination plan's start time to jitter. We observe the green wave coordination plan's start time increase by a few seconds during one time step then revert back down by the same amount during the next time step.

Intersections must adjust their schedules every time step to accommodate the new local green wave constraints. Intersections may have made decisions that were optimal under the old set of constraints during the previous time step but that are no longer ideal under the new constraints, leading to inefficiency at the intersection. Frequently adjusting the green wave coordination plan's start time causes these inefficiencies to accumulate.

We suspect that when *Re-Planning* more frequently than every 4 seconds, the costs of contin-

	Delay (s)		Runtime (Hours:Minutes)	
	Average	% Improvement	Average	% Increase
No Waves (Baseline)	129.2	0.0%	1:49	0.0%
Waves, Never Replan	126.2	2.3%	1:54	4.4%
Waves, Replan Every 32 Seconds	124.7	3.5%	1:52	3.0%
Waves, Replan Every 16 Seconds	123.6	4.3%	1:54	4.4%
Waves, Replan Every 8 Seconds	122.0	5.5%	1:56	6.5%
Waves, Replan Every 4 Seconds	122.3	5.3%	2:01	10.6%
Waves, Replan Every 2 Seconds	124.9	3.3%	2:12	21.3%
Waves, Replan Every 1 Second	125.5	2.9%	2:34	41.2%

Table 4.1: Evaluation for Different Re-Planning Frequencies

uously modifying the green wave coordination plan's start time outweighs the potential benefits of selecting a slightly better start time. Conversely, when *Re-Planning* less frequently than every 8 seconds, the benefits of a better start time outweigh the potential costs. These intermediate frequencies of *Re-Planning* every 4 or 8 seconds are ideal. For the remainder of our evaluation, we will use a *Re-Planning* frequency of 8 seconds.

We now discuss the runtimes from the experiment. *Re-Planning* more frequently causes the average runtime to increase. *Re-Planning* every second causes an average increase in simulation runtime from 1:49 to 2:34 or a total of 45 minutes. If we assume that this additional 45 minutes is divided evenly across the 5 intersections over the 3600 second simulation, then an average schedule calculation increases by approximately 0.15 seconds ($45 * 60/3600/5 \approx 0.15$). This is not a negligible increase in runtime, but it is low enough so that ERIS can continue to run in under 1 second.

4.5.4 Validation Experiment

We run 20 additional trials to validate that CARIC (*Re-Planning* every *freq* $= 8$ seconds) reduces average vehicle delay as well as average fuel usage and the average number of times a vehicle stops when compared against ERIS. These values, along with their standard errors, are presented in Table 4.2. Fuel usage is calculated using SUMO's built-in emissions model, HBEFA v2.1 [38]. This model updates a vehicle's estimated cumulative fuel usage based on its acceleration and velocity every time step. Recall that we have argued that green wave coordination plans will allow vehicles to travel along the green wave route without stopping; measuring the number of stops will allow us to confirm this.

	Average Delay (s)	Average Fuel Usage (mL)	Average Number of Stops
Baseline ERIS	131.9 ± 3.6	173.3 ± 2.3	2.96 ± 0.08
CARIC	122.3 ± 1.6	166.8 ± 1.1	2.65 ± 0.04
% Improvement	7.2%	3.8%	10.4%

Table 4.2: Evaluation at Provided Demand for Portland-Franklin Network

CARIC reduces average delay by approximately 7.2%, average fuel usage by approximately 3.8%, and the average number of times that a vehicles stops by 10.4% All three of these reductions are statistically significant when performing two-sided paired t-tests with a 95% confidence level.

When observing a simulation of ERIS, we note that queues often form along the eastbound movement of the intersection at Franklin Street and Cumberland Avenue (referenced as the upstream intersection for this discussion) and also along the eastbound movement of the intersection at Franklin Street and Congress Street (referenced as the downstream intersection for this discussion). Even though ERIS communicates downstream arrivals, the controller at the downstream intersection may not be able to immediately accommodate vehicles from the upstream intersection. For example, if the downstream intersection must serve an additional left turn phase, it will not begin the eastbound straight phase until several seconds after vehicles from the upstream intersection have arrived. This delays these vehicles and can also cause backup such that vehicles cannot proceed eastbound through the upstream intersection.

Green wave coordination plans discourage the eastbound movement at the upstream intersection from beginning too early. Several seconds after the upstream intersection provides a green signal to eastbound vehicles, the downstream intersection does the same. (Recall that the baseline offset is based on the travel time between the two intersections. This value is reduced by the estimated time to clear the queue at the downstream intersection). With green wave coordination plans, vehicles are typically able to proceed through the downstream intersection without stopping and thus it is less likely that a queue will form along this movement backing up traffic at the upstream intersection. In addition to reducing the average number of stops, CARIC also decreases the average vehicle delay and reduces average fuel usage.

Our result demonstrates that the benefits of CARIC apply to realistic networks with realistic demands. For the remainder of this section, we will modify the network demand to determine whether CARIC is beneficial under other levels of congestion.

4.5.5 Evaluation using Scaled Down Demands

Next, we evaluate whether CARIC remains beneficial under reduced demand. Our experimental setup is the same as above, except all demand is reduced using a demand scaling factor between 0.6 and 0.95. We present average delay in Table 4.3, average fuel usage in Table 4.4, and the average number of stops in Table 4.5. The results from the above evaluation are included in these tables for easier comparisons.

Demand Scaling Factor	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95	1.0
Baseline ERIS	54.6	55.9	58.1	63.4	66.6	73.2	79.9	104.6	131.9
CARIC	55.5	57.5	59.1	62.3	67.3	73.8	79.9	101.2	122.3
% Improvement	-1.7%	-2.8%	-1.6%	1.8%	-1.0%	-0.8%	0.0%	3.2%	7.2%

Table 4.3: Average Delay (s) at Scaled Down Demands for Portland-Franklin Network

Demand Scaling Factor	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95	1.0
Baseline ERIS	114.3	116.0	117.8	122.2	125.1	130.1	135.7	154.0	173.3
CARIC	113.6	115.9	117.5	120.2	124.6	129.9	135.2	151.1	166.8
% Improvement	0.6%	0.1%	0.3%	1.6%	0.4%	0.2%	0.4%	1.9%	3.8%

Table 4.4: Average Fuel Usage (mL) at Scaled Down Demands for Portland-Franklin Network

Demand Scaling Factor	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95	1.0
Baseline ERIS	1.39	1.41	1.44	1.52	1.57	1.67	1.79	2.33	2.96
CARIC	1.35	1.36	1.39	1.43	1.50	1.61	1.71	2.18	2.65
% Improvement	3.5%	3.4%	4.1%	5.9%	4.2%	3.8%	4.7%	6.4%	10.4%

Table 4.5: Average Number of Stops at Scaled Down Demands for Portland-Franklin Network

Before comparing the two strategies, we note the major reductions in delay as demand is decreased. Decreasing demand by 5%, from a scaling factor of 1.0 to a scaling factor of 0.95, leads to a greater than 15% reduction in delay for both strategies. Similarly, decreasing the scaling factor from 0.95 to 0.9 leads to a further greater than 15% reduction in delay for both strategies. Decreases in delay are much smaller as we continue to decrease the scaling factor. At scaling factors of 0.95 and 1.0, the network appears to run very close to its capacity and we observe backup (less frequently at 0.95 than at 1.0).

CARIC, when compared against ERIS, reduces delay by approximately 3.2% and reduces fuel usage by approximately 1.9% at the demand scaling factor of 0.95; however, when performing a two-sided paired t-test with a 95% confidence level, these reductions are not statistically significant. At lower levels of demand, CARIC does not reduce delay (the 1.8% reduction at the weighting factor 0.75 appears to be noise).

Even though CARIC does not reduce delay, we observe that for all scaling factors, CARIC reduces the average number of stops. When performing a two-sided paired t-test with a 95% confidence level, these reductions are statistically significant for all levels of demand except for 0.95. As a benefit of reducing the number of stops, CARIC also offers a consistent reduction in fuel usage at all levels of demands.

The results of this experiment and the previous experiment suggest that green wave coordination plans are most useful when demand is relatively high, approaching the network capacity. When demand is high, green wave coordination plans reduce the likelihood of long queues forming, ultimately reducing delay. When demand is lower, queues are less likely to form and thus the potential benefits are lower. Additionally, green wave coordination plans are restrictive in that they reduce the total set of schedules that can be implemented by the intersections within the network. Overall, at lower demands, the costs outweigh the benefits and delay increases.

However, even when demand is low, green wave coordination plans allow some vehicles (namely those travelling along the green wave route) to travel with fewer stops. This ultimately reduces average fuel usage (across all vehicles) at the expense of increasing average vehicle delay.

4.5.6 Evaluation using Scaled Down Wave Demands

In the previous experiment, we argued that CARIC is more beneficial in terms of reducing delay when the intersection is congested. In this experiment, we consider what happens when the number of vehicles travelling along the green wave route decreases. Recall from above that approximately 11.7% of vehicles travel along the full green wave route. For this experiment, we reduce this percentage by scaling down the number of vehicles travelling along this route.

Our experimental setup is the same as above, except only the number of vehicles travelling along the full green wave route are scaled down using a demand scaling factor. We present average delay in Table 4.6, average fuel usage in Table 4.7, and the average number of stops in Table 4.8. The results from the first evaluation are again included in these tables for easier comparisons.

Wave Demand Scaling Factor	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95	1.0
% of Traffic on Full Wave	7.4%	7.9%	8.5%	9.0%	9.6%	10.1%	10.7%	11.2%	11.7%
Baseline ERIS	75.3	82.9	86.8	88.2	98.7	106.9	117.2	120.9	131.9
CARIC	73.3	80.6	81.8	84.3	93.2	101.3	108.6	115.4	122.3
% Improvement	2.7%	2.8%	5.7%	4.4%	5.6%	5.3%	7.3%	4.6%	7.2%

Table 4.6: Average Delay (s) at Scaled Down Wave Demands for Portland-Franklin Network

Wave Demand Scaling Factor	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95	1.0
% of Traffic on Full Wave	7.4%	7.9%	8.5%	9.0%	9.6%	10.1%	10.7%	11.2%	11.7%
Baseline ERIS	131.1	136.9	140.0	141.4	149.2	155.5	162.5	166.0	173.3
CARIC	128.7	134.8	136.0	138.2	144.5	150.9	156.4	161.5	166.8
% Improvement	1.8%	1.5%	2.9%	2.3%	3.1%	2.9%	3.8%	2.7%	3.8%

Table 4.7: Average Fuel Usage (mL) at Scaled Down Wave Demands for Portland-Franklin Network

Wave Demand Scaling Factor	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95	1.0
% of Traffic on Full Wave	7.4%	7.9%	8.5%	9.0%	9.6%	10.1%	10.7%	11.2%	11.7%
Baseline ERIS	1.75	1.88	1.95	1.96	2.21	2.37	2.60	2.70	2.96
CARIC	1.61	1.75	1.77	1.81	1.99	2.16	2.32	2.49	2.65
% Improvement	7.6%	7.0%	9.1%	7.6%	9.7%	8.8%	10.5%	7.9%	10.4%

Table 4.8: Average Number of Stops at Scaled Down Wave Demands for Portland-Franklin Network

For all examined scaling factors, CARIC reduces delay, fuel usage, and the number of stops when compared against baseline ERIS. When performing two-sided paired t-tests with a 95% confidence level, the differences in delay are statistically significant for wave demand scaling factors of 0.7 and above. For all wave demand scaling factors except 0.65, the differences in fuel usage are statistically significant. For all wave demand scaling factors, the differences in the number of stops are statistically significant.

While the benefits of CARIC decrease as the number of vehicles travelling along the full green wave route decreases, there is still a positive benefit to CARIC. Even when the percentage of vehicles travelling along the full green wave route is below 10%, there may still exist a significant improvement to using green wave coordination plans.

In addition to reducing delay for vehicles travelling along the full green wave route, CARIC also reduces delay for vehicles that travel along part of the green wave route. For example, a vehicle that begins at the first intersection along the green wave route, travels straight through the first three intersections of the green wave route, and then turns right at the fourth intersection of the green wave route will, similar to vehicles travelling along the full green wave route, likely not need to stop after proceeding through the first intersection. This experiment demonstrates that because there are still many vehicles that travel along part of the green wave route, green wave coordination plans continue to be useful even as the number of vehicles using the full green wave route is relatively low.

4.5.7 Discussion

In this section, we examined CARIC along the Portland-Franklin network. First, we evaluated the *Re-Planning* frequency and demonstrated that CARIC performs best when *Re-Planning* every 4 or 8 seconds. We then validated our results and demonstrated that CARIC reduces average vehicle delay, average fuel usage, and the average number of stops at the provided demand associated with the AM rush hour. This is an important result as it demonstrates that CARIC works on a real-world network.

We then compared CARIC to ERIS under lower demand settings. CARIC continued to reduce delay when demand was decreased by 5%, but at larger decreases in demand, CARIC did not reduce delay. However, CARIC reduced the average number of stops and fuel usage when compared against ERIS. A traffic planner who values reducing fuel usage (and the associated pollution) may still wish to consider CARIC under lower congestion settings.

Finally, we compared CARIC to ERIS when the number of vehicles travelling along the full green wave route was decreased. We continued to notice reductions in delay, fuel usage, and the average number of stops when using CARIC. This is in part due to the fact that there are many vehicles that travel along a portion of the green wave route andn are still able to obtain some benefits.

4.6 Conclusion

This chapter presented CARIC, a real-time central agent that coordinates intersections running ERIS. CARIC proposes potential green wave coordination plans for intersections to assess and based on the estimated costs of these plans, selects one for the intersections to enforce. We presented the algorithm which CARIC uses to calculate green wave coordination plans and detailed other activities it performs. We explained necessary modifications to ERIS that allow it to analyze and enact local green wave constraints. We demonstrated that CARIC was able to reduce delay by up to 7.2% and fuel usage by up to 3.8% when compared against ERIS on the Portland-Franklin network, and we discussed how CARIC is most beneficial under heavy demand.

In Chapter 5, we continue to investigate when green wave coordination plans are beneficial. We begin by constructing artificial networks based on the Portland-Franklin network to generate intuition about what makes a network a good candidate for green wave coordination plans. We then train a machine learning model that uses measured demand to determine when to activate green wave coordination plans.

Chapter 5

Considerations for Applying Green Wave Coordination

5.1 Overview

In Chapter 4, we demonstrated that CARIC can reduce average vehicle delay. However, we also observed situations when CARIC increased average vehicle delay. In this chapter, we consider two separate questions to better understand when CARIC and the associated green wave coordination plans are beneficial.

First, we briefly consider two modified networks based on the Portland-Franklin network. We consider whether green wave coordination plans are more beneficial on networks with intersections that are closer together or networks with intersections that are farther apart.

Second, we use real-time demand information to dynamically determine whether or not to trigger green wave coordination plans. We begin by training a support vector machine to predict at which demand combinations green wave coordination plans are beneficial. We then run an experiment demonstrating that adaptively triggering green wave coordination plans (according to the trained model) provides an improvement when compared against the baselines of always running green wave coordination plans and never running green wave coordination plans.

5.2 Network Considerations

5.2.1 Overview

In this section, we examine how the spacing of intersections within a network influences the benefit of green wave coordination plans. We consider the Portland-Franklin network introduced in Chapter 4. We generate two artificial networks based on the Portland-Franklin network. First, we increase the distance between all intersections in the network, but otherwise keep all other network properties unchanged. We refer to this network as the Long-Synthetic-Portland-Franklin network. Second, we generate a shorter network by squeezing together the intersections. Specifically, there are two pairs of intersections that are over 250 meters apart (from the left, the second and third intersection are over 250 meters apart, as are the fourth and fifth intersection). We approximately halve these distances so that all intersections are within 200 meters of each other in our second synthetic network, Short-Synthetic-Portland-Franklin. We present the three separate networks in Figures 5.1 - 5.3 below for comparison. So that all networks can be viewed on the same page, we trim the western and southern approaches to the leftmost intersection in all three figures. Additionally, we align the networks along the leftmost intersection and use the same scale to demonstrate the intersection spacing differences between the networks.

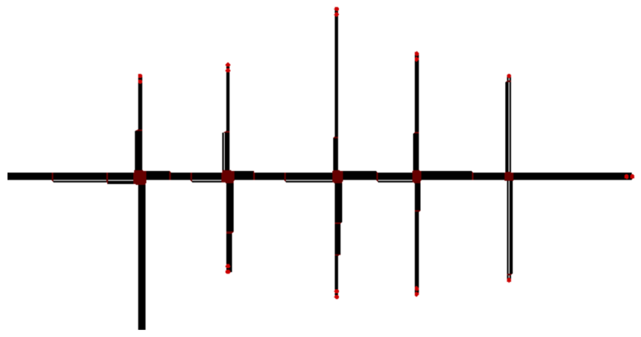


Figure 5.1: Short-Synthetic-Portland-Franklin Network

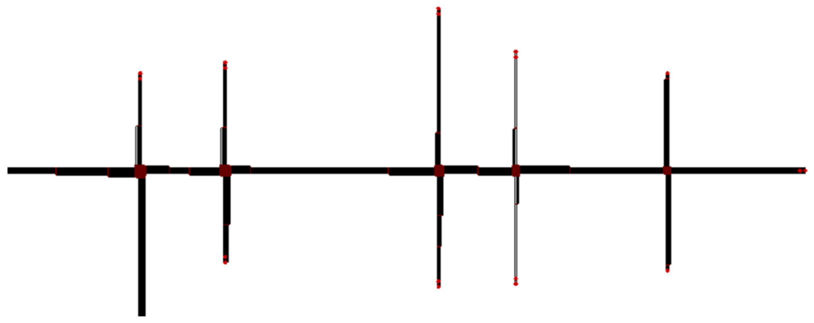


Figure 5.2: Portland-Franklin Network

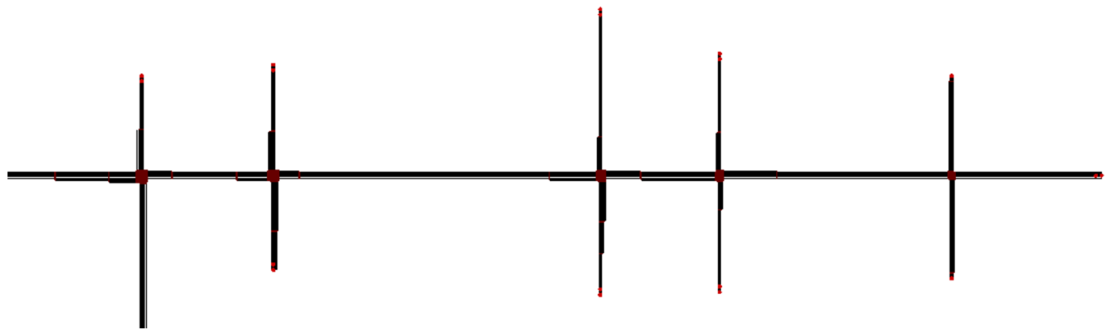


Figure 5.3: Long-Synthetic-Portland-Franklin Network

Other properties of the networks, including the demand profile and the minimum and maximum phase times, are unaffected by these modifications.

5.2.2 Experimental Setup

We compare CARIC against ERIS across these two synthetic networks. Other than using different networks, our experimental setup is identical to the experiment we ran on the original Portland-Franklin network (please refer to Section 4.5.2 for details regarding the experimental setup). Similar to what was done in Section 4.5.2, we analyze CARIC at the provided demand, as well as at several scaled down demands.

5.2.3 Results

Average vehicle delay for the Short-Synthetic-Portland-Franklin network is presented in Table 5.1. For comparison, we reproduce our results regarding average vehicle delay from the original Portland-Franklin network in Table 5.2. Average vehicle delay for the Long-Synthetic-Portland-Franklin network is presented in Table 5.3.

Demand Scaling Factor	0.75	0.8	0.85	0.9	0.95	1.0
Baseline ERIS	47.3	50.9	56.4	68.4	108.1	137.7
CARIC	47.3	50.7	55.7	66.4	99.0	129.0
% Improvement	-0.2%	0.4%	1.2%	3.0%	8.4%	6.3%

Table 5.1: Delay (s) at Scaled Down Demands for the Short-Synthetic-Portland-Franklin Network

Demand Scaling Factor	0.75	0.8	0.85	0.9	0.95	1.0
Baseline ERIS	63.4	66.6	73.2	79.9	104.6	131.9
CARIC	62.3	67.3	73.8	79.9	101.2	122.3
% Improvement	1.8%	-1.0%	-0.8%	0.0%	3.2%	7.2%

Table 5.2: Delay (s) at Scaled Down Demands for the Portland-Franklin Network

Demand Scaling Factor	0.75	0.8	0.85	0.9	0.95	1.0
Baseline ERIS	47.1	50.8	55.5	62.9	83.7	116.9
CARIC	47.5	51.1	56.5	63.4	85.3	116.6
% Improvement	-0.9%	-0.5%	-1.9%	-0.9%	-2.0%	0.2%

Table 5.3: Delay (s) at Scaled Down Demands for the Long-Synthetic-Portland-Franklin

For the Short-Synthetic-Portland-Franklin network, we notice similar results to the original network. For both of these networks, CARIC reduces delay at the two highest demands (scaling factors of 1.0 and 0.95). For the Short-Synthetic-Portland-Franklin network, both of these reductions are statistically significant when performing two-sided paired t-tests with a 95% confidence level. CARIC appears to reduce delay at the two middle scaling factors (0.9 and 0.85) for the Short-Synthetic-Portland-Franklin network, but these results are not statistically significant when performing two-sided paired t-tests with a 95% confidence level. Thus, we conclude that CARIC offers an improvement (compared to ERIS) on the Short-Synthetic-Portland-Franklin network at high levels of demand. Additionally, CARIC may offer a smaller improvement at intermediate levels of demand.

We now consider the results of the Long-Synthetic-Portland-Franklin network (Table 5.3). There is a tiny reduction of 0.2% in delay at a scaling factor of 1.0; this is likely due to noise. At other demand scaling factors, CARIC appears to increase delay. Thus, we conclude that for the Long-Synthetic-Portland-Franklin network, at the provided levels of demand, CARIC does not reduce delay.

5.2.4 Discussion

The results of this experiment suggest that green wave coordination plans are more beneficial when intersections are closer together. We identify three reasons supporting these results.

First, recall that with ERIS, an upstream intersection will share projected arrivals with its downstream neighbors. If the upstream intersection frequently changes its schedule, the downstream intersection may receive vehicles earlier than expected. When intersections are farther apart, the downstream intersection is more likely to have sufficient flexibility in its schedule to accommodate these vehicles upon their arrival (because there is more time before these vehicles arrive to progress through phase minimums and required change-overs). However, when intersections are closer together, they may not have sufficient flexibility to update their schedule. CARIC coordinates the successive start times of nearby intersections to reduce the likelihood that vehicles need to wait at downstream intersections.

Second, when intersections are closer together, the approaches to downstream intersections have less queue capacity, and it is more likely that queues will overflow, causing backup at upstream intersections. Recall that in Chapter 4, when we ran ERIS along the original network, we noticed queues forming at the third (downstream) intersection (from the west) which consequently caused backup at the second (upstream) intersection. CARIC reduced the average queue length at the downstream intersection and thus, reduced the frequency of backup at the upstream intersection. We also observe overflow and backup for the Short-Synthetic-Portland-Franklin network but not for the Long-Synthetic-Portland-Franklin network when running ERIS. As with the original network, for the Short-Synthetic-Portland-Franklin network, CARIC successfully reduces queue build-up at downstream intersections and the frequency of backup at upstream intersections.

Third, for shorter networks, CARIC can more accurately estimate the impact of a green wave coordination plan, which allows it to select better green wave coordination plans. When intersections are farther apart, the travel time between intersections is larger, and thus, the offsets between the start times of local green wave constraints are larger. For the intersection at the end of the green wave route for the Long-Synthetic-Portland-Franklin network, the start time of the local green wave constraint may be up to two minutes after the start of the green wave coordination plan. It is hard for our scheduler to accurately estimate the local green wave constraint's impact on delay at this intersection since the intersection only has arrival information for the next minute. Conversely, for more tightly packed networks, the scheduler can better estimate the impact of local green wave constraints since intersections are more likely to have arrival information for vehicles that arrive while the constraints are active.

5.3 Learning when Green Wave Coordination Plans are Beneficial

In this section, we train a support vector machine (SVM) to predict at what combinations of demand green wave coordination plans are beneficial. After training this model, we demonstrate that adaptively triggering green wave coordination plans according to the output of this model improves upon our two baselines (always running green wave coordination plans and never running green wave coordination plans).

We begin this section by presenting our network and demand distribution. We detail the SVM training procedure and discuss the the trained model. We describe our adaptive strategy. We then detail our experimental setup, present our results, and discuss the results.

5.3.1 Network and Demand Overview

For this analysis, we consider a synthetic network of six evenly spaced intersections. The network is presented in Figure 5.4. All traffic is assumed to travel straight. We run all simulations for an hour of simulated time and report delay, per vehicle, for all vehicles arriving during the middle 40 minutes.

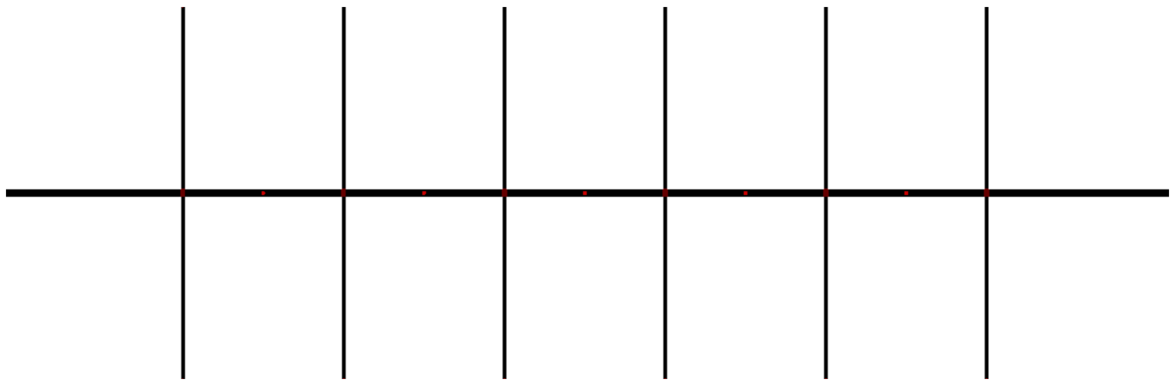


Figure 5.4: Synthetic Network

An exploratory analysis on this network demonstrated that at some combinations of demand, CARIC can offer a 10% improvement when compared against ERIS, yet at other demand combinations, CARIC can increase delay by 10%. (Note that we did not consider the Portland-Franklin network for this analysis because across all levels of considered demand, we observed a maximum of a 2.8% increase in delay due to green wave coordination plans. Thus, we conjectured that an optimally adaptive strategy for this network may be unlikely to be more than 3% better than CARIC. To increase the likelihood of detecting a larger and statistically significant improvement, we decided to conduct our evaluation on a synthetic network).

We generated a highly variable demand distribution for this network. Specifically, we first assign a baseline demand to each of three arrival groups (approximately 1000 vehicles/hour for eastbound traffic, approximately 2800 vehicles/hour for side street traffic, and approximately 350 vehicles/hour for westbound traffic). At this baseline demand, the overall network is not congested and when calculating traffic per lane, the eastbound direction is the busiest.

We then modify all three demands by a randomly chosen shared multiplicative factor (between 1.0 and 1.4). This emulates the fact that some days will be busier than others.

We also modify the demand for each movement group by a random multiplicative factor (eastbound demand by a factor between 1.0 and 1.6, side street demand by a factor between 1.0 and 1.6, and westbound demand by a factor between 1.0 and 3.0). For some random draws of our multiplicative factors, the eastbound and side streets are sufficiently busy that the network becomes congested, suggesting that triggering eastbound green wave coordination plans may offer an advantage. Finally, even if the network is congested, the high variance in our westbound traffic means that in some instances, westbound traffic will approach or exceed eastbound traffic so triggering eastbound green wave coordination plans may no longer be beneficial.

This demand distribution is meant to emulate a network with highly variable demand. For example, consider a network near a sports stadium near the downtown of a city. During evenings without an event, the majority the traffic likely travels out of the city and away from the stadium. However, when there is an event, there will be additional traffic travelling towards the stadium. One may wish to utilize green wave coordination plans out of the city when there is no event, but deactivate green wave coordination plans when there is an event.

5.3.2 Training the Support Vector Machine

In this subsection, we describe the training procedure to train a support vector machine that predicts when green waves are beneficial.

We begin by gathering training data. To obtain a single data point, we generate a random set of vehicle arrivals for a single hour, using the aforementioned demand distribution. We independently run two scheduling strategies (ERIS and CARIC) on the set of arrivals. We compare the associated delays to determine which method leads to a lower total delay. Additionally, we collect the observed vehicle count along each of three arrival groups (eastbound traffic, westbound traffic, and side street traffic). Combining this information provides a single data point with the vehicle counts from each arrival group serving as independent variables and the binary outcome pointing the better (lower delay) strategy as the dependent variable.

We gather 100 such points. We then utilize scikit learn's support vector machine implementation to train a support vector machine with a linear kernal [8].

5.3.3 Outputted SVM Model

Recall that when training a linear SVM, a plane divides our decision area into two regions. For our model, one region suggests that triggering green wave coordination plans is better and the other region suggests that not triggering green wave coordination plans is better. Our trained model suggests that triggering eastbound green wave coordination plans is better when:
(Eastbound Demand)$-1.24*$(Westbound Demand) $-0.12*$(Side Demand) $+158 > 0$.

The signs of the coefficients in this equality are reasonable. Additional vehicles travelling along the green wave route (eastbound) increase the value of the left-hand side of the equation, making the model more likely to predict that activating green wave coordination plans will be the better strategy. Additional vehicles travelling against the green wave route (westbound) and

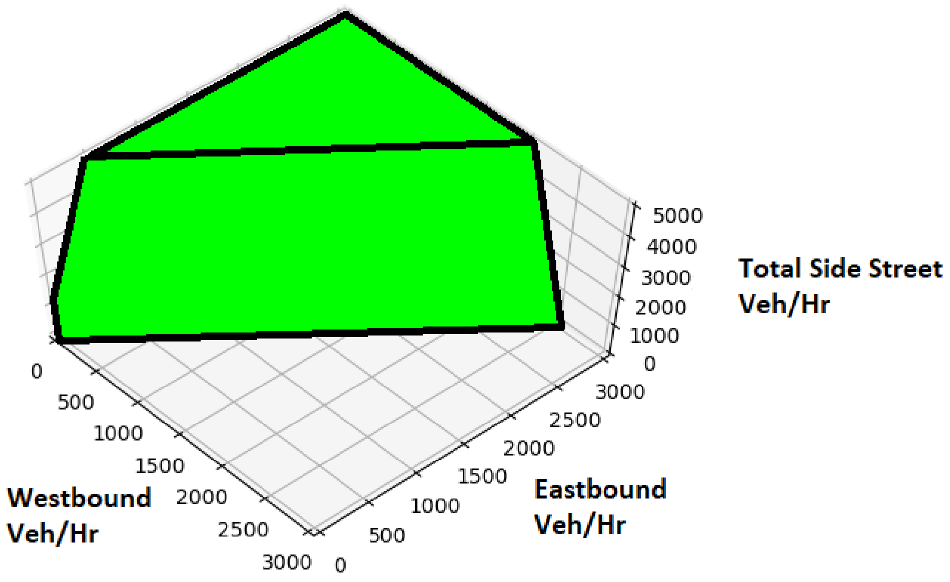


Figure 5.5: Support Vector Machine's Predictions of when to Activate Green Wave Coordination Plans (shown in green)

along the side streets make the model less likely to predict that triggering green wave coordination plans will be the better strategy.

The magnitude of the westbound coefficient is 24% higher than the magnitude of the eastbound coefficient, indicating that an additional vehicle along the westbound direction is more detrimental than an additional vehicle along the eastbound direction is beneficial. The magnitudes of both of these coefficients are approximately 8-10 times higher than the magnitude of the side demand coefficient indicating that the impact of increasing vehicles along the side streets has a relatively smaller impact.

We present a 3-dimensional representation of the decision space from our SVM in Figure 5.5. The plot considers points where eastbound demand is less than 3000 vehicles per hour, westbound demand is less than 3000 vehicles per hour, and side street demand is less than 5000 vehicles per hour. Points where the SVM suggests triggering green wave coordination plans are those enclosed by and on the surface of the green 3-D solid in the graph.

The lower face of the 3-D solid is a subset of the aforementioned plane that divides our decision space. (The upper face is a plane at 5000 side street veh/hr, resulting from our choice of limits for the graph). Our earlier discussion of the decision boundary is confirmed by the graph. As eastbound traffic increases, we move towards the green region. As westbound or side street traffic increases, we move away from the green region.

5.3.4 Adaptive Green Wave Strategy

We run a modified version of CARIC to implement an adaptive green wave strategy, which we refer to as Adaptive-CARIC. Adaptive-CARIC functions the same as CARIC, except at times, Adaptive-CARIC will decide to not plan green wave coordination plans. When Adaptive-CARIC is refraining from *Planning* green wave coordination plans, we refer to Adaptive-CARIC as being in *Standby Mode*. Otherwise (when Adaptive-CARIC is *Planning* green wave coordination plans), Adaptive-CARIC is in *Planning Mode*. We now specify how Adaptive-CARIC switches between these modes.

Adaptive-CARIC begins in *Planning Mode* when the simulation starts.

Every 100 seconds, Adaptive-CARIC calculates the number of vehicles that travelled along each of the three direction categories (eastbound, westbound, and side streets) during the previous 600 seconds. These values are scaled up by a factor of 6 to estimate average hourly demand for each of these three categories. These values are inputted into our trained SVM to determine if green wave coordination plans are predicted to be beneficial.

Based on the previous mode (*Standby Mode* or *Planning Mode*) and the current SVM prediction (beneficial or detrimental), we have four cases, which are all treated differently:

- Adaptive-CARIC is in *Standby Mode* and green wave coordination plans are predicted to be detrimental: Adaptive-CARIC remains in *Standby Mode*.
- Adaptive-CARIC is in *Standby Mode* and green wave coordination plans are predicted to be beneficial: Adaptive-CARIC switches to *Planning Mode*. A green wave coordination plans will be calculated during the next time step. Additional green wave coordination plans will be calculated after the prior green wave coordination plans finishes serving vehicles at the first intersection, as detailed in Chapter 4.
- Adaptive-CARIC is in *Planning Mode* and green wave coordination plans are predicted to be detrimental: Adaptive-CARIC switches to *Standby Mode*. Adaptive-CARIC will stop *Planning* green wave coordination plans. However, Adaptive-CARIC will continue to conduct *Re-Planning* and *Re-Calibration* on existing green wave coordination plans and execute them at their appropriate start time(s).
- Adaptive-CARIC is in *Planning Mode* and green wave coordination plans are predicted to be beneficial: Adaptive-CARIC remains in *Planning Mode*. The next green wave coordination plans is scheduled after the prior green wave coordination plans finishes serving vehicles at the first intersection, as detailed in Chapter 4.

5.3.5 Experimental Evaluation

Experiment Setup

In this subsection, we present our experimental evaluation of Adaptive-CARIC. We compare Adaptive-CARIC against two other strategies, CARIC (which always calculates green wave coordination plans), and ERIS (which does not generate green wave coordination plans) to determine if Adaptive-CARIC outperforms these two baselines when demand across days is variable.

In our experiment, we select a random demand profile from our distribution (detailed in Section 5.3.1) and generate demand according to this distribution. We then evaluate each of the three strategies on this demand during an hour long simulation. We conduct 100 such repetitions, each with a new demand profile.

	Delay (s)	Fuel Usage (mL)
Adaptive-CARIC	69.7	109.6
CARIC	73.9	110.8
ERIS	71.9	111.7
Adaptive-CARIC Improvement VS CARIC	5.7%	1.2%
Adaptive-CARIC Improvement VS ERIS	3.0%	2.0%

Table 5.4: Evaluation of Scheduling Strategies

Results

Average vehicle delay and average fuel usage are presented in Table 5.4.

Adaptive-CARIC reduces delay and fuel usage when compared against both CARIC and ERIS. These differences are statistically significant when performing two-sided paired t-tests with a 95% confidence level. Adaptive-CARIC outperforms ERIS by 3.0% and CARIC by 5.7% in terms of reducing delay. Similarly, Adaptive-CARIC outperforms ERIS by 2.0% and CARIC by 1.2% in terms of reducing fuel usage.

5.3.6 Discussion

Our experiment demonstrates that Adaptive-CARIC offers small yet statistically significant improvements when compared against ERIS and CARIC for the selected demand distribution.

However, this result comes with several caveats. First, these improvements are relatively smaller than other improvements presented throughout this thesis, so time may be better spent considering other improvements.

Second, our network had a very specific demand distribution. Sometimes, traffic was considerably higher along the green wave route. However, at other times, traffic was roughly equal along the green wave route and opposing the green wave route.

We suggested that this behavior may reflect traffic near a sports stadium; however, this likely does not apply to many networks. Many networks experience heavy traffic in one direction for the AM rush hour and heavy traffic in the other direction for the PM rush hour. When this is the case, one may simply be better off activating green wave coordination plans in the corresponding direction during the AM rush hour, de-activating green wave coordination plans around lunch time, and then activating green wave coordination plans in the opposite direction during the PM rush hour.

5.4 Conclusion

This chapter and Chapter 4 demonstrated that green wave coordination plans can reduce delay across a network. In Chapter 4, we presented CARIC and demonstrated that it is most beneficial when demand is relatively high. In this chapter, we presented two additional experimental analyses to consider when green waves offer an improvement.

First, we considered networks with different amounts of space between intersections. We found that green waves are beneficial when intersections are close together and may be detrimental when intersections are spaced farther apart. We hypothesized that scheduler reaction time, queue capacity, and wave cost estimation were all factors supporting these results.

In our second experimental analysis, we trained a support vector machine to estimate when green wave coordination plans are beneficial. Our model suggested that as traffic increases along the green wave route, green wave coordination plans are more likely to offer an improvement. As traffic increases along other movements, green wave coordination plans are less likely to offer an improvement. We then presented Adaptive-CARIC which uses the trained model to decide whether or not to schedule green wave coordination plans. We observed a small improvement when compared against our baselines. However, we noted that this effect was only for a highly variable demand distribution.

While our experiments demonstrate the utility of CARIC and green wave coordination plans, they also highlight several limitations that are not present for ERIS or other schedule-driven control methods, such as Surtrac2012. First, ERIS and Surtrac2012 tend to offer an improvement regardless of network geometry or congestion. However, CARIC requires that intersections are tightly spaced, limiting the percentage of networks where green wave coordination plans are beneficial. When intersections are farther apart, intersections at the end of the green wave pattern may not have cluster information when estimating the local impact of green wave coordination plans.

Second, CARIC only reduces delay for a subset of demand profiles, so the demand profile must be considered when deciding whether or not to implement CARIC. The networks we examined had a dominant flow travelling eastbound. When there is not a dominant flow, CARIC may increase the delay for vehicles not travelling along the green wave route more than it decreases delay for vehicles travelling along the green wave route, ultimately leading to an increase in delay.

Even for a network with tightly spaced intersections and a dominant flow, it is feasible that the relative congestion across different movements gradually changes over time and green wave coordination plans no longer remain beneficial. Adaptive-CARIC offers the ability to deactivate green wave coordination plans once they are detected to no longer be beneficial. Alternatively, one could train a more complicated model to make a larger set of decisions, namely a model which both adaptively activates green wave coordination plans and adaptively selects which intersections to include in the green wave route for the green wave coordination plan. Implementing such a model in the field may require a human-in-the-loop to occasionally (once or several times per year) develop up-to-date demand profiles and train a new model to update Adaptive-CARIC. As such, much of the “hands-off” resiliency of decentralized schedule-driven approaches (like ERIS) is lost.

Third, we note that green wave coordination plans are brittle to changes in the queues at interior intersections. Our model assumes that all vehicles enter the network from specific origin points and then travel along the full approach. However, in reality, vehicles may enter the network along the middle of a block (e.g. by leaving an on-street parking spot). These vehicles can unexpectedly increase the queue length at interior intersections along the green wave pattern. Intersections must serve these vehicles before the vehicles travelling along the green wave route can pass, but there may not be enough time to re-optimize or otherwise modify the phase start times to ensure that queues have sufficient time to clear before the vehicles travelling along the full green wave route arrive. This can cause some vehicles to decelerate; there is no longer a guarantee that these vehicles will not need to stop at this intersection or at a later intersection along the green wave pattern.

Additionally, it may be the case that vehicles unexpected arrive along a left-turn movement and delay the start time of the green wave phase. For example, suppose there is an eastbound green wave coordination plan. Also, suppose that there are no vehicles travelling from the east turning left. Interior intersections will calculate a schedule that skips over this phase (transitioning from the phase serving the northbound and southbound straight movements to a phase serving the eastbound straight movement). However, if vehicles unexpectedly arrive along from the east to turn left, time will now need to be spent serving these vehicles. Vehicles travelling along the green wave route will be delayed at this intersection and will arrive at downstream intersections later than expected.

Finally, our analysis only considered applying green wave coordination plans across an entire network. For future work, we propose examining whether shorter green wave coordination plans that only consider two intersections can provide benefits without the limitations mentioned above. First, when a green wave coordination plan only optimizes two intersections, less communication is necessary. Second, when only two intersections are considered, the start times of the local green wave constraints are closer to the present so more up-to-date information can be used when calculating their cost. Third, since the second intersection will begin soon after the first intersection (and there are no other intersections beginning later), there is a lower chance that vehicles arrive at interior intersections to unexpectedly turn left to disrupt the planned phase start time along the green wave route. We envision a system where the network uses observed congestion to dynamically select pairs of intersections to group together to coordinate. When congestion changes, these intersections can easily be ungrouped and possibly another pair of intersections can be grouped.

Given the aforementioned considerations, CARIC is more challenging to implement than a decentralized system such as ERIS or Surtrac2012. In the third part of this thesis, we will consider several other extensions to improve upon baseline ERIS that do not require as much upkeep as CARIC.

Part III

Addressing Inefficiencies of the ERIS Scheduler

Chapter 6

Three Proposed Extensions to Reduce Inefficiencies of the ERIS Scheduler

6.1 Overview

This chapter considers two inefficiencies with the ERIS scheduler and proposes three extensions to address these inefficiencies. These inefficiencies are closely related to two of the challenges presented in Chapter 1:

5. **Finite Sensing Horizon** - A scheduler only schedules the vehicles it has sensed through its detection system, which has limited range, or from shared information between intersections. We refer to this as the finite sensing horizon. Vehicles outside of the finite sensing horizon will continue to arrive at the intersection and most methods fail to consider the impact of their decisions on these vehicles.
6. **Inadequate Objective Function** - A scheduler that minimizes delay every time step will not necessarily minimize delay during the entire evaluation period (even if the evaluation period is as short as 5 minutes). A decision that minimizes delay over the scheduling horizon may cause queues to form along some movements, ultimately increasing delay during future time steps.

The ERIS Scheduler only schedules vehicles within its finite sensing horizon. For our first proposed extension, we will investigate whether including an estimate of vehicles outside of the sensing horizon allows ERIS to better estimate the delay for schedules and ultimately select better schedules.

ERIS' minimization of delay every time step occasionally allows queues to form along some movements, ultimately causing congestion and increasing delay during future time steps. For our second proposed extension, we will consider a more complex objective function that optimizes a weighted average of delay and schedule makespan to determine whether this encourages ERIS to prioritize shorter schedules, ultimately allowing ERIS to serve vehicles more efficiently.

Our third extension also considers ERIS' objective function. For our third extension, we will consider whether weighting the objective function to prioritize or de-prioritize the vehicles travelling along pre-selected movements allows ERIS to more efficiently allocate time across movements. For example, if ERIS spends too little time serving a particular movement and queues form, prioritizing the importance of the movement may reduce the frequency of queues forming, ultimately decreasing delay.

We note that these two inefficiencies apply to most schedule-driven control methods, not solely ERIS. (For example, Surtrac2012 also uses a finite sensing horizon and minimizes delay over the scheduling horizon). While we only examine the impact of our proposed extensions on ERIS, we would expect similar modifications to provide a similar impact to other scheduling techniques.

This chapter is a high-level overview and considers these three extensions to reduce the aforementioned inefficiencies. Chapter 7 will examine the most promising of these three strategies in additional detail.

In this chapter, we begin by presenting a motivating example demonstrating the potential of each extension. We define our three candidate methods to reduce delay (adding phantom vehicle clusters, penalizing schedule makespan, and weighting movements). We then present several experiments comparing these three methods on several two-movement single intersection networks. We briefly detour to define critical movements and then present several experiments comparing these three methods on a larger intersection with critical movements. We find a statistically significant improvement for the weighting movement strategies. We conclude by summarizing our experiments and overviewing additional weighting evaluation that will be explored in Chapter 7.

6.2 Motivating Example

We begin by considering the following example demonstrating why these extensions may improve solution quality.

Consider the intersection in Figure 6.1 with two lanes along the eastbound movement and one lane along the southbound movement. To ease the presentation, we assume a 5 second change-over time, no start-up lost time, and that each vehicle requires 2 seconds to clear the intersection. Additionally, we assume that the first vehicle along each movement is 5 seconds away from the intersection while all other vehicles are 2 seconds farther away than the preceding vehicle. (So the final vehicle along the eastbound movement is 15 seconds away from the intersection).

A scheduler has two reasonable options.

1. First, the intersection could remain green and serve all vehicles along the eastbound movement. Because the vehicles along the eastbound movement are spaced two seconds apart, starting at time 5, every two seconds, one vehicles will pass through the intersection. The final vehicle will finish passing through the intersection at time 17. Then, the southbound vehicle will be served starting at time 22 (17 + 5 second change-over) for a total delay of 17 seconds (departure time of 22 minus arrival time of 5). Note that this final vehicle finishes passing through the intersection at time 24, which is the makespan of the schedule.
2. The other option is to immediately switch to a red phase so that the intersection can serve the southbound vehicle, when it arrives, at time 5. This will require 2 seconds. After another 5 second change-over, the eastbound vehicles will begin at time 12. The vehicles along the eastbound movement will have bunched up by time 12. Every two seconds, two vehicles (one from each lane) will pass through the intersection. The first two vehicles will begin to cross at time 12, the next two at time 14, and the final two at time 16. This leads

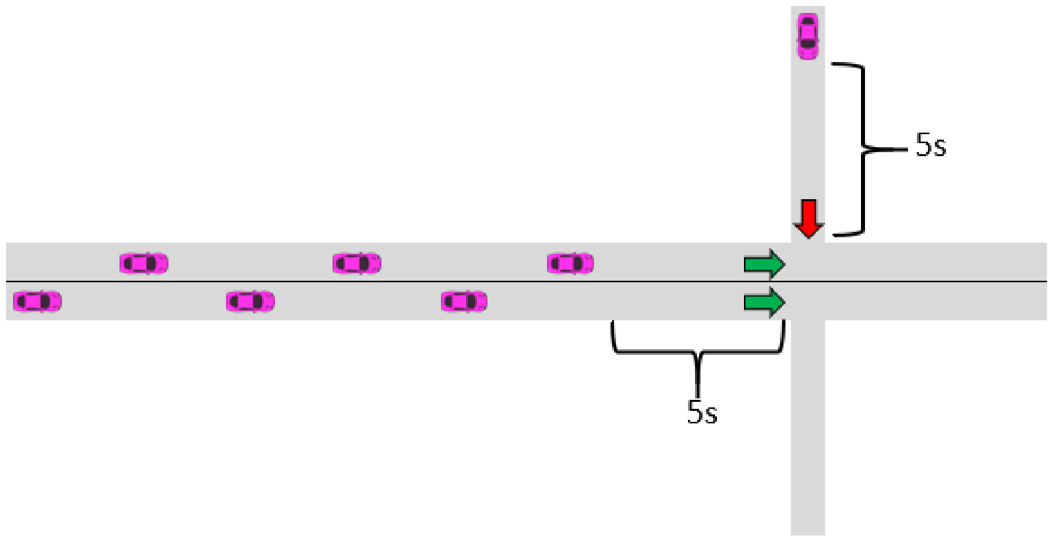


Figure 6.1: Motivating Example

to a total delay of $(12-5)+(12-7)+(14-9)+(14-11)+(16-13)+(16-15) = 24$ seconds. Also note that the final vehicles finish passing through the intersection at time 18, which is the makespan of the schedule.

ERIS will select the first option since it has a lower delay than the second option $(17 < 24)$. In the case that no more vehicles arrive in the next 20 seconds, this will be the better selection since it leads to a lower delay.

However, if over the next 20 seconds, more vehicles arrive, the first option may turn out to be the wrong choice. For the first option, along the eastbound movement, ERIS served one vehicle at a time (whereas the second option served these vehicles two at a time). The additional time required to serve these eastbound vehicles may cause a larger queue to build-up along the southbound movement, ultimately increasing delay.

6.3 Proposed Extensions

In this section, we introduce our three proposed extensions to ERIS. For each extension, we present motivating intuition, provide a high-level overview, explain how it would alter the decision from the example presented above, and then formalize our implementation details.

6.3.1 Phantom Clusters

Motivation & Extension Overview

ERIS' delay minimization only includes the delay for known vehicle clusters, yet as discussed above, we are also concerned about ERIS' impact on the delay for unknown vehicle clusters that will arrive in the future. We hypothesized that if ERIS includes an estimate for future vehicles in its vehicle clusters, ERIS will approximate the impact of its decisions on the delay for future vehicles and be less likely to make inefficient decisions.

By calculating empirical arrival rates for each movement of the intersection, we can approximate how many vehicles will join the queue of a given lane over a given time period. We incorporate logic in ERIS' Cluster Construction Module to use these empirical arrival rates to generate phantom vehicle clusters along each lane.

Changes to Motivating Example

Let us return to the motivating example presented earlier. Suppose we learned that the average arrival rate on the eastbound approach was quite high, and decided to add a phantom cluster of 6 vehicles arriving just after the end of the real cluster. A graphic of the intersection, with the phantom cluster in blue, is presented below in Figure 6.2.

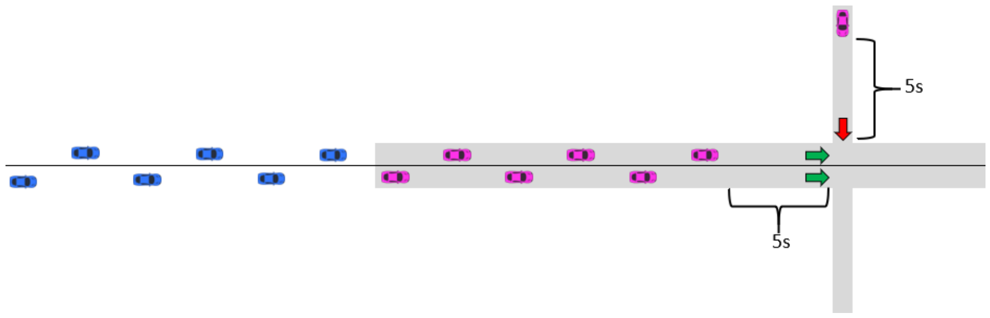


Figure 6.2: Modifying the Motivating Example with a Phantom Cluster

We revisit the two scheduling options presented above taking into account the phantom cluster.

1. First, the intersection could remain green and serve all vehicles along the eastbound movement. Because the vehicles along the eastbound movement are spaced two seconds apart, starting at time 5, every two seconds, one vehicles will pass through the intersection. The final vehicle will finish passing through the intersection at time 29. Then, the southbound vehicle will be served at time 34 (29 + 5 second change-over) for a total delay of 29 (34-5) seconds.
2. The other option is to immediately switch to a red phase so that the intersection can serve the southbound vehicle, when it arrives, at time 5. This will require 2 seconds. After another 5 second change-over, the eastbound vehicles will begin at time 12. The vehicles along the eastbound movement will have bunched up by time 12. The first two vehicles (one in each lane) will begin to cross at time 12, the next two at time 14, the next two at time 16, and the next one at time 18. The remaining vehicles will not arrive until there are no vehicles in front of them and will not need to wait. This leads to a total delay of $(12-5)+(12-7)+(14-9)+(14-11)+(16-13)+(16-15)+(18-17)+5*0 = 25$ seconds.

Running ERIS with the phantom cluster extension will select the second option since it has a lower delay than the first option ($25 < 29$). ERIS is deliberately being more efficient with its capacity and thus may calculate schedules that reduce long-run delay.

Implementation Details

There are a variety of parameters one must select when determining how to generate phantom clusters. We must consider how many phantom clusters to generate, which movements to generate phantom clusters along, and each cluster's arrival time, size, flow rate, and relative importance.

Here, we present each parameter individually and detail the range of choices we consider for the parameter. In our presentation of the experiments, we list the combinations of parameters that we evaluate.

- We parameterize the number of phantom clusters to generate. We will generate 1 - 3 phantom clusters for each selected movement.
- We must consider which movements receive phantom clusters. For most of our examination, all movements will receive phantom clusters. However, we will also consider only assigning phantom clusters to a subset of movements known as critical movements (a term which will be defined later in this chapter).
- We must specify the "arrival time" for the first phantom cluster along each selected movement. We specify that for each movement, the first phantom cluster will arrive at time 30.
- We must specify a method to generate the number of vehicles in a phantom cluster. For each movement, we calculate the number of vehicles that we expect to arrive during an average 30 second interval. To obtain this average, we multiply 30 by the average number of vehicles arriving every second.
- We must specify the flow rate of a phantom cluster. We will use the same average flow rate used for actual vehicles.
- If we are generating multiple clusters, we must specify the arrival times for future clusters. In our evaluation, phantom clusters are 30 seconds apart. Thus, the second cluster (if one exists) "arrives" at time 60. The third cluster (if one exists) "arrives" at time 90.
- We consider the relative importance for phantom clusters (when compared to regular clusters). We consider relative importances between 0.5 - 1.25 of the importance of a regular cluster.

6.3.2 Penalizing Higher Makespans

Motivation & Extension Overview

In our motivating example, we observed that the schedule with the lower delay (option #1) had a higher makespan. Recall that along the eastbound movement, this schedule served one vehicle every two seconds (whereas the other schedule served two vehicles every two seconds), leading to a higher makespan.

We hypothesized that schedules with shorter makespans may use capacity more efficiently (e.g. serving two vehicles at a time instead of one) and thus may lead to better long-run delay outcomes. By modifying our objective function to consider both delay and schedule makespan, we may be able to encourage ERIS to select schedules that are more efficient.

Changes to Motivating Example

Let us again return to the motivating example presented earlier. Suppose that instead of selecting the schedule based on minimum delay, we select the schedule based on a weighted average of delay and makespan where delay is weighted by 25% and makespan is weighted by 75%.

We revisit the two options presented above.

1. First, the intersection could remain green and serve all vehicles along the eastbound movement. Because the vehicles along the eastbound movement are spaced two seconds apart, starting at time 5, every two seconds, one vehicles will pass through the intersection. The final vehicle will finish passing through the intersection at time 17. Then, the southbound vehicle will be served starting at time 22 (17 + 5 second change-over) for a total delay of 17 seconds (departure time of 22 minus arrival time of 5). Note that this final vehicle finishes passing through the intersection at time 24, which is the makespan of the schedule. Thus, our final objective is $17 * .25 + 24 * .75 = 22.25$.
2. The other option is to immediately switch to a red phase so that the intersection can serve the southbound vehicle, when it arrives, at time 5. This will require 2 seconds. After another 5 second change-over, the eastbound vehicles will begin at time 12. The vehicles along the eastbound movement will have bunched up by time 12. Every two seconds, two vehicles (one from each lane) will pass through the intersection. The first two vehicles will begin to cross at time 12, the next two at time 14, and the final two at time 16. This leads to a total delay of $(12-5)+(12-7)+(14-9)+(14-11)+(16-13)+(16-15) = 24$ seconds. Also note that the final vehicles finish passing through the intersection at time 18, which is the makespan of the schedule. Thus, our final objective is $24 * .25 + 18 * .75 = 19.5$.

Running ERIS with the modified objective function will now select the second option ($19.5 < 22.25$).

Implementation Details

We modify our ERIS code in two steps. First, we modify the g-value to now be a weighted average of delay and makespan.

Note that delay and makespan may have different scales. Makespan is the schedule duration and will tend to be under 180 seconds. Delay can be a factor of 10 or even 100 higher than makespan. As such, we will scale these values by their averages observed over recent time steps. At each intersection, we will maintain a moving average of both average delay and average makespan. Our modified objective function becomes:

$$w * \frac{\text{schedule makespan}}{\text{average makespan}} + (1-w) * \frac{\text{schedule delay}}{\text{average delay}}$$

w represents the relative weighting factor between makespan and delay (e.g. a weight of 1 indicates to solely use makespan). In our experimental analysis, we will examine multiple values for w.

Second, as a result of modifying our objective function, we must also modify our heuristic function. We continue to run the existing heuristic pipeline to calculate a lower-bound on delay. However, we modify Step 6, where we calculate the delay for individual scheduling problems, to also calculate a lower-bound on makespan within each sub-problem. From the two sub-problems, we take the higher of these values as a lower-bound on schedule makespan. We subtract the current time, T, from the lower-bound on schedule makespan to calculate the minimum amount of remaining schedule duration. Finally, we combine the minimum remaining schedule duration with the total unavoidable delay using the above weighting formula to obtain a lower bound on unavoidable cost (our h-value). This modified heuristic is then incorporated with the newly calculated cost to calculate a state's f-value.

6.3.3 Weighting Approaches

Motivation & Extension Overview

In our motivating example, we observed that when the eastbound movement occurred first, it served vehicles one at a time, half of its maximum flow rate of two vehicles at a time. We may wish to discourage the intersection from serving a movement when the movement will not be served at its maximum flow rate. We hypothesized that if an intersection serves individual movements more efficiently, less time will be spent on each movement, queues will be less likely to form along other movements, and ultimately, overall delay will decrease.

For our third extension, we consider downgrading the importance of selected movements using a constant weighting factor less than 1 to discourage ERIS from inefficiently serving movements.

Changes to Motivating Example

Once more, we revisit our motivating example and consider how the choice of a weighting strategy can change ERIS' decision. Suppose we downgrade the importance of each movement by the inverse of the number of lanes. The eastbound movement is weighted by $\frac{1}{2}$ (because it has two lanes) while the southbound movement is weighted by 1 (because it has one lane).

1. First, the intersection could remain green and serve all vehicles along the eastbound movement. Because the vehicles along the eastbound movement are spaced two seconds apart, starting at time 5, every two seconds, one vehicles will pass through the intersection. The final vehicle will finish passing through the intersection at time 17. Then, the southbound vehicle will be served starting at time 22 (17 + 5 second change-over) for a total delay of 17 seconds (departure time of 22 minus arrival time of 5). Thus, our objective, after applying the appropriate weightings, is $1.0 * 17 + 0.5 * 0 = 17$.
2. The other option is to immediately switch to a red phase so that the intersection can serve the southbound vehicle, when it arrives, at time 5. This will require 2 seconds. After another 5 second change-over, the eastbound vehicles will begin at time 12. The vehicles along the eastbound movement will have bunched up by time 12. Every two seconds, two vehicles (one from each lane) will pass through the intersection. The first two vehicles will begin to cross at time 12, the next two at time 14, and the final two at time 16. This leads to a total delay of $(12-5)+(12-7)+(14-9)+(14-11)+(16-13)+(16-15) = 24$ seconds. Thus, our final objective, after applying the appropriate weightings, is $1.0*0+0.5*24 = 12$.

Running ERIS with the proposed weighting strategy will now select the second option $(12 < 17)$.

Implementation Details

Previously, all vehicles were weighted equally. We adapt our delay function to include a movement-specific weighting term for each vehicle. Specifically, our new delay function for a cluster becomes:

$$\text{Weighted Delay} = (\text{Vehicles in Cluster}) * (\text{Delay Per Vehicle}) * (\text{Movement Weight})$$

When calculating the total delay for each candidate schedule, we will calculate the total weighted delay (instead of the regular delay) across all clusters.

Note that for the remainder of this document, when a movement receives a weight of 1.0, it will be referred to as being "unweighted," since it is being treated normally. When a movement receives a weight that is not 1.0, we will refer to this movement as "weighted." The majority of our experimental analysis will use a weight below 1.0. When this is the case, we are reducing the importance of the movement receiving this weight. We refer to this as "downgrading" the importance of the movement.

In our experimental evaluation, will consider two questions regarding weighting. First, we consider which movements should be weighted. In the above example, we weighted the movement with two lanes and did not weight the movement with one lane. When we consider an intersection with 8 different movements, each movement can either be weighted or not be weighted. There are 2^8 possible combinations of weighted/not weighted. Weighting all movements by the same value is equivalent to weighting none of the movements, so we have $2^8 - 1 = 255$ possible weight/no-weight strategies. (If all clusters are weighted by the same factor, all schedules generated by the search, including incomplete partial schedules, will have f-values that are scaled by this shared weighting factor but otherwise the same as what they would be if no weighting occurred). In one of our experiments below, we examine all 255 possible weighting strategies.

Second, we consider what weighting factor to use. In the above example, we used a weighting factor of 0.5. In our experiments, we examine a variety of weighting factors. (While these weighting factors will vary within experiments, within each trial, we use the same weighting factor for each weighted movement. However, we note that this is not necessary).

ERIS' Cluster Construction module is modified to run a cluster weighting procedure to enable this extension. After constructing clusters, ERIS weights all clusters according to the movement's weighting factor. A cluster's size and flow rate are both multiplied by the appropriate weighting factor. Desired cluster start time is unchanged. Note that weighting a cluster's size means that the cluster's delay which filters into the overall objective will now be appropriately weighted. We must also change the flow rate to ensure that the cluster requires the correct amount of time to pass through the intersection. After making these modifications, no other changes to ERIS are necessary. (We do not need to change the objective function or heuristic calculation pipeline).

6.4 Related Work

Others have realized that minimizing local delays for a schedule-driven control method do not necessarily minimize overall delay. For example, Guler et al. (2014) presents a schedule-driven control method for connected vehicles. In their experiments, they find that minimizing the number of stops at each time step leads to a lower total delay than trying to directly minimize the delay at each time step.

Several works discuss prioritizing selected movements using different weights. Gregoire et al. (2014) proposes weighting movements by the multiplicative inverse of their capacity before running a Max Pressure algorithm. The authors define capacity as the product of the number of lanes for this movement and the average number of vehicles that can fit in a lane.

Two works from Hu and Smith (2017a, 2017b) propose using dynamic weights for each movement based on relative congestion before running Surtrac. These dynamic weights are calculated using effective queue length, which is a function of the current queue length for a movement, plus a factor for vehicles that are expected to arrive along the movement, minus a factor for vehicles waiting at downstream intersections. We employ the same modified cluster delay function as Hu and Smith (2017a, 2017b), but utilize a fixed movement weighting function similar to Gregoire et al. (2014).

6.5 Experimental Analysis on Two-Movement Intersections

This section presents our first experimental evaluation of the three extensions. Specifically, in this section, we examine these extensions on small intersections with two movements. Our experiments on two of these intersections fail to demonstrate an improvement from the extensions. However, our experiment on a third intersection demonstrates improvement for each extension and we explain the difference between this result and the prior results. The intuition gained from these experiments motivates our next set of experiments.

6.5.1 1x1 and 2x2 Intersections

In our first experiment, we examine our proposed extensions on an intersection with two lanes - one southbound lane and one eastbound lane (which we refer to as the 1x1 Intersection). Similarly, in our second experiment, we examine our proposed extensions on a intersection with four lanes - two southbound lanes and two eastbound lanes (which we refer to as the 2x2 Intersection). The average vehicle delay when running ERIS with these extensions on the 1x1 Intersection are presented in Table 6.1. Similarly, the average vehicle delay when running ERIS with these extensions on the 2x2 Intersection are presented in Table 6.2.

For each extension, we consider four possible parameter settings. For weighting approaches, we weight the eastbound movement by factors of 0.6, 0.8, 1.2, and 1.4 (the first two weights correspond to downgrading the importance of this movement, the final two weights correspond to increasing the importance of this movement). For makespan, we consider a wide range of weighting factors in our objective function for the makespan term. Specifically, we consider the following weighting factors: 0.15, 0.3, 0.6, and 0.9. For phantom clusters, we generate 1, 2, or 3 phantom clusters along all movements with the same importance as real clusters and generate 1 phantom cluster along all movements that is half as important as real clusters (all other parameters are fixed and detailed in Section 6.3.1).

For each parameter setting, we examine how the setting influences delay when demand is uniform and when demand is heavier across one of the movements. Additionally, we present a comparison to the baseline ERIS scheduler (as described in Chapter 3). Negative percentages represent the reduction in delay compared to ERIS; positive percentages represent an increase in delay.

	Even Demands		Eastbound Busier	
Method	Delay (s)	vs Baseline	Delay (s)	vs Baseline
Baseline (ERIS)	37.0	0%	34.2	0%
Eastbound Weight = .6	61.4	66%	48.5	42%
Eastbound Weight = .8	37.5	1%	34.8	2%
Eastbound Weight = 1.2	37.5	2%	35.9	5%
Eastbound Weight = 1.4	38.1	3%	37.8	10%
Makespan Factor = 0.15	36.9	0%	34.3	0%
Makespan Factor = 0.3	37.7	2%	33.8	-1%
Makespan Factor = 0.6	37.1	0%	34.9	2%
Makespan Factor = 0.9	40.8	10%	38.7	13%
1 Phantom Cluster	37.1	0%	34.4	0%
2 Phantom Clusters	36.7	-1%	34.6	1%
3 Phantom Clusters	37.2	1%	36.1	6%
1 Phantom Cluster, half as important	36.6	-1%	34.9	2%

Table 6.1: Average Vehicle Delay (s) at 1x1 Intersection

	Even Demands		Eastbound Busier	
Method	Delay (s)	vs Baseline	Delay (s)	vs Baseline
Baseline (ERIS)	39.1	0%	38.1	0%
Eastbound Weight = .6	55.3	42%	45.8	20%
Eastbound Weight = .8	40.2	3%	38.6	1%
Eastbound Weight = 1.2	39.9	2%	39.4	3%
Eastbound Weight = 1.4	41.2	5%	39.7	4%
Makespan Factor = 0.15	39.6	1%	38.3	0%
Makespan Factor = 0.3	39.7	2%	38.1	0%
Makespan Factor = 0.6	39.5	1%	38.6	1%
Makespan Factor = 0.9	40.7	4%	40.3	6%
1 Phantom Cluster	39.5	1%	38.0	0%
2 Phantom Clusters	39.5	1%	38.7	1%
3 Phantom Clusters	39.6	1%	38.0	0%
1 Phantom Cluster, half as important	39.0	0%	38.0	0%

Table 6.2: Average Vehicle Delay (s) at 2x2 Intersection

For both experiments, we observe that none of the extensions reduce delay by more than 1% (compared to the baseline). Additionally, the majority of the extensions actually increase delay. We discuss this result in more detail after presenting another experiment.

6.5.2 2x1 Intersection

In our third experiment, we examine the proposed extensions on an intersection with three lanes - one southbound lane and two eastbound lanes (which we refer to as the 2x1 Intersection). The average vehicle delay when running ERIS with these extensions on the 2x1 Intersection are presented in Table 6.3.

As above, for each extension, we consider four possible parameter settings. For weighting approaches, we wish to downgrade the importance of the eastbound movement, since it has more lanes than the other movement (as we did in the motivating example). We downgrade the importance of the eastbound movement using weighting factors of 0.4, 0.5, 0.6, and 0.8. Our parameter selections for the other two extensions are the same as the two experiments presented above.

For each parameter setting, we examine how the setting influences delay when demand is uniform and when demand is heavier across one of the movements. Additionally, we present a comparison to the baseline ERIS scheduler (as described in Chapter 3). Negative percentages represent the reduction in delay compared to ERIS; positive percentages represent an increase in delay.

	Even Demands		East Busier	
Method	Delay (s)	vs Baseline	Delay (s)	vs Baseline
Baseline (ERIS)	37.6	0%	30.2	0%
Eastbound Weight = .4	27.1	-28%	28.2	-7%
Eastbound Weight = .5	25.5	-32%	26.4	-13%
Eastbound Weight = .6	25.4	-33%	25.7	-15%
Eastbound Weight = .8	26.1	-31%	26.3	-13%
Makespan Factor = 0.15	36.4	-3%	30.1	0%
Makespan Factor = 0.3	38.2	1%	28.7	-5%
Makespan Factor = 0.6	29.5	-22%	27.5	-9%
Makespan Factor = 0.9	27.4	-27%	27.9	-8%
1 Phantom Cluster	36.2	-4%	28.9	-4%
2 Phantom Clusters	26.4	-30%	27.1	-10%
3 Phantom Clusters	27.3	-28%	27.1	-10%
1 Phantom Cluster, half as important	38.6	2%	29.3	-3%

Table 6.3: Average Vehicle Delay (s) at 2x1 Intersection

In contrast to the previous experiments, we notice that the majority of the parameter settings reduce delay when compared to the baseline. Weighting the eastbound movement by 0.6 is the best setting of all examined, for both demand distributions. For this weighting factor, we notice a delay reduction of 33% for the balanced demand case and a delay reduction of 15% for the imbalanced demand case. While the improvement is not as large, the best makespan factor offers a delay reduction of 27% for the balanced demand case and a delay reduction of 9% for the imbalanced demand case. Similarly, the best phantom cluster settings offers a delay reduction of 30% for the balanced demand case and a delay reduction of 10% for the imbalanced demand case.

6.5.3 Discussion

The above experiments demonstrated that all three methods offer an improvement for the 2x1 Intersection but not for the 1x1 Intersection or the 2x2 Intersection. We hypothesize that is due to an asymmetry in the relative importance of serving a vehicle on a given movement that occurs only at the 2x1 Intersection.

Let us denote τ as the time required to serve one vehicle (for all three intersections). When the 1x1 Intersection serves one vehicle over another, the total length of all queues increases by one vehicle regardless of which vehicle is prioritized. Since vehicles are served one at a time in this intersection, the total amount of time required to serve all queued vehicles increases by τ, regardless of which vehicle is served first.

For the 2x2 Intersection, if we assume that vehicles arriving at the intersection will join the shorter queue if one exists and otherwise join one at random, then when a vehicle is added to a queue, the amount of time required to serve the queue is increased by an average of $\frac{\tau}{2}$. Half of the time, the vehicle joins the shorter queue and the required time is not increased. The other half of the time, queues are the same length and the time is increased by τ. On average, regardless of which movement is served, the delayed vehicle increases the amount of time required to serve all queued vehicles by $\frac{\tau}{2}$.

However, the increase in the amount of time required to serve all queued vehicles is not equal when considering the 2x1 Intersection. If a vehicle arrives along both movements and ERIS serves the vehicle along the eastbound movement (the movement with two lanes), the delayed vehicle along the southbound movement increases the amount of time to serve all queued vehicles by τ. If ERIS serves the vehicle along the southbound movement, then the vehicle along the eastbound movement is delayed, and the average time required to serve all queued vehicles is increased by $\frac{\tau}{2}$. Thus, for the 2x1 Intersection, serving a single vehicle along the southbound movement is more important than serving a single vehicle along the eastbound movement.

The three extensions discourage ERIS from serving the eastbound movement, resulting in more time spent serving the southbound movement. Weighting movements directly discourages ERIS from serving vehicles along the eastbound movement by using a weight below 1.0 to downgrade the importance of serving these vehicles. Incorporating a term from makespan into our objective penalizes schedules that serve vehicles one at a time along the eastbound movement because (as we saw in the motivating example) this leads to a higher schedule makespan. Finally, when vehicles along the eastbound movement require longer to serve (because they are being served one at a time), phantom clusters arriving along the southbound movement experience a higher delay, again discouraging ERIS from selecting schedules that spend relatively more time serving the eastbound movement.

Since there is no asymmetry in the relative importance of serving a vehicle on a given movement for the 1x1 Intersection or the 2x2 Intersection, it is not surprising that weighting strategies that are meant to prioritize one movement over another are not useful. However, it is not obvious that the other two extensions would not offer an improvement either. We conjecture that the intersections are fairly simple and since we are unable to identify an inefficiency for these intersections, ERIS does a reasonably good job optimizing the intersections without the need for any modifications.

6.6 Critical Movements

In the previous section, we argued that for the 2x1 Intersection, it was important to prioritize the movement with one lane over the movement with two lanes. In this section, we introduce the notion of critical movements, which is important background to understand the demands selected for our next experiment, our phantom cluster parameter settings, and the results of our weighting experiments.

Chapter 3 of the Traffic Signal Timing Manual (first edition) defines critical movements as ”[The movements(elements) in] the set of movements that cannot time concurrently and require the most time to serve demand” [36]. This is somewhat of an abstruse definition. In this section, we begin by presenting an approximate definition of critical movements. We present the necessary components to build up to a formal definition for critical movements while in the process explaining how one can calculate the critical movements. We then provide an example calculation. Finally, we highlight minor modifications to the procedure when straight movements and right turn movements do not share lanes.

6.6.1 Approximate Definition of Critical Movements

Roughly, the critical movements of an intersection are the busiest movements that cannot occur simultaneously. The following formal definition reminds the reader what constitutes a movement, how we define “busiest,” and highlights how to determine if movements can occur simultaneously.

6.6.2 Formal Definition of Critical Movements

As we have discussed throughout this document, intersections have distinct movements, for example, three such movements might be vehicles from the west travelling straight, vehicles from the west turning left, and vehicles from the east travelling straight. Some of these movements may occur simultaneously without any issue. For example, vehicles from the west travelling straight and vehicles from the west turning left do not conflict with each other. On the other hand, vehicles from the west turning left and vehicles from the east travelling straight cannot travel simultaneously (as their paths cross and there could be an accident if vehicles from both movements proceed simultaneously). These are conflicting movements.

Similar to the heuristic calculation used by ERIS within its search (please refer to Figure 3.14), one can build sets of mutually conflicting movements. For example, an eight movement intersection with dedicated left turn lanes and shared right/straight lanes has four sets with four conflicting movements. They are:

1. East left, West straight/right, North straight/right, South left
2. East left, West straight/right, North left, South straight/right
3. East straight/right, West left, North straight/right, South left
4. East straight/right, West left, North left, South straight/right

From the Traffic Signal Timing Manual definition above, each of these sets is a “set of movements that cannot time concurrently.” We return to our discussion of these sets in three paragraphs.

Also, observe that no set of 5 mutually conflicting movements exists; if two movements have the same origin (e.g. east left and east straight/right), they cannot conflict. Additionally, any set of three (or fewer) conflicting movements is a subset of at least one of the above sets.

Each movement has an expected demand (per unit of time) as well as a fixed number of lanes. For example, the west straight/right movement may have 800 vehicles per hour and 2 lanes; the west left movement may have 300 vehicles per hour and 1 lane. This information can be used to calculate an expected number of vehicles per lane for each movement. In our example, the west straight/right movement has 400 vehicles per hour per lane. Similarly, the west left movement has 300 vehicles per hour per lane.

One may calculate a "score" for each set of conflicting movements by summing the total vehicles per hour per lane across each movement in the set. The set of conflicting movements with the highest score is our critical movement set and by definition, each movement within the set is a critical movement. From the Traffic Signal Timing Manual definition above, this score represents "the time to serve demand" and the set with the highest score is "the set of movements that cannot time concurrently and require the **most** time to serve demand."

Additionally, we note that our definition assumes that there are no significant differences between the types of vehicles travelling across different movements. However, it may be the case that some movements have a higher percentage of passenger vehicles while other movements have a higher percentage of trucks. Trucks are longer than passenger vehicles and often travel and accelerate slower than passenger vehicles. As such, trucks require more time to pass through an intersection than passenger vehicles.

When one suspects that there are significant differences between the types of vehicles travelling across different movements, when calculating the critical movements, one should scale the importance of each vehicle by a factor relating to their expected time to clear the intersection. One can pick a unit to calibrate and calculate the congestion on each movement according to this unit. For example, one may wish to calculate the passenger-vehicle-equivalents travelling along each movement. Consider the following simplified example:

Suppose that an average passenger vehicle is 5 meters long and an average truck is 20 meters long (20 meters is a model estimate; semi-tractor-trailer trucks tend to be slightly longer than 20 meters but other trucks are shorter). Additionally, assume that at an intersection, vehicles leave a 5 meter gap between them and the preceding vehicle. Finally, assume that vehicles all travel at the same fixed speed. It will require roughly $\frac{20+5}{5+5} = 2.5$ times as much space (and thus time) to serve a truck. Thus, by scaling trucks to each count as 2.5 passenger-vehicle-equivalents, we account for the fact that trucks will require more time to serve. (Passenger vehicles will each count as 1.0 passenger-vehicle-equivalents in our calculation).

The remainder of our discussion of critical movements will assume that the vehicle profiles across different movements are identical and will not apply these vehicle type-specific scaling factors. However, in the appendix, we present an experiment demonstrating that scaling trucks to count as approximately $2.\overline{3}$ passenger-vehicle-equivalents is appropriate to calculate the critical movements.

6.6.3 Example Calculation

Consider the intersection in Figure 6.3 which lists the hourly demand for each movement. (Each of the 8 movements has one lane).

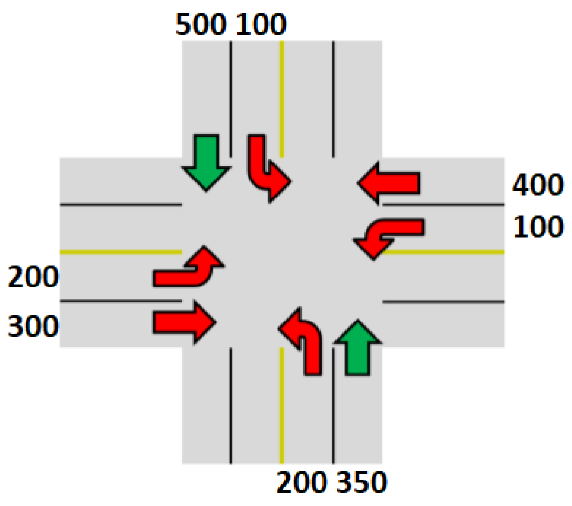


Figure 6.3: Eight-Movement Intersection with Movement-Specific Demands

To calculate the critical movements, we want to find the set of conflicting movements with the highest score (where score, as defined above, is the sum of vehicles per hour per lane for each movement in the set). We will calculate the score for each of the four sets listed above.

First, we calculate the vehicles per lane per hour for each movement. Since all movements have one lane, this step is trivial and simply requires modifying the units for each movement. Next, we calculate the score for each set. The calculations are as follows:

1. East left (100) + West straight/right (300) + North straight/right (500) + South left (200) = 1100
2. East left (100) + West straight/right (300) + North left (100) + South straight/right (350) = 850
3. East straight/right (400) + West left (200) + North straight/right (500) + South left (200) = 1300
4. East straight/right (400) + West left (200) + North left (100) + South straight/right (350) = 1050

The third set has the highest score and thus our critical movements are east straight/right, west left, north straight/right, and south left.

6.6.4 Additional Considerations

In the above example, the straight and right turn movements share the same lane and thus can be calculated together. At many intersections, this will not be the case; straight movements will have one (or more) dedicated lane(s) as will right turn movements. When this is the case, we split the straight and right turn movements up into separate sets. For example, if the west movement has dedicated straight lanes and dedicated right turn lanes (while all other straight and right movements are shared), our list of sets becomes:

1. East left, **West straight**, North straight/right, South left
2. East left, **West right**, North straight/right, South left
3. East left, **West straight**, North left, South straight/right
4. East left, **West right**, North left, South straight/right
5. East straight/right, West left, North straight/right, South left
6. East straight/right, West left, North left, South straight/right

If all four directions have separate straight and right turn lanes, we instead must compare 16 sets. (It should be noted that while there are more "elegant" recursive algorithms to calculate the critical movements, a brute force algorithm comparing 16 sums is simple to explain, simple to implement, and the runtime is negligible for our use-case).

It is also necessary to consider the case when a lane serves both movements but dedicated lanes also exist. For example, an intersection may have one dedicated straight lane and one lane that is shared by both the right turn and straight movements. When this is the case, we calculate whether vehicles travelling straight using the shared lane would make relative flows more equal. If this is the case, then we utilize a shared movement in our set generation. Otherwise, we split the movements into separate sets.

For example, suppose there are 300 vehicles travelling right and 500 vehicles travelling straight. If the right turn lane were instead a dedicated right turn lane, the split would be 500 for the straight lane and 300 for the right turn lane. However, since the lane is shared, 100 vehicles travelling straight could use the shared lane to bring the split to 400:400 (making the split equal). Thus, we would use a shared movement in our sets.

On the other hand, if the original split were 500 vehicles travelling right and 300 vehicles travelling straight, vehicles travelling straight shifting to the shared lane would make the split more unequal. As such, we assume that straight vehicles will use the dedicated lane only and we keep the movements separate in our set generation.

There are other considerations for shared left turn lanes that are beyond the scope of this thesis. We refer the reader to Chapter 16 of the Highway Capacity Manual for further discussion [44].

6.7 Experimental Analysis on Eight-Movement Intersection

In our first set of experiments, we demonstrated that all three methods had the potential to significantly reduce delay. However, we only observed this improvement for the 2x1 Intersection where it was important to efficiently serve the movement with two lanes. In our next set of experiments, we examine another intersection with some movements that may be more important to serve efficiently than other movements. Specifically, we examine the intersection presented earlier in Figure 6.3, which we refer to as the "Eight-Movement Intersection." As discussed above, the Eight-Movement Intersection has four critical movements (east straight/right, west left, north straight/right, and south left). We examine whether our proposed extensions reduce the delay for an intersection with critical movements.

For each method, we run a parameter sweep to explore a wide range of parameter settings to estimate the ideal parameters. We then run an experiment to validate these ideal settings to determine whether the results from the parameter sweeps are statistically significant. We observe a statistically significant improvement from one of the extensions (weighting) and run an additional follow-up experiment to further validate our weighting results.

6.7.1 Phantom Clusters - Parameter Sweep

Experimental Setup

We begin by running ERIS to include phantom clusters when scheduling the Eight-Movement Intersection. We examine a variety of parameter settings. For our first twelve combinations, we generate 1, 2, or 3 phantom clusters along all movements and assign them a weight of 1.0, $\frac{1}{2}$, $\frac{1}{3}$, or 1.25. We also consider only generating phantom clusters for the critical movements. We generate 1, 2, or 3 phantom clusters along the critical movements with the same importance as real clusters and generate 1 phantom cluster along each critical movement that is half as important as real clusters (all other parameters are fixed and detailed in 6.3.1).

For each parameter setting, we run 5 half-hour simulation trials on the above intersection.

Results

Average vehicle delay is presented in Table 6.4.

Method	Delay (s)	vs Baseline
Baseline (ERIS)	79.3	0.0%
1 Phantom Cluster	80.1	1.0%
2 Phantom Clusters	114.8	44.8%
3 Phantom Clusters	136.3	71.8%
1 Phantom Cluster, half as important	80.2	1.1%
2 Phantom Clusters, half as important	101.6	28.0%
3 Phantom Clusters, half as important	108.9	37.3%
1 Phantom Cluster, 1/3 as important	80.0	0.8%
2 Phantom Clusters, 1/3 as important	82.2	3.7%
3 Phantom Clusters, 1/3 as important	86.7	9.3%
1 Phantom Cluster, 1.25 as important	81.5	2.8%
2 Phantom Clusters, 1.25 as important	122.0	53.7%
3 Phantom Clusters, 1.25 as important	152.2	91.9%
1 Phantom Cluster, only crit movements	77.7	-2.0%
2 Phantom Clusters, only crit movements	77.5	-2.3%
3 Phantom Clusters, only crit movements	87.3	10.0%
1 Phantom Cluster, only crit movements, half as important	76.9	-3.0%

Table 6.4: Average Vehicle Delay (s) for Phantom Clusters Parameter Sweep

We observe that most of these settings lead to an increase in delay when compared to the baseline. We notice a small improvement, of 2% to 3%, for three (of the four) settings when phantom clusters are only generated along the critical movements. Otherwise, we do not observe an improvement due to phantom clusters. We defer a discussion of this result until the validation experiment presented later in this section. In the validation experiment, we run additional trials for the best performing setting (1 Phantom Cluster, only critical movements, half as important).

6.7.2 Penalizing Higher Makespans - Parameter Sweep

Experimental Setup

We next examine the impact of penalizing higher makespans when scheduling the Eight-Movement Intersection. We examine a variety of parameter settings (relative weightings for combining delay and makespan), which are detailed in our results. As before, for each parameter setting, we run 5 half-hour simulation trials on the above intersection.

Results

Average vehicle delay is presented in Table 6.5. (Please note that we use different random seeds to generate our demand file for all experiments; thus, the baseline delay for this parameter sweep is different than for the previous parameter sweep).

Makespan Weight	Delay (s)	vs Baseline
0.00 (Baseline, Ignore Makespan)	77.6	0.0%
0.05	75.5	-2.8%
0.10	76.0	-2.1%
0.15	74.9	-3.5%
0.20	74.1	-4.5%
0.25	77.0	-0.9%
0.30	71.1	-8.4%
0.35	69.6	-10.4%
0.40	69.9	-10.0%
0.45	70.4	-9.4%
0.50	69.6	-10.3%
0.55	70.2	-9.6%
0.60	69.2	-10.9%
0.65	68.1	-12.2%
0.70	68.8	-11.4%
0.75	68.3	-12.1%
0.80	68.6	-11.6%
0.85	69.9	-9.9%
0.90	69.8	-10.1%
0.95	73.4	-5.4%
1.00 (Makespan Only)	71.4	-8.1%

Table 6.5: Average Vehicle Delay (s) for Makespan Parameter Sweep

For all of the weights between 0.35 and 0.9, we observe a reduction in delay of approximately 10% when compared to the baseline. We defer a discussion of this result until the validation experiment presented later in this section. In the validation experiment, we run additional trials for the best performing setting (weighting makespan at 0.65, weighting delay at 0.35).

6.7.3 Weighting Approaches - Parameter Sweep I

Experimental Setup

We now examine the impact of weighting approaches when scheduling the Eight-Movement Intersection. We examine all $(2^8 - 1)$ possible combinations of weighted/unweighted movements. We use a weight of 0.3 to downgrade selected movements. Because we are examining a larger set of candidates, for each candidate, we run only 3 half-hour simulation trials on the above intersection.

Results

We assign an integer (0 - 254) to each of the 255 weighting strategies. The bits set to one in the binary representation of this integer indicates which movements should be downgraded. For example, if the right-most bit is set to one, we downgrade the north straight movement. The full bit to movement mapping is presented in Figure 6.4.

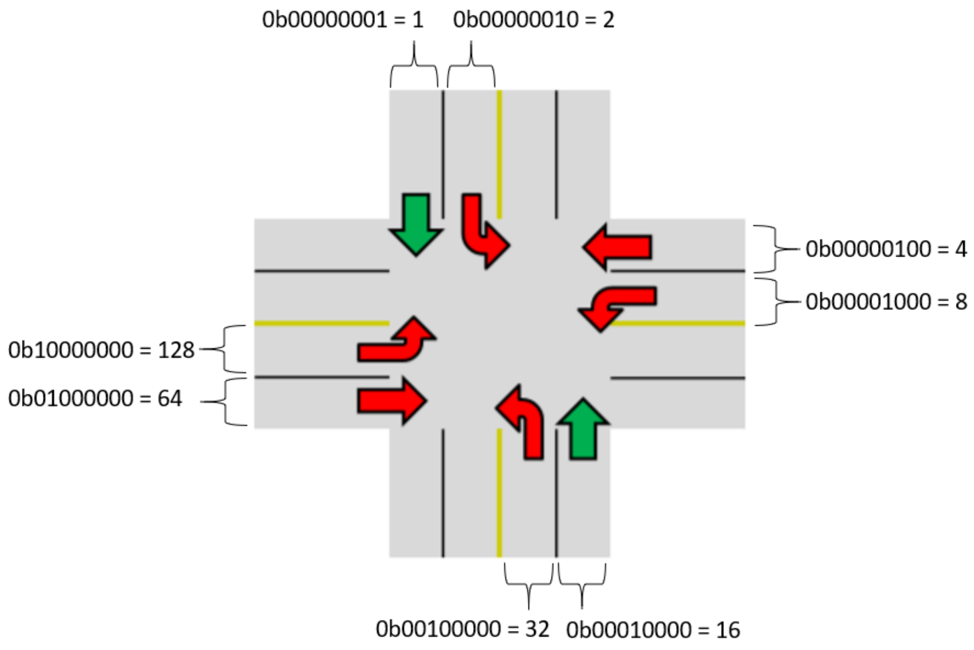


Figure 6.4: Bit to Movement Mapping for Weighting Extension

For a given strategy, one converts the integer to binary and then finds the matching bits in the mapping to downgrade. For example, Strategy #0 = 0b00000000 does not downgrade any

ID	Binary Representation	Critical Movements Prioritized	Non-Critical Movements Prioritized	Delay (s)	vs Baseline
90	01011010	4	0	64.1	-16.5%
82	01010010	4	1	64.8	-15.6%
80	01010000	4	2	66.8	-13.0%
88	01011000	4	1	67.0	-12.8%
26	00011010	4	1	70.4	-8.4%
10	00001010	4	2	70.5	-8.3%
18	00010010	4	2	71.4	-7.0%
24	00011000	4	2	72.1	-6.1%
2	00000010	4	3	73.3	-4.6%
8	00001000	4	3	73.6	-4.2%
66	01000010	4	2	74.4	-3.2%
16	00010000	4	3	74.5	-3.0%
74	01001010	4	1	74.7	-2.8%
0	00000000	4	4	76.8	0.0%
72	01001000	4	2	77.1	0.4%
64	01000000	4	3	80.6	4.9%
222	11011110	2	0	80.9	5.4%
1	00000001	3	4	81.8	6.6%
4	00000100	3	4	82.2	7.0%
214	11010110	2	1	82.8	7.9%

Table 6.6: Average Vehicle Delay (s) for Weighting Parameter Sweep I - Top 20 Strategies

movements, which is equivalent to running baseline ERIS. Strategy #90 = 0b01011010 downgrades the north left, east left, south straight/right, and west straight/right movements, which are the four movements that are not critical movements.

We present the 20 best performing strategies (those with the lowest delay) in Table 6.6. For each strategy, we also include the count of how many critical movements were prioritized (not downgraded) and how many non-critical movements were prioritized (not downgraded).

While we caution reading too much into these results given the small sample size, we observe that the best strategy is Strategy #90, which downgrades the four non-critical movements while keeping the critical movements unweighted. Additionally, of the four next best strategies, three of these downgrade three of the non-critical movements while keeping the critical movements unweighted. Furthermore, all 16 of the best strategies keep the critical movements unweighted (one of these is the baseline). We defer further discussion of this result until the validation experiment presented later in this section. In the validation experiment, we run additional trials for the best performing setting (Strategy #90).

6.7.4 Weighting Approaches - Parameter Sweep II

Experimental Setup

The previous experiment demonstrated that Strategy #90, which downgraded the non-critical movements, outperformed other strategies. In that experiment, we utilized a weight of 0.3 for the downgraded movements. In this experiment, we examine a variety of potential weights for the downgraded movements to determine if any particular weight leads to a lower delay for Strategy #90. For each parameter setting, we run 5 half-hour simulation trials on the above intersection.

Results

Average vehicle delay is presented in Table 6.7.

Weighting Factor	Delay (s)	vs Baseline
1.00 (Baseline)	79.7	0.0%
0.96	78.0	-2.2%
0.92	73.6	-7.7%
0.88	73.9	-7.3%
0.84	78.2	-1.9%
0.80	74.3	-6.9%
0.76	72.8	-8.7%
0.72	69.4	-13.0%
0.68	67.2	-15.7%
0.64	68.8	-13.7%
0.60	65.4	-17.9%
0.56	66.8	-16.2%
0.52	67.4	-15.5%
0.48	65.3	-18.1%
0.44	64.2	-19.5%
0.40	65.1	-18.4%
0.36	64.5	-19.1%
0.32	63.9	-19.8%
0.28	65.4	-17.9%
0.24	63.8	-20.0%
0.20	64.4	-19.3%
0.16	63.3	-20.6%
0.12	64.9	-18.6%
0.08	65.4	-18.0%
0.04	64.6	-19.0%

Table 6.7: Average Vehicle Delay (s) for Weighting Parameter Sweep II

We observe that as the weight for the non-critical movements falls from 1.00 (the base case of not weighting) to 0.48, delay generally decreases (with some increases along the way, likely due to noise). The percentage improvement is largest, and relatively consistent, between 0.48 and 0.04 (inclusive). When compared to the baseline, these weights reduce delay between 18% and 21%. For this experiment, we cannot declare an ideal weight; all weights in the range [0.04,0.48] perform fairly well. In the validation experiment presented below, we run additionally trials for Strategy #90 while continuing to use a weight of 0.3 since it falls within this well-performing range.

6.7.5 Validating and Combining the Three Extensions

Experimental Setup

Throughout this section, we have presented parameter sweeps of the three methods on our Eight-Movement Intersection. We found that in the parameter sweeps, weighting movements led to the largest decrease in delay of 20.6%. Including a term for makespan decreased delay by as much as 12.2% and generating phantom clusters for the critical movements reduced delay by as much 3.0%.

However, it should be noted that these parameters sweeps are inconclusive. Each trial was only half an hour long and we only conducted 3 - 5 trials for each parameter. Additionally, we explored a variety of parameter settings so selecting the best may be improperly attributing the improvement to noise.

Thus, in this section, we select the best parameter setting for each of these methods and run them on 10 one-hour long experiments to attempt to validate the results of our earlier parameter sweeps. Specifically, we consider the following parameter selections for our methods:

1. Generating 1 phantom cluster only along critical movements, with a weight half as important as real clusters
2. Weighting makespan at 0.65 and weighting delay at 0.35
3. Downgrading clusters along non-critical movements (Strategy #90); their importance is multiplied by 0.3

Additionally, we also examine combining these methods which each other to determine whether they offer complementary benefits to any extent or if only one method is necessary for the delay reduction.

Results & Discussion

Average vehicle delay and standard errors are presented in Table 6.8.

First, we note that the improvement from these experiments is lower than observed during the parameter sweeps. The largest reduction in delay from this experiment is 5.1% while the largest reduction was 20.6% in the parameter sweeps. We briefly discuss two factors that may contribute to the drop. First, it is possible that this drop is solely due to noise. Trials in this experiment are an hour long (compared to 30 minutes for the parameter sweeps), so we measure the delay of more vehicles in each trial. Additionally, we run more trials for this experiment than for the parameter sweeps. Every delay value in Table 6.8 considers approximately 8 times as many vehicles as the first weighting parameter sweep and approximately 5 times as many vehicles as the other parameter sweeps.

Strategy	Delay (s)	vs Baseline
Baseline ERIS	66.9 $\pm$ 1.3	0.0%
Clusters Only	66.1 $\pm$ 1.1	-1.2%
Objective Only	65.8 $\pm$ 1.2	-1.7%
Weight Only	63.7 $\pm$ 0.9	-4.8%
Clusters & Objective	70.4 $\pm$ 1.4	5.2%
Clusters & Weight	63.5 $\pm$ 0.8	-5.1%
Objective & Weight	64.9 $\pm$ 0.8	-3.0%
Clusters, Objective, & Weight	66.7 $\pm$ 1.0	-0.3%

Table 6.8: Average Vehicle Delay (s) for Validation Experiment

Second, recall that in the parameter sweeps, we focused on selecting the parameter settings with the largest improvements. Since we are evaluating many combinations of parameters, due to chance, we would expect some strategies to appear to offer an improvement larger than their true average improvement. For example, only three of the sixteen Phantom Clusters settings considered in Section 6.7.1 offered an improvement when compared against the baseline. It is possible that these three strategies overestimated the benefit of the extension.

Returning to the data presented in Table 6.8, we are interested in determining which of the strategy combinations offer an improvement over baseline ERIS. We compare the baseline to each of the seven proposed extensions using a two-sided paired t-test with a 95% confidence level, and as in Section 3.6.6, we apply the Holm-Bonferroni method to maintain an overall error rate of 5% [30]. Two methods - Weight Only and Clusters & Weight are statistically significantly better than baseline ERIS. These methods are not statistically significantly different from each other.

Our primary take-away from the validation experiment is that the Weight Only strategy that downgrades the importance of the non-critical movements decreases delay (compared to the baseline) by 4.8% and this improvement is statistically significant.

When compared against the baseline, Clusters Only and Objective Only offer small improvements of 1.2% and 1.7%, respectively. These improvements are not statistically significant and they are smaller than the improvement from Weight Only. Thus, while it may be the case that these methods offer some improvement against the baseline, independently, their benefit is lower than the Weight Only strategy.

Since Weight Only and Clusters & Weight are not statistically significantly different from each other, we suspect that Clusters & Weight slightly outperforming (by approximately 0.3%) Weight Only is due to noise. We suspect that the majority of the improvement from the Clusters & Weight strategy comes from the Weighting extension and there is little, if any, improvement due to the marginal incorporation of phantom clusters. The other two combinations involving weighting (Objective & Weight and Clusters, Objective, & Weight) offer a smaller improvement than Weight Only. These results suggest that incorporating the other two extensions along with weighting likely does not offer any additional utility than solely applying the weighting strategy that downgrades the non-critical movements.

6.7.6 Weighting Approaches - Validation Experiment

Experimental Setup

Our previous experiment demonstrated that Strategy #90, which downgrades the non-critical movements, offers a statistically significant improvement over the baseline of not weighting movements. In this section, we run one additional validation experiment investigating whether this weighting strategy (Strategy #90) is actually the best strategy or just appears that way from our low sample size from the parameter sweep presented in Section 6.7.3. We run 10 one-hour simulations for the top 20 weighting strategies from Section 6.7.3.

Results

Average vehicle delay is presented in Table 6.9. We again notice that the best strategy is Strategy #90 where the non-critical movements are downgraded. The two next best strategies downgrade 3 of the 4 non-critical movements (and none of the non-critical movements).

ID	Binary Representation	Critical Movements Prioritized	Non-Critical Movements Prioritized	Delay (s)	vs Baseline
90	01011010	4	0	66.2	-6.4%
82	01010010	4	1	67.2	-5.0%
88	01011000	4	1	67.2	-5.0%
64	01000000	4	3	67.5	-4.5%
66	01000010	4	2	67.5	-4.5%
80	01010000	4	2	68.1	-3.7%
72	01001000	4	2	68.2	-3.6%
26	00011010	4	1	68.3	-3.4%
2	00000010	4	3	69.0	-2.4%
8	00001000	4	3	69.2	-2.2%
16	00010000	4	3	69.4	-1.9%
10	00001010	4	2	69.5	-1.8%
24	00011000	4	2	69.5	-1.7%
74	01001010	4	1	69.8	-1.4%
18	00010010	4	2	70.3	-0.6%
0	00000000	4	4	70.7	0.0%
1	00000001	3	4	87.5	23.7%
4	00000100	3	4	89.8	27.0%
222	11011110	2	0	95.7	35.3%
214	11010110	2	1	97.1	37.2%

Table 6.9: Average Vehicle Delay (s) for Weighting Validation Experiment

6.7.7 Summary

In this section, we examined the three proposed extensions on an Eight-Movement Intersection with well-defined critical movements. After running parameter sweeps for the three methods, we evaluated the ideal parameter settings in a validation experiment. We found a statistically significant reduction in delay for weighting movements but not for the other two extensions. Specifically, downgrading the non-critical movements using a factor of 0.3 led to a delay reduction of approximately 5%. Incorporating the other extensions did not appear to offer any additional utility.

We then re-examined weighting movements to confirm that downgrading the non-critical movements appears to be the best pattern for this intersection with this specific demand profile.

Our results support our earlier hypothesis that when some movements are more important than other movements, it is valuable to prioritize the more important movements. In Chapter 7, we present a formal model further supporting this hypothesis and examine whether prioritizing the critical movements by downgrading the non-critical movements is always the best strategy.

6.8 Conclusion

In this chapter, we proposed three extensions to improve our core ERIS scheduler from Chapter 3. Specifically, we examined generating phantom vehicles clusters, including a term for makespan in our objective function, and weighting movements. We evaluated these methods on four separate intersections. For two of the intersections, the 2x1 Intersection and for the Eight-Movement Intersection, we noticed that the proposed methods reduced delay. This improvement was largest and statistically significant for the weighting movements extension. For the 2x1 Intersection, downgrading the importance of clusters along the movement with two lanes was the ideal weighting strategy. For the Eight-Movement Intersection, downgrading the importance of clusters along the non-critical movements was the ideal weighting strategy.

For the 2x1 Intersection, we also observed improvement from the other two extensions (generating phantom vehicles clusters and including a term for makespan in our objective function). However, for these extensions, we did not observe a statistically significant improvement in the Eight-Movement Intersection validation experiment. We do not rule out that these extensions may hold promise that we did not uncover in this analysis. Further exploration of these extensions across other networks and demand profiles may be a worthwhile future pursuit. (For example, intersections with short approaches and no advance detection may greatly benefit from phantom clusters since these intersections currently utilize limited cluster information to inform their decisions). However, we do not consider these extensions further in this document.

Rather, in Chapter 7, we explore the idea of weighting in additional detail. We present a deterministic queuing model in which we examine why prioritizing critical movements can be beneficial for long-term delay. We examine weighting strategies on a simulated model of a real network to demonstrate that our results apply to more complicated settings. We carefully examine the busiest intersection within this network and try to characterize which weighting strategies seem to be beneficial. Specifically, we tie together the results of the 2x1 Intersection experiment, which suggested weighting based on the number of lanes with the results of the Eight-Movement

Intersection experiment, which suggested downgrading the non-critical movements.

6.9 Appendix

In this appendix, we present a brief experiment demonstrating that one should scale the importance of different vehicle types when calculating the critical movements (as discussed in Section 6.6.4) and present the full results for the experiment from Section 6.7.3.

6.9.1 Critical Movement Experiment for Different Vehicle Types

As discussed in Section 6.6.4, when calculating the critical movements, it is important to scale vehicles by a factor relating to the amount of time required to serve them. We present an experiment where we vary the relative amount of traffic along different movements and use the observed critical movements to estimate an appropriate scaling factor for trucks.

Experimental Setup

We consider a single intersection with four straight movements (north, south, east, and west). Only trucks arrive along the north and east movements. Only passenger vehicles arrive along the south and west movements. In this experiment, passenger vehicles are assumed to be 5 meters long while trucks are assumed to be 20 meters long. Additionally, passenger vehicles accelerate three times faster than trucks.

In our evaluation, we will consider a variety of average arrival rates across different movements. The north and east movements will have one average arrival rate (for trucks) and the south and west movements will have a different average arrival rate (for passenger vehicles).

For each demand profile, we will examine whether it is ideal to weight all movements equally (providing a weight of 1.0 to all movements), prioritize passenger vehicles (by assigning a weight of 0.5 to trucks), or prioritize trucks (by assigning a weight of 2.0 to trucks). Given the discussion throughout the chapter that it is important to prioritize critical movements, the best performing weighting strategy will indicate which movements are critical. After obtaining which movements are critical given the demand profile, we will be able to estimate the passenger-vehicle-equivalent scaling factor to use for trucks.

We run ten 1-hour trials for each combination of demand profile and weighting strategy.

Experimental Results

In Table 6.10, we present the average delay for each weighting strategy for five demand profiles. Additionally, we assign each strategy a key (C/X/T) and also report which strategies are best for each demand profile.

Discussion

First, we note that at the two lowest levels of demand (500/600 passenger vehicles, 300 trucks), more passenger vehicles than trucks use the intersection. However, it is best to prioritize the

Average Passenger Vehs per Movement (South & West)	500	600	700	800	850
Average Trucks per Movement (North & East)	300	300	300	300	250
Prioritize Passenger Vehicles (C)	50.1	47.3	40.8	46.5	59.2
No Prioritization/Baseline ERIS (X)	38.5	41.4	38.4	46.6	61.7
Prioritize Trucks (T)	34.3	36.2	39.2	58.9	80.0
Best Method(s)	T	T	X	C, X	C

Table 6.10: Average Vehicle Delay (s) for Different Weighting Strategies

trucks at this level of demand. This confirms the earlier claim that we should not treat all vehicles equally when calculating the critical movements.

Given our earlier discussion that prioritizing the critical movements can reduce delay, we claim that at these levels of demand, the north and east movements (which serve trucks) are the critical movements. We can then use this claim to bound the appropriate scaling factor one should use when estimating the critical movements from the number of vehicles. We must select a scaling factor to apply to the truck demand so that at these levels of demand, the passenger-vehicle-equivalents from the north and east movements are higher than the passenger-vehicle-equivalents from the south and west movements. For the lowest demand, we find that selecting a scaling factor greater than $\frac{500}{300} = 1.\overline{6}$ will make our calculation suggest that the north and east movements are critical. Similarly, for the second lowest demand, we find that selecting a scaling factor greater than $\frac{600}{300} = 2$ will make our calculation suggest that the north and east movements are critical.

At the highest demand (850 passenger vehicles, 250 trucks), prioritizing the passenger vehicles is best. We claim that at this level of demand, the south and west movements (which serve passenger vehicles) are the critical movements. Again, we can then use this claim to bound the appropriate scaling factor one should use when estimating the critical movements from the number of vehicles. We must select a scaling factor to apply to the truck demand so that at this level of demand, the passenger-vehicle-equivalents from the south and west movements are higher than the passenger-vehicle-equivalents from the north and east movements. At this demand, we find that selecting a scaling factor less than than $\frac{850}{250} = 3.4$ will make our calculation suggest that the south and west movements are critical.

We can combine these results to bound the appropriate scaling factor for trucks to be between 2.0 and 3.4 passenger-vehicle-equivalents.

At the middle demand (700 passenger vehicles, 300 trucks), we see that no prioritization is the best strategy. This leads us to hypothesize that within the aforementioned bounds, the ideal scaling factor might lie pretty close to $\frac{700}{300} = 2.\overline{3}$ (Note, however, that this is an approximation specific to the parameter selections of our model).

We now argue that had we known to use a scaling factor of $2.\overline{3}$ to calculate the critical movements, we would have optimized delay regardless of the selected demand. Observe that for the lowest demand, when applying a scaling factor of $2.\overline{3}$ to trucks, we obtain 700 ($300 * 2.\overline{3}$) passenger-vehicle-equivalents on the north and east movements. This is higher than the 500 passenger vehicles on the south and west movements; this suggests prioritizing the north and east movements, which we observed was the best strategy. For the highest demand, when applying a scaling factor of $2.\overline{3}$ to trucks, we obtain 583 ($250 * 2.\overline{3}$) passenger-vehicle-equivalents on the north and east movements. This is lower than the 850 passenger vehicles on the south and west movements; this suggests prioritizing the south and west movements, which we observed was the best strategy. Finally, for the middle demand, when applying a scaling factor of $2.\overline{3}$ to trucks, we obtain 700 ($300 * 2.\overline{3}$) passenger-vehicle-equivalents on the north and east movements. This is equal to the 700 passenger vehicles on the south and west movements, which suggests that prioritization of critical movements may not be useful at these demands.

Note that in this experiment, we presented the extreme case where we observed distinct types of vehicles along different movements. In the real-world, we would not expect to observe these extreme differences, but we may observe minor differences. The different vehicle compositions may influence which movements are critical and whether it is better to prioritize a specific movement or to apply baseline ERIS without any weighting. Perhaps most importantly, if one were to apply weighting dynamically using on-line calculations of the critical movements, one should remember to apply appropriate scaling factors if vehicle types differ across movements.

6.9.2 Full Results for Experiment from Section 6.7.3

ID	Binary Representation	Critical Movements Prioritized	Non-Critical Movements Prioritized	Delay (s)	vs Baseline
90	01011010	4	0	64.1	-16.5%
82	01010010	4	1	64.8	-15.6%
80	01010000	4	2	66.8	-13.0%
88	01011000	4	1	67.0	-12.8%
26	00011010	4	1	70.4	-8.4%
10	00001010	4	2	70.5	-8.3%
18	00010010	4	2	71.4	-7.0%
24	00011000	4	2	72.1	-6.1%
2	00000010	4	3	73.3	-4.6%
8	00001000	4	3	73.6	-4.2%
66	01000010	4	2	74.4	-3.2%
16	00010000	4	3	74.5	-3.0%
74	01001010	4	1	74.7	-2.8%
0	00000000	4	4	76.8	0.0%
72	01001000	4	2	77.1	0.4%
64	01000000	4	3	80.6	4.9%
222	11011110	2	0	80.9	5.4%
1	00000001	3	4	81.8	6.6%
4	00000100	3	4	82.2	7.0%
214	11010110	2	1	82.8	7.9%
220	11011100	2	1	83.9	9.2%
6	00000110	3	3	83.9	9.2%
9	00001001	3	3	84.2	9.6%

94	01011110	3	0	84.3	9.8%
75	01001011	3	1	84.4	9.9%
14	00001110	3	2	84.7	10.3%
76	01001100	3	2	84.8	10.4%
65	01000001	3	3	85.0	10.7%
105	01101001	2	2	85.1	10.9%
70	01000110	3	2	85.4	11.1%
3	00000011	3	3	85.4	11.2%
20	00010100	3	3	85.7	11.5%
78	01001110	3	1	85.9	11.9%
196	11000100	2	3	86.2	12.2%
86	01010110	3	1	86.3	12.4%
92	01011100	3	1	86.5	12.6%
12	00001100	3	3	86.5	12.7%
205	11001101	1	2	86.6	12.7%
67	01000011	3	2	86.6	12.8%
73	01001001	3	2	86.7	12.8%
84	01010100	3	2	87.0	13.3%
68	01000100	3	3	87.0	13.3%
22	00010110	3	2	87.3	13.7%
212	11010100	2	2	87.4	13.8%
28	00011100	3	2	87.7	14.2%
11	00001011	3	2	88.0	14.6%
43	00101011	2	2	88.4	15.1%
30	00011110	3	1	88.7	15.4%
198	11000110	2	2	89.3	16.3%
229	11100101	0	3	89.4	16.3%
207	11001111	1	1	89.5	16.6%
83	01010011	3	1	89.6	16.6%
231	11100111	0	2	90.3	17.6%
81	01010001	3	2	90.7	18.0%
97	01100001	2	3	91.0	18.5%
204	11001100	2	2	91.2	18.8%
121	01111001	2	1	91.4	19.0%
91	01011011	3	0	91.4	19.0%
89	01011001	3	1	91.7	19.4%
199	11000111	1	2	92.0	19.8%
253	11111101	0	1	92.1	19.9%
113	01110001	2	2	92.3	20.2%
115	01110011	2	1	93.2	21.3%
183	10110111	0	2	93.6	21.9%
127	01111111	1	0	94.1	22.6%
55	00110111	1	2	94.2	22.7%
19	00010011	3	2	94.7	23.3%

239	11101111	0	1	95.2	24.0%
63	00111111	1	1	95.3	24.1%
151	10010111	1	2	95.4	24.2%
101	01100101	1	3	96.1	25.2%
99	01100011	2	2	96.3	25.4%
158	10011110	2	1	96.6	25.8%
25	00011001	3	2	97.2	26.5%
156	10011100	2	2	97.3	26.7%
167	10100111	0	3	97.6	27.1%
223	11011111	1	0	98.3	27.9%
107	01101011	2	1	98.3	28.0%
17	00010001	3	3	98.7	28.5%
142	10001110	2	2	98.7	28.5%
109	01101101	1	2	99.0	28.9%
123	01111011	2	0	99.4	29.4%
191	10111111	0	1	99.4	29.4%
247	11110111	0	1	99.7	29.8%
31	00011111	2	1	99.8	30.0%
35	00100011	2	3	100.2	30.5%
79	01001111	2	1	100.3	30.6%
33	00100001	2	4	100.4	30.8%
27	00011011	3	1	100.4	30.8%
53	00110101	1	3	100.5	30.9%
159	10011111	1	1	100.7	31.2%
77	01001101	2	2	100.9	31.4%
119	01110111	1	1	101.0	31.5%
237	11101101	0	2	101.3	31.9%
173	10101101	0	3	101.9	32.6%
245	11110101	0	2	102.0	32.8%
189	10111101	0	2	102.2	33.1%
165	10100101	0	4	102.7	33.7%
111	01101111	1	1	103.4	34.7%
175	10101111	0	2	103.7	35.1%
150	10010110	2	2	104.2	35.7%
206	11001110	2	1	104.4	35.9%
41	00101001	2	3	104.8	36.5%
149	10010101	1	3	104.8	36.5%
197	11000101	1	3	104.9	36.6%
157	10011101	1	2	105.2	37.0%
216	11011000	3	1	105.6	37.6%
29	00011101	2	2	105.9	37.9%
49	00110001	2	3	106.1	38.2%
181	10110101	0	3	106.2	38.3%
21	00010101	2	3	106.4	38.6%

148	10010100	2	3	106.6	38.8%
103	01100111	1	2	107.0	39.3%
218	11011010	3	0	107.7	40.2%
200	11001000	3	2	108.0	40.6%
202	11001010	3	1	108.0	40.6%
61	00111101	1	2	108.1	40.7%
210	11010010	3	1	108.1	40.8%
23	00010111	2	2	108.1	40.8%
215	11010111	1	1	108.2	40.9%
125	01111101	1	1	109.0	42.0%
48	00110000	3	3	109.4	42.4%
69	01000101	2	3	109.5	42.6%
51	00110011	2	2	109.7	42.8%
241	11110001	1	2	109.8	43.0%
192	11000000	3	3	110.0	43.2%
15	00001111	2	2	110.3	43.7%
208	11010000	3	2	110.5	43.9%
194	11000010	3	2	110.6	44.0%
71	01000111	2	2	111.2	44.8%
112	01110000	3	2	111.3	44.9%
95	01011111	2	0	111.3	45.0%
213	11010101	1	2	111.4	45.0%
57	00111001	2	2	111.5	45.2%
47	00101111	1	2	111.9	45.6%
134	10000110	2	3	112.2	46.1%
7	00000111	2	3	112.3	46.3%
58	00111010	3	1	112.6	46.6%
225	11100001	1	3	112.6	46.7%
128	10000000	3	4	112.7	46.7%
114	01110010	3	1	112.8	46.8%
45	00101101	1	3	112.8	46.9%
39	00100111	1	3	113.0	47.1%
136	10001000	3	3	113.0	47.2%
135	10000111	1	3	113.1	47.2%
143	10001111	1	2	113.5	47.8%
211	11010011	2	1	113.5	47.8%
37	00100101	1	4	113.6	47.9%
244	11110100	1	2	113.7	48.0%
56	00111000	3	2	113.9	48.3%
141	10001101	1	3	114.0	48.4%
120	01111000	3	1	114.1	48.5%
209	11010001	2	2	114.3	48.8%
59	00111011	2	1	114.6	49.2%
117	01110101	1	2	114.6	49.2%

122	01111010	3	0	114.9	49.6%
87	01010111	2	1	115.9	50.9%
217	11011001	2	1	116.0	51.1%
50	00110010	3	2	116.6	51.8%
227	11100011	1	2	116.6	51.8%
243	11110011	1	1	116.7	52.0%
221	11011101	1	1	117.1	52.5%
132	10000100	2	4	117.3	52.7%
32	00100000	3	4	117.5	53.0%
13	00001101	2	3	117.5	53.0%
161	10100001	1	4	118.2	53.9%
146	10010010	3	2	118.8	54.7%
5	00000101	2	4	118.9	54.8%
140	10001100	2	3	119.1	55.0%
93	01011101	2	1	119.2	55.2%
85	01010101	2	2	120.4	56.7%
144	10010000	3	3	121.3	57.9%
116	01110100	2	2	121.3	58.0%
195	11000011	2	2	121.9	58.8%
133	10000101	1	4	122.0	58.9%
251	11111011	1	0	122.3	59.2%
130	10000010	3	3	122.4	59.4%
219	11011011	2	0	122.8	59.8%
40	00101000	3	3	123.0	60.1%
154	10011010	3	1	123.5	60.8%
252	11111100	1	1	123.6	60.9%
124	01111100	2	1	123.7	61.0%
249	11111001	1	1	124.1	61.6%
104	01101000	3	2	124.4	62.0%
163	10100011	1	3	124.7	62.4%
201	11001001	2	2	125.1	62.9%
34	00100010	3	3	125.1	62.9%
188	10111100	1	2	125.6	63.6%
129	10000001	2	4	126.1	64.1%
203	11001011	2	1	126.4	64.5%
42	00101010	3	2	126.6	64.8%
177	10110001	1	3	126.8	65.0%
131	10000011	2	3	128.7	67.6%
145	10010001	2	3	128.7	67.6%
152	10011000	3	2	128.8	67.7%
126	01111110	2	0	128.8	67.8%
193	11000001	2	3	129.1	68.1%
254	11111110	1	0	129.4	68.4%
233	11101001	1	2	130.2	69.6%

118	01110110	2	1	130.3	69.7%
60	00111100	2	2	131.0	70.5%
96	01100000	3	3	131.2	70.8%
246	11110110	1	1	131.2	70.8%
180	10110100	1	3	131.4	71.0%
179	10110011	1	2	132.1	72.0%
164	10100100	1	4	132.1	72.0%
235	11101011	1	1	132.2	72.1%
228	11100100	1	3	132.2	72.2%
190	10111110	1	1	132.5	72.6%
138	10001010	3	2	132.7	72.8%
155	10011011	2	1	132.8	72.9%
137	10001001	2	3	133.8	74.3%
52	00110100	2	3	134.0	74.5%
182	10110110	1	2	134.1	74.6%
153	10011001	2	2	134.3	74.9%
169	10101001	1	3	134.5	75.1%
187	10111011	1	1	136.2	77.4%
147	10010011	2	2	136.3	77.5%
62	00111110	2	1	136.6	77.9%
171	10101011	1	2	137.1	78.6%
36	00100100	2	4	137.3	78.7%
44	00101100	2	3	137.6	79.2%
185	10111001	1	2	137.9	79.6%
54	00110110	2	2	138.3	80.0%
236	11101100	1	2	139.1	81.1%
172	10101100	1	3	139.2	81.2%
238	11101110	1	1	141.1	83.7%
108	01101100	2	2	141.5	84.3%
100	01100100	2	3	141.6	84.4%
106	01101010	3	1	143.1	86.4%
98	01100010	3	2	144.1	87.6%
139	10001011	2	2	144.2	87.7%
166	10100110	1	3	149.4	94.6%
46	00101110	2	2	150.0	95.3%
102	01100110	2	2	150.7	96.2%
38	00100110	2	3	151.4	97.1%
110	01101110	2	1	153.3	99.7%
230	11100110	1	2	154.1	100.7%
240	11110000	2	2	160.6	109.1%
250	11111010	2	0	162.7	111.9%
178	10110010	2	2	165.3	115.2%
168	10101000	2	3	167.6	118.2%
232	11101000	2	2	168.4	119.2%

248	11111000	2	1	168.5	119.3%
174	10101110	1	2	169.0	120.0%
162	10100010	2	3	174.2	126.9%
160	10100000	2	4	174.6	127.4%
176	10110000	2	3	179.1	133.2%
242	11110010	2	1	180.4	134.8%
186	10111010	2	1	181.5	136.4%
184	10111000	2	2	181.9	136.9%
224	11100000	2	3	188.6	145.5%
226	11100010	2	2	194.3	153.0%
170	10101010	2	2	198.5	158.5%
234	11101010	2	1	207.8	170.5%

Table 6.11: Average Vehicle Delay (s) for Weighting Parameter Sweep I - All Strategies

Chapter 7

Additional Weighting Considerations to Reduce Inefficiencies of the ERIS Scheduler

7.1 Overview

Chapter 6 explored three extensions to our baseline ERIS scheduler to further reduce delay. We observed that downgrading the importance of select movements (by using a weighting factor below 1.0) led to a reduction in delay when compared against the baseline as well as when compared against the other two extensions. In this chapter, we explore weighting movements in additional detail.

We first present a deterministic queuing model to support our results from Chapter 6. We then examine prioritizing the critical movements on a real-world network and observe a reduction in delay. Next, we present several additional experiments further comparing different weighting strategies. We summarize our analyses with a discussion of when to apply weighting strategies and then conclude.

7.2 Deterministic Queuing Model

In this section, we present a deterministic queuing model to provide intuition supporting the result from Chapter 6 that prioritizing critical movements reduces overall delay.

We begin by introducing two actuated control strategies that we will examine throughout this section. Next, we present our formal model. We present several examples calculating phase duration and delay for our actuated control strategies. We explore how higher arrival rates lead to longer cycle durations and higher average delays. We then employ the model to contrast the two actuated control strategies to demonstrate that as demand increases, prioritizing the critical movements is more important.

7.2.1 Actuated Control Strategies

Before presenting the base model, we briefly introduce the two strategies that we will compare. Recall from our discussion of actuated control in Section 2.3.4 that at a traffic signal running an actuated control strategy, a phase will remain green as long as at least one vehicle is detected along the movements of the phase. While there is often a delay between the last detection of a vehicle and the end of the phase, here we will assume that movements end instantaneously once no more vehicles are queued. We consider two versions of actuated control:

1. Fully Actuated Control - whenever a vehicle is queued along any movement of the active phase, the phase will remain green; otherwise, the phase will be terminated.
2. Critical Movement Actuated Control - whenever a vehicle is queued along the **critical** movement of the active phase, the phase will remain green; otherwise, the phase will be terminated.

7.2.2 Model Definition

Overview

We model a single intersection with four movements - north straight, south straight, east straight, and west straight. We specify that the north and east movements are critical movements while the south and west movements are non-critical.

We model the number of vehicles along each the four movements using a separate state-dependent queuing system. For the critical movements (north and east), vehicles continually arrive in the queue a deterministic rate, λ_C. Similarly, for the non-critical movements (south and west) vehicles continually arrive in the queue at a strictly lower deterministic rate, λ_N.

Our intersection has two signal phases - a shared green phase for the north and south movements and a shared green phase for the east and west movements. When a phase is active, the two corresponding movements receive a green signal. Vehicles in both of the corresponding queuing systems are serviced at deterministic rate S. (When a movement is inactive, the queue is not serviced). To ensure that the intersection has sufficient capacity to serve all vehicles, we require that $S > 2 * \lambda_C$. Additionally, we dictate that when a movement receives a green signal and the corresponding queue is empty, vehicles arriving are instantly served by the movement without experiencing delay.

A phase remains active until the selected actuated control strategy decides to terminate the phase: Once both queues for an active phase are empty (for Fully Actuated Control) or the critical movement queue is empty (for Critical Movement Actuated Control), the phase terminates and we advance to a short change-over that is D seconds long. (During this time, vehicles continue to arrive at the appropriate arrival rates and no vehicles are served). After the change-over completes, the other phase receives a green signal, and the respective queues gradually decrease as described above. This process repeats.

In addition to the aforementioned arrival process (using λ_C and λ_N), we also specify that during the very first phase (north/south), at the moment that the north queue empties, an additional one-time burst of Δ vehicles instantaneously arrives and is added to the south (non-critical) movement queue. (This one-time burst only occurs during the first north/south phase, not during any future phases).

Observe that during the first phase, Fully Actuated Control will extend the phase for these Δ vehicles while Critical Movement Actuated Control will not extend the phase (as these vehicles are on a non-critical movement). This difference in behavior during the first phase is the primary factor that allows our model to contrast the actuated control strategies.

In order to simplify our calculations, we make two additional assumptions. First, we treat vehicles as divisible entities. Second, arrivals and departures are deterministic.

Our model will calculate the phase duration, the number of vehicles served, total vehicle delay, and other pieces of information for 10 successive phases. Our model begins with a north/south phase. To initialize the starting queues for the first phase, we specify the duration Z (which can be thought of as the duration since the previous north/south phase ended) and multiply it by the aforementioned arrival rates.

Formulation, $\Delta = 0$

We now present the general calculations that our model performs to calculate successive phase start times, initial queues, and phase durations. We begin by presenting these calculations for the case when $\Delta = 0$, as this is the simplest case for our model.

Our model calculates the relevant information for phases in order from the first phase to the final phase. The model repeats the following process for every phase. Note that we will use the subscript i to denote that a certain value corresponds to the i-th phase. The first (initial) phase will be indexed using $i = 0$.

First, the model calculates the phase's start time (denoted phase-start-time). The first phase is initialized to begin at time 0. Using the start time of the previous phase, the phase duration (denoted phase-dur) of the previous phase, and the change-over duration (D), the model can calculate the time at which other phases begin:

$$\text{phase-start-time}_i = \begin{cases} 0 & \text{if } i = 0 \\ \text{phase-start-time}_{i-1} + \text{phase-dur}_{i-1} + \text{D} & \text{otherwise} \end{cases}$$

Next, the model calculates the duration since the phase was last green (denoted phase-queuing-dur). For the very first phase, this value is seeded from our input parameter Z. For all other phases, this value is the sum of the duration of the previous phase and two times the change-over duration (D):

$$\text{phase-queuing-dur}_i = \begin{cases} Z & \text{if } i = 0 \\ \text{phase-dur}_{i-1} + 2 * \text{D} & \text{otherwise} \end{cases}$$

The model then uses this value to calculate the initial queues for the critical movement and the non-critical movement (denoted crit-init-queue and non-crit-init-queue, respectively). The initial queues are the product of the duration since the phase was last green and the corresponding arrival rate; this is because vehicles have been arriving since the phase ended and as we will see below, with $\Delta = 0$, both queues were empty when the phase ended.

crit-init-queue$_i = \lambda_C *$ phase-queuing-dur$_i$

non-crit-init-queue$_i = \lambda_N *$ phase-queuing-dur$_i$

After calculating the initial queues, the model calculates the duration required to empty each queue. When a movement is active, the queue serves vehicles at rate S, while vehicles continue to join the queue at rate λ_C or λ_N. Thus, the following equations specify the duration required to empty the critical or non-critical queues (denoted crit-empty-dur and non-crit-empty-dur, respectively) given the initial queue lengths:

$$\text{crit-empty-dur}_i = \frac{\text{crit-init-queue}_i}{S - \lambda_C}$$

$$\text{non-crit-empty-dur}_i = \frac{\text{non-crit-init-queue}_i}{S - \lambda_N}$$

Note that crit-empty-dur$_i >$ non-crit-empty-dur$_i$ because crit-init-queue$_i >$ non-crit-init-queue$_i$ and $\lambda_C > \lambda_N$.

Now that the model knows how long is required to empty queues, the model calculates the phase duration. Critical Movement Actuated Control will end the phase after the critical movement empties, which requires crit-empty-dur$_i$. Fully Actuated Control will end the phase after both movements empty, which is the maximum of crit-empty-dur$_i$ and non-crit-empty-dur$_i$, namely crit-empty-dur$_i$. Thus, regardless of the control strategy chosen:

$$\text{phase-dur}_i = \frac{\text{crit-init-queue}_i}{S - \lambda_C}$$

(Note that this will not be the case when $\Delta > 0$; we explore this later).

After calculating the phase duration, the model calculates the number of vehicles served along each movement as well as the total delay for these vehicles. As these calculations are quite detailed and do not influence the calculation of successive phases, these calculations are presented in this chapter's appendix.

After completing all calculations for the initial phase, our model uses the calculated phase duration to begin the calculation for the next phase. This process repeats for the ten phases in the model's horizon.

Formulation, $\Delta > 0$, Fully Actuated Control

When $\Delta > 0$, calculating the phase duration, the number of vehicles served, initial queues, and vehicle delay is more complex. Here, we detail modifications to our above formulas when $\Delta > 0$ for phase duration and initial queues. We begin by discussing modifications for Fully Actuated Control.

Recall from the model overview that Δ vehicles arrive during the first phase at the instant that the north queue empties. These vehicles join the south queue. Fully Actuated Control decides to extend the north/south phase to serve these vehicles. First, this increases the phase duration.

Note that the critical movement queue is empty when these Δ vehicles arrive and remains empty as vehicles arriving are instantaneously served. Thus, the duration required to clear the south queue is the additional duration required for the phase. The model calculates the duration required to serve these vehicles (denoted Δ-dur) and increases the total phase duration accordingly.

$$\Delta\text{-dur}_{0,FAC} = \frac{\Delta}{S - \lambda_N}$$

Thus, the new phase duration becomes:

$$\text{phase-dur}_{0,FAC} = \frac{\text{crit-init-queue}_i}{S - \lambda_C} + \frac{\Delta}{S - \lambda_N}$$

As denoted in the subscripts, this modification only applies to the initial phase and only to Fully Actuated Control (FAC). Our model continues to use the prior formula to calculate phase duration for future phases.

The calculations for the number of vehicles served and the total delay is also modified. As above, we defer our presentation of these calculations to the chapter's appendix. Other formulas (e.g. the formula for phase-start-time) are unchanged.

Formulation, $\Delta > 0$, Critical Movement Actuated Control

We now discuss formula modifications for Critical Movement Actuated Control (CMAC). Since these vehicles are on the south (non-critical) movement, Critical Movement Actuated Control will not immediately serve the Δ vehicles. Rather, they will be delayed until the following north/south phase. Thus, when our model calculates the sequence of phases for Critical Movement Actuated Control, it must keep track of unserved vehicles along the south movement between phases (denoted south-unserved). After the first north/south phase, this value will be equal to Δ. Formally,

$$\text{south-unserved}_{0,CMAC} = \Delta$$

The initial non-critical queue length must be updated accordingly for these unserved vehicles for the second north/south phase and possibly later south phases.

$$\text{non-crit-init-queue}_{i,CMAC} = \lambda_N * \text{phase-queuing-dur}_i + \text{south-unserved}_{i-2,CMAC} \qquad \text{for } i \geq 2 \text{ and } i \text{ is even}$$

Since Critical Movement Actuated Control does not extend the initial phase for these Δ vehicles, we use the same formula for phase duration that was presented for the $\Delta = 0$ case. However, as before, the model must update its calculations for the number of vehicles served and total delay. We present these calculations in this chapter's appendix. Additionally, we also detail how south-unserved is calculated for later phases in the appendix. Other formulas (e.g. the formula for phase-start-time) are unchanged.

Formulation, Equilibrium Phase Duration

Before presenting several examples of our model, we define and calculate the equilibrium phase duration.

We define the equilibrium phase duration as the phase duration when $\Delta = 0$ such that once a phase is this duration, all future phases will also be this duration. Alternatively, the equilibrium phase duration can be viewed as an attractive fixed point. We are interested in calculating this value because we will utilize it to determine the value for our parameter Z to seed the queues for the initial phase.

Given the values of λ_C, D, and S, we perform the following steps to derive the equilibrium phase duration, P_e. (We do not need to consider λ_N because as we saw earlier, only λ_C influences phase duration when $\Delta = 0$):

- If the equilibrium phase is P_e seconds, then the full equilibrium cycle (C_e) which consists of two phases and two change-overs is $C_e = 2 * P_e + 2 * D$.
- Along a given critical movement, the number of vehicles that arrive during a single equilibrium cycle is $\lambda_C * C_e$.
- For the equilibrium phase to completely serve these vehicles, it requires $\frac{\lambda_C * C_e}{S}$ seconds.
- From above, we know that an equilibrium phase is P_e seconds.
- Setting these equal we obtain, $P_e = \frac{\lambda_C * C_e}{S}$.
- Plugging in our earlier value of C_e and solving for P_e, we obtain $P_e = \frac{2 * \lambda_C * D}{S - 2 * \lambda_C}$.
- Similarly, we find that $C_e = \frac{2 * D * S}{S - 2 * \lambda_C}$.
- By selecting $Z = P_e + 2 * D = \frac{2 * D * (S - \lambda_C)}{S - 2 * \lambda_C}$, we will ensure that the next phase is P_e seconds and that the cycle duration matches the equilibrium cycle duration.

7.2.3 Model Evaluation with Δ = 0

In this subsection, we walk-through two evaluations of our model with $\Delta = 0$ (no demand burst along the south movement prior to the end of the first phase). As mentioned above, when setting $\Delta = 0$, both actuated control strategies perform identically; this allows us to focus on explaining how we calculate phase duration and delay.

Model Example #1: Cycle Duration Initialized to Equilibrium Value

In our first model evaluation, we will specify that every minute, 10 vehicles arrive along each critical movement ($\lambda_C = \frac{10}{60} = \frac{1}{6}$) and 5 vehicles arrive along each non-critical movement ($\lambda_N = \frac{1}{12}$). When a movement is active and vehicles are present, it serves a vehicle every 2.5 seconds ($S = \frac{1}{2.5} = 0.4$). We require a 5 second change-over between successive green phases ($D = 5$). We select Z such that the cycle duration will be initialized to its equilibrium value ($Z = 35$). Finally, as mentioned above, there is no demand increase along the south movement prior to the end of the first phase ($\Delta = 0$). In summary, we use the following parameter settings for this example:

- $\lambda_C = \frac{1}{6}$
- $\lambda_N = \frac{1}{12}$
- $S = 0.4$
- $D = 5$
- $Z = 35$
- $\Delta = 0$

Our model evaluation proceeds as follows: First, our model specifies that the first phase is a north/south phase and given our parameter $Z = 35$, it has been 35 seconds since the end of the last north/south phase. The model uses this information and the provided values of λ_C and λ_N to calculate the initial queues at the instant this phase begins (e.g. $35 * \frac{1}{6} \approx 5.8$ for the critical movement; $35 * \frac{1}{12} \approx 2.9$ for the non-critical movement). Partial results are presented in Table 7.1. (All values presented in tables for the remainder of this section will be rounded to one decimal point).

Start Time	Cycle #	Phase	Duration Since Phase Last Green	Crit Initial Queue	Non-Crit Initial Queue	Phase Duration	Crit Served	Non-Crit Served	Crit Delay	Non-Crit Delay
0	1	North/South	35	5.8	2.9					

Table 7.1: Model Example #1, Partial Calculation #1

Next, the model calculations the duration of this phase: While the north/south phase is active, for the north movement, every second, until the initial queue empties, $\lambda_C = \frac{1}{6}$ vehicles arrive while $S = 0.4$ vehicles depart. Thus the queue decreases by $(0.4 - \frac{1}{6})$ vehicles per second until it empties. We can calculate the necessary time (crit-empty-dur$_0$) to empty the queue by solving crit-empty-dur$_0$ $* (0.4 - \frac{1}{6}) = \frac{35}{6}$ for crit-empty-dur$_0$. We find that crit-empty-dur$_0$ $= 25$. Similarly, we can calculate the time required to empty the south queue using the equation non-crit-empty-dur$_0$ $* (0.4 - \frac{1}{12}) = \frac{35}{12}$. Solving yields non-crit-empty-dur$_0$ $= \frac{175}{19} \approx 9.2$.

For this example, regardless of which actuated control strategy we select, the phase will be 25 seconds long. Critical Movement Actuated Control keeps the phase green for 25 seconds since this is the amount of time required to empty the critical queue. Fully Actuated Control keeps the phase green for the maximum of crit-empty-dur$_0$ and non-crit-empty-dur$_0$, which is also 25 seconds.

We then use this duration (25) along with the arrival rates to calculate the additional number of vehicles that arrive along each movement while the phase is active. For example, for the critical movement, we perform the following calculation to determine the additional arrivals while the phase is active: $(25) * \frac{1}{6} \approx 4.2$. Adding these arrivals to the initial queue, we calculate that a total of 10 $(5.8 + 4.2)$ vehicles wish to proceed through the intersection along this movement. (One can also view 10 as the number of vehicles that arrive along the north movement between the end of the previous north/south phase and the end of this north/south phase).

From our earlier calculation of the phase duration, we know that there is sufficient time to clear the north movement queue. However, we can confirm this by multiplying the service rate by the phase duration to calculate the maximum phase capacity: $25 * 0.4 = 10$. As expected, there is sufficient time to serve the 10 vehicles along the north movement.

We perform an analogous calculation for the south movement and update our partial results table, presented in Table 7.2.

Start Time	Cycle #	Phase	Duration Since Phase Last Green	Crit Initial Queue	Non-Crit Initial Queue	Phase Duration	Crit Served	Non-Crit Served	Crit Delay	Non-Crit Delay
0	1	North/South	35	5.8	2.9	25	10	5		

Table 7.2: Model Example #1, Partial Calculation #2

We next calculate the delay for vehicles along both movements. For the north movement, vehicles arrived uniformly over the prior 60 seconds and vehicles departed uniformly over the last 35 seconds. We can obtain their average delay by subtracting the average arrival time from the average departure time: $\frac{35+60}{2} - \frac{0+60}{2} = \frac{35}{2}$. A total of 10 vehicles travelled along the north movement, so the total delay for this movement is 175. (Unless specified, for all delay calculations, we use the end time of the previous matching phase as the zero-reference point to simplify our average delay calculations).

Calculating the delay for the south movement requires an additional step. First, recall that it required approximately 9.2 seconds to empty the south movement's queue. We first determine how many vehicles arrived up until this point in time along the south movement: $(35+9.2)*\frac{1}{12} = 3.68\overline{3}$. We also calculate the average delay using the same process as above, subtracting the average arrival time from the average departure time: $\frac{35+(35+9.2)}{2} - \frac{0+(35+9.2)}{2} = \frac{35}{2}$. We multiply these values to obtain the total south movement delay ($3.68\overline{3} * \frac{35}{2} \approx 64.5$). (Recall that vehicles arriving after the south queue empties will not be delayed). The result of this computation as well as the north movement delay are added to our partial results table, presented in Table 7.3.

Start Time	Cycle #	Phase	Duration Since Phase Last Green	Crit Initial Queue	Non-Crit Initial Queue	Phase Duration	Crit Served	Non-Crit Served	Crit Delay	Non-Crit Delay
0	1	North/South	35	5.8	2.9	25	10	5	175	64.5

Table 7.3: Model Example #1, Partial Calculation #3

We next consider the following phase, which will be an east/west phase. The phase begins at time 30 since the first phase was 25 seconds long and $D = 5$. Additionally, it has been 35 seconds since the most recent east/west phase. (5 seconds of change-over after the end of the last east/west phase, 25 seconds for the north/south phase we discussed above, and 5 more seconds of change-over after the end of the north/south phase). These values are added to our partial results table, updated in Table 7.4.

Start Time	Cycle #	Phase	Duration Since Phase Last Green	Crit Initial Queue	Non-Crit Initial Queue	Phase Duration	Crit Served	Non-Crit Served	Crit Delay	Non-Crit Delay
0	1	North/South	35	5.8	2.9	25	10	5	175	64.5
30	1	East/West	35							

Table 7.4: Model Example #1, Partial Calculation #4

Following an identical procedure as described above, we can populate the rest of this row. Since *Duration Since Phase Last Green* has not changed and $\Delta = 0$, the calculation results for this row are identical to the calculations from the previous row. Our partial calculations are updated in Table 7.5.

Start Time	Cycle #	Phase	Duration Since Phase Last Green	Crit Initial Queue	Non-Crit Initial Queue	Phase Duration	Crit Served	Non-Crit Served	Crit Delay	Non-Crit Delay
0	1	North/South	35	5.8	2.9	25	10	5	175	64.5
30	1	East/West	35	5.8	2.9	25	10	5	175	64.5

Table 7.5: Model Example #1, Partial Calculation #5

We continue our calculations across the next 8 phases. Since *Duration Since Phase Last Green* will remain at 35 for all rows in this table and $\Delta = 0$, the calculation results for remaining rows are identical to the calculations for the first two rows (excluding the start time). The fully populated table, with additional lines for totals and averages, is presented in Table 7.6.

Start Time	Cycle #	Phase	Duration Since Phase Last Green	Crit Initial Queue	Non-Crit Initial Queue	Phase Duration	Crit Served	Non-Crit Served	Crit Delay	Non-Crit Delay
0	1	North/South	35	5.8	2.9	25	10	5	175	64.5
30	1	East/West	35	5.8	2.9	25	10	5	175	64.5
60	2	North/South	35	5.8	2.9	25	10	5	175	64.5
90	2	East/West	35	5.8	2.9	25	10	5	175	64.5
120	3	North/South	35	5.8	2.9	25	10	5	175	64.5
150	3	East/West	35	5.8	2.9	25	10	5	175	64.5
180	4	North/South	35	5.8	2.9	25	10	5	175	64.5
210	4	East/West	35	5.8	2.9	25	10	5	175	64.5
240	5	North/South	35	5.8	2.9	25	10	5	175	64.5
270	5	East/West	35	5.8	2.9	25	10	5	175	64.5
					Totals:	250	100	50	1750	645
					Average Delay (all vehs):	16.0				

Table 7.6: Model Example #1, Full Model

Model Example #2: Cycle Duration Not Initialized to Equilibrium

If we had chosen a different value for Z, we would not start at our equilibrium cycle duration, but rather, would approach it asymptotically. An example with $Z = 45$ (former value: 35) and other parameters unchanged, is presented in Table 7.7.

Start Time	Cycle #	Phase	Duration Since Phase Last Green	Crit Initial Queue	Non-Crit Initial Queue	Phase Duration	Crit Served	Non-Crit Served	Crit Delay	Non-Crit Delay
0.0	1	North/South	**45.0**	7.5	3.8	32.1	12.9	6.4	289.3	106.6
37.1	1	East/West	42.1	7.0	3.5	30.1	12.0	6.0	253.7	93.5
72.2	2	North/South	40.1	6.7	3.3	28.6	11.5	5.7	229.7	84.6
105.9	2	East/West	38.6	6.4	3.2	27.6	11.0	5.5	213.3	78.6
138.5	3	North/South	37.6	6.3	3.1	26.9	10.7	5.4	202.0	74.4
170.4	3	East/West	36.9	6.1	3.1	26.3	10.5	5.3	194.1	71.5
201.7	4	North/South	36.3	6.1	3.0	25.9	10.4	5.2	188.5	69.5
232.6	4	East/West	35.9	6.0	3.0	25.7	10.3	5.1	184.6	68.0
363.3	5	North/South	35.7	5.9	3.0	25.5	10.2	5.1	181.8	67.0
293.8	5	East/West	35.5	5.9	3.0	25.3	10.1	5.1	179.9	66.3
					Totals:	274.1	109.7	54.8	2117.0	780.0
					Average Delay (all vehs):	17.6				

Table 7.7: Model Example #2

As in the previous example, the first *Duration Since Phase Last Green* is set to the provided Z parameter. The other values in this row can be calculated from this value as we did in the previous example. For all rows after the first row, this value is the sum of the previous row's phase duration and 2 times the change-over duration (D). Again, the other values of each row can be calculated from the row's *Duration Since Phase Last Green*. Following this process, we can calculate all values to fully populate the table.

We note that compared to the first example, the initial queues in the example are longer and thus the initial phase duration is higher. As we progress through phases, phase durations asymptotically approach the equilibrium phase duration of 25 seconds. We also observe that the average delay per vehicle has increased from 16.0 to 17.6. Longer cycles require vehicles to wait longer. This is an important property of our model that is relevant when we compare the two actuated control strategies later in this section.

7.2.4 Model Evaluation with $\Delta > 0$

Overview

In the previous subsection, we detailed two evaluations of our model when $\Delta = 0$. In these examples, both actuated strategies performed identically since there was never a point at which a critical queue was empty while the corresponding non-critical queue was not empty.

In this subsection, we will examine a case with $\Delta > 0$. When $\Delta > 0$, the actuated control strategies will not act identically. The goal of this subsection is to examine how each of these actuated control strategies modifies our calculations. We begin by briefly presenting an overview of the differences between how the strategies handle these vehicles. We then present calculation examples.

First recall our definition of Δ. When $\Delta > 0$, during the first north/south phase, Δ vehicles arrive along the south (non-critical) movement at the moment at which the north (critical) movement empties. The Fully Actuated Control strategy will immediately serve these vehicles. This leads to increasing the phase duration and thus when the next phase begins, additional vehicles have queued. As in Model Example #2 presented above, we observe the phase duration jump above its steady-state value and then gradually decrease asymptotically to the steady-state duration. Additionally, delay will increase for vehicles arriving during later phases.

On the other hand, Critical Movement Actuated Control will not serve these vehicles immediately. These vehicles will be delayed until the beginning of the next north/south phase. Vehicles arriving along the south movement for the second cycle will not be served until the Δ vehicles are served. Their delay will also increase. Even though delay is increased, phase durations do not change. Vehicles travelling along the other movements do not experience a change in their delay (compared to the $\Delta = 0$ case).

We are interested in considering the increase in delay for both actuated control strategies to determine which strategy is better. We present an example of the relevant calculations for both actuated control strategies. These examples will consider a single set of parameters.

For this example, except for Δ which is now non-zero ($\Delta = 2$), parameters are unchanged from our first example. Our parameters are listed below:

- $\lambda_C = \frac{1}{6}$
- $\lambda_N = \frac{1}{12}$
- $S = 0.4$
- $D = 5$
- $Z = 35$
- $\Delta = 2$

Fully Actuated Control

We begin by presenting the calculations for the Fully Actuated Control setting. We use the aforementioned process to calculate delay but make several adjustments to account for the $\Delta = 2$ vehicles that arrive as the first north/south phase is about to end. We discuss these modifications by presenting partial results tables as we did above.

First, we note that the first 25 seconds of the first north/south phase will proceed identically to Model Example #1 presented above. 10 vehicles will depart along the north movement, 5 vehicles will depart along the south movement, and the respective delays of 175 and 64.5 are unchanged.

We consider how our model changes after these first 25 seconds when the $\Delta = 2$ vehicles arrive along the south movement and join the queue. By definition, the Fully Actuated Control strategy will keep the phase active to serve these vehicles since the south queue is no longer empty.

We then calculate how long is required to clear these additional vehicles from the south movement queue. Note that additional vehicles will continue to arrive along both the north movement and the south movement as the south queue clears.

Recall that in our first example, we divided the south movement queue length by the difference between S and λ_N to calculate how long it would require to clear the south movement queue. We again perform this calculation to determine how long it will require to clear the south queue. We divide the south movement queue length (now equal to Δ) by the difference between S and λ_N. We obtain $\frac{2.0}{0.4-\frac{1}{12}} \approx 6.3$. Thus in our results table, the phase duration for the first north/south movement is increased from 25.0 seconds to 31.3 seconds.

Given the longer phase duration, we must update the number of critical (north) arrivals as they continue to arrive in this 6.3 second window. An additional $6.3 * \frac{1}{6} \approx 1.1$ vehicles arrive during this interval. Thus, a total of $10 + 1.1 = 11.1$ vehicles are served by the north movement. Since the north movement queue is empty while these vehicles arrive, they depart instantaneously and do not impact the total north movement delay. We briefly pause to present an updated calculation in Table 7.8 below.

Start Time	Cycle #	Phase	Duration Since Phase Last Green	Crit Initial Queue	Non-Crit Initial Queue	Phase Duration	Crit Served	Non-Crit Served	Crit Delay	Non-Crit Delay
0	1	North/South	35	5.8	2.9	31.3	11.1		175	

Table 7.8: Model Example when $\Delta = 2$, Fully Actuated Control, Partial Calculation #1

The number of non-critical arrivals is increased by the sum of Δ and the number of vehicles arriving during the 6.3 second window. Specifically, $6.3 * \frac{1}{12} \approx 0.5$ vehicles arrive during the 6.3 second window (excluding the $\Delta = 2$ arrivals).

Recall that our non-critical delay for the first 5 vehicles remains at 64.5 as we calculated in our first example above. We must increase the non-critical delay for these additional vehicles.

For the Δ vehicles, their average arrival time is 60 (35 + 25) because from our definition, they all arrive at the instant when the north movement queue empties. They depart over the next 5 seconds; their average departure time is thus $\frac{(60+0)+(60+5.0)}{2} = 62.5$. Their average delay is $62.5 - 60 = 2.5$ seconds. We have 2 such vehicles, so non-critical delay increases by 5.0 seconds due to these vehicles.

We must also increase delay from the fractional 0.5 vehicle arriving during the 6.3 second window. Its average arrival time is $\frac{60+66.3}{2} \approx 63.2$. Its average departure time is $\frac{65+66.3}{2} \approx 65.7$. (It does not begin departing until the Δ vehicles finish departing at time 65. It finishes departing when the phase ends, 1.3 seconds later). The average delay for this vehicle is thus $65.7 - 63.2 = 2.5$ seconds. Multiplying by the vehicle's size (0.5) yields a total delay of 1.3 (rounded).

Thus, non-critical delay increases by a total of 6.3 seconds ($5.0 + 1.3$) over the baseline value of 64.5. Our final non-critical delay is now 70.8. We again pause to present our partial calculation in Table 7.9.

Start Time	Cycle #	Phase	Duration Since Phase Last Green	Crit Initial Queue	Non-Crit Initial Queue	Phase Duration	Crit Served	Non-Crit Served	Crit Delay	Non-Crit Delay
0	1	North/South	35	5.8	2.9	31.3	11.1	7.5	175	70.8

Table 7.9: Model Example when $\Delta = 2$, Fully Actuated Control, Partial Calculation #2

This phase required 31.3 seconds. The following phase's *Duration Since Phase Last Green* is thus 41.3 (31.3+2*5). (Each 5 represents a single change-over). The remainder of this line is populated according to this value. Similar to the second example presented above, our phase duration asymptotically approaches 25.0 seconds. The full results are presented in Table 7.10.

Start Time	Cycle #	Phase	Duration Since Phase Last Green	Crit Initial Queue	Non-Crit Initial Queue	Phase Duration	Crit Served	Non-Crit Served	Crit Delay	Non-Crit Delay
0.0	1	North/South	35.0	5.8	2.9	31.3	11.1	7.5	175.0	70.8
36.3	1	East/West	41.3	6.9	3.4	29.5	11.8	5.9	243.9	89.8
70.8	2	North/South	39.5	6.6	3.3	28.2	11.3	5.6	223.0	82.2
104.0	2	East/West	38.2	6.4	3.2	27.3	10.9	5.5	208.7	76.9
136.4	3	North/South	37.3	6.2	3.1	26.6	10.7	5.3	198.8	73.2
168.0	3	East/West	36.6	6.1	3.1	26.2	10.5	5.2	191.8	70.7
199.2	4	North/South	36.2	6.0	3.0	25.8	10.3	5.2	186.9	68.9
230.0	4	East/West	35.8	6.0	3.0	25.6	10.2	5.1	183.5	67.6
260.6	5	North/South	35.6	5.9	3.0	25.4	10.2	5.1	181.0	66.7
291.0	5	East/West	35.4	5.9	3.0	25.3	10.1	5.1	179.3	66.1
					Totals:	271.3	107.1	55.5	1972.0	732.8
					Average Delay (all vehs):	16.6				

Table 7.10: Model Example when $\Delta = 2$, Fully Actuated Control, Full Model

Critical Movement Actuated Control

We now consider the Critical Movement Actuated Control setting. Again, we will use a similar process to calculate delay but again must make several adjustments to account for the $\Delta = 2$

vehicles that arrive as the first north/south phase is about to end. We discuss these modifications by presenting partial results tables as we did above.

With Critical Movement Actuated Control, the first north/south phase ends even though $\Delta = 2$ vehicles have arrived along the south movement since this movement is non-critical. We begin with a table very similar to that from our first example. We only need to calculate the initial non-critical queue, the number of non-critical vehicles served, and the new delay for the south movement (non-critical) vehicles departing during the second cycle and update our table accordingly. Our partial results are presented below in Table 7.11 with the appropriate spaces indicated with a "**?**". This table is identical to Table 7.6 except for these missing values and the totals which are not displayed (because they are based on these yet to be calculated values).

Start Time	Cycle #	Phase	Duration Since Phase Last Green	Crit Initial Queue	Non-Crit Initial Queue	Phase Duration	Crit Served	Non-Crit Served	Crit Delay	Non-Crit Delay
0	1	North/South	35	5.8	2.9	25	10	5	175	64.5
30	1	East/West	35	5.8	2.9	25	10	5	175	64.5
60	2	North/South	35	5.8	**?**	25	10	**?**	175	**?**
90	2	East/West	35	5.8	2.9	25	10	5	175	64.5
120	3	North/South	35	5.8	2.9	25	10	5	175	64.5
150	3	East/West	35	5.8	2.9	25	10	5	175	64.5
180	4	North/South	35	5.8	2.9	25	10	5	175	64.5
210	4	East/West	35	5.8	2.9	25	10	5	175	64.5
240	5	North/South	35	5.8	2.9	25	10	5	175	64.5
270	5	East/West	35	5.8	2.9	25	10	5	175	64.5

Table 7.11: Model Example when Δ = 2, Critical Movement Actuated Control, Partial Calculation

We now discuss how to calculate these missing values for the second north/south phase. First, observe that excluding the $\Delta = 2$ arrivals, it is still the case that $\frac{35}{12} \approx 2.9$ vehicles arrive along the south movement during the 35 seconds since last the last north/south phase. Adding these to our $\Delta = 2$ arrivals, we obtain an initial non-critical queue of $\frac{59}{12} \approx 4.9$ vehicles. Similarly, it is still the case that 2.1 additional vehicles arrive along the non-critical movement during the phase. This means that there will be a total of 7 vehicles to be served by the movement during the phase $(2.9 + 2 + 2.1 = 7)$. Secondly, observe that there is sufficient time to serve these 7 vehicles $(\frac{7}{0.4} = 17.5 < 25)$ within the usual phase duration of 25 seconds. Thus, the number of non-critical vehicles served becomes 7.

Calculating the delay for these 7 vehicles is performed in several steps. First, we calculate the delay for the $\Delta = 2$ vehicles that arrived at the end of the previous north/south phase. These vehicles arrived 35 seconds prior to the beginning of the phase. They will be served between the beginning of the phase and 5 seconds after the phase begins, which leads to an average delay of 37.5 or a total delay of 75.0 seconds across both vehicles.

We now calculate the delay for the five other non-critical vehicles. As we have done several times before, we solve for the time at which the non-critical queue empties (note that we use the initial non-critical queue length of 4.9): non-crit-empty-dur$_2$ $* (0.4 - \frac{1}{12}) = 4.9$. Solving for non-crit-empty-dur$_2$, we obtain non-crit-empty-dur$_2$ $= 15.5$. This says that it will take 15.5 seconds to clear the non-critical queue, including time spent serving the $\Delta = 2$ vehicles. During these 15.5 seconds, we serve $15.5 * 0.4 = 6.2$ vehicles. Since we have already calculated the delay for the first 2 vehicles, we have 4.2 vehicles for which we must calculate the delay.

These 4.2 vehicles begin proceeding through the intersection 5.0 seconds after the phase begins (once the Δ vehicles have been served). They finish 15.5 seconds after the phase begins. Adding this to the *Duration Since Phase Last Green*, these vehicles depart an average of $\frac{(35+5.0)+(35+15.5)}{2} \approx 45.3$ seconds after the phase begins. Their average arrival time can be calculated as the average of 0 (vehicles begin to arrive immediately after the previous north/south phase ends) and 35+15.5 (the time at which the queue clears). Our average arrival time is thus 25.3 (rounded). Our average delay is $45.3 - 25.3 = 20.0$ seconds. This applies to 4.2 vehicles; thus, the total delay for these vehicles is $4.2 * 20.0 = 84.0$ seconds. Combining this delay with the earlier 75.0 second delay for the Δ vehicles, we obtain 159.0 seconds. These values are incorporated into Table 7.12.

Start Time	Cycle #	Phase	Duration Since Phase Last Green	Crit Initial Queue	Non-Crit Initial Queue	Phase Duration	Crit Served	Non-Crit Served	Crit Delay	Non-Crit Delay
0	1	North/South	35	5.8	2.9	25	10	5	175	64.5
30	1	East/West	35	5.8	2.9	25	10	5	175	64.5
60	2	North/South	35	5.8	**4.9**	25	10	**7**	175	**159.0**
90	2	East/West	35	5.8	2.9	25	10	5	175	64.5
120	3	North/South	35	5.8	2.9	25	10	5	175	64.5
150	3	East/West	35	5.8	2.9	25	10	5	175	64.5
180	4	North/South	35	5.8	2.9	25	10	5	175	64.5
210	4	East/West	35	5.8	2.9	25	10	5	175	64.5
240	5	North/South	35	5.8	2.9	25	10	5	175	64.5
270	5	East/West	35	5.8	2.9	25	10	5	175	64.5
					Totals:	250.0	100.0	52.0	1750.0	739.5
					Average Delay (all vehs):	16.4				

Table 7.12: Model Example when Δ = 2, Critical Movement Actuated Control, Full Model

We note that for this particular example, the south (non-critical) queue cleared during the second north/south phase. For some combinations of parameters, it will require more than one phase to fully clear the south queue. This requires us to calculate how many south vehicles were served and how many south vehicles remain at the end of each north/south phase. As with the Δ vehicles, care must be taken to separately calculate the delay for such vehicles; otherwise the calculation proceeds similarly. We do not present an example of such a calculation here, but have implemented this functionality in the code which we present below.

7.2.5 Considering Different Arrival Rates

We developed a Python program that calculates the schedule of phases and resulting delay as we did in our examples above. For a given set of parameters, our program reports the total average delay, which is the primary metric on which we assess different parameters and compare the actuated control strategies. Our program implements the appropriate logic for the two actuated control strategies as described above. When running with the parameters from the examples presented above, our program's output matches the output from the above models.

In this subsection, we run our model on a variety of arrival rates (and resulting cycle durations). We begin by considering a variety of values for λ_C, within the range (0.0,0.2). (If $\lambda_C = 0.0$, no vehicles arrive so the model is not interesting. We set $S = 0.4$; thus if $\lambda_C \geq 0.2$, our model will be overcapacitated).

For the first portion of our analysis, we set $\Delta = 0$ to examine the interactions between arrival rate, cycle duration, and average delay. We then proceed to consider non-zero values for Δ and discuss how the two control strategies differ.

Our other parameters are defined as follows for all experiments:

- $\lambda_N = \frac{1}{2} * \lambda_C$
- $S = 0.4$
- $D = 5$
- $Z = P_e = \frac{2*D*(S-\lambda_C)}{S-2*\lambda_C}$. (Recall from Section 7.2.2 that this definition of Z initializes the phase duration to the equilibrium phase duration).

Considering the Interactions between Arrival Rate, Cycle Duration, and Delay

We begin by considering the interactions between arrival rate, cycle duration, and average delay for the given range of arrival rates. Figure 7.1 plots cycle duration and average delay for all values of λ_C in our feasible range of (0.0, 0.2). Figure 7.2 zooms in and plots cycle duration and average delay for values of λ_C in the range [0.1, $0.18\overline{6}$], where equilibrium cycle durations are between 20 seconds and 150 seconds.

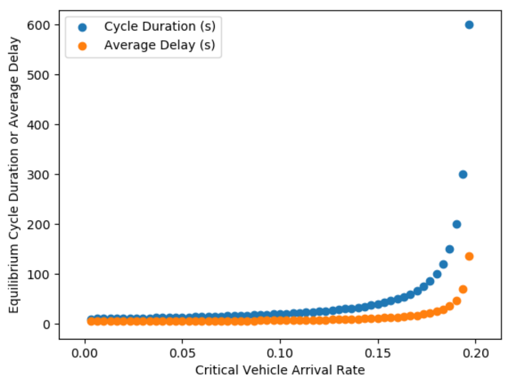


Figure 7.1: Cycle Duration and Average Delay for All Arrival Rates

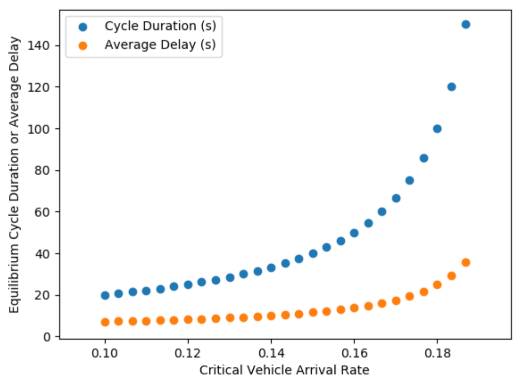


Figure 7.2: Cycle Duration and Average Delay for Selected Arrival Rates

From these two figures, we observe that both cycle duration and average delay are increasing in terms of the critical movement arrival rate (λ_C). These functions are convex, meaning that at low arrival rates, increasing the arrival rate slightly leads to relatively small increases in cycle duration and average delay. Whereas at higher arrival rates, small increases in arrival rates lead to relatively large increases in cycle duration and average delay. The final increase in the critical movement arrival rate from $0.19\overline{3}$ to $0.19\overline{6}$ in Figure 7.1 increases the cycle duration from 300 seconds to 600 seconds.

These extreme cycle durations do not match with our experience; we have not observed traffic signals with cycle durations in excess of 160 seconds. Figure 7.2 presents a tighter range of cycle durations (between 20 seconds and 150 seconds) and it is easier to view the growth in cycle duration and average delay on this figure. For the remainder of this section, we will focus our investigation on this tighter range of arrival rates and cycle durations.

Furthermore, from this analysis, we note that the relationship between cycle duration and delay is almost linear. We present this relationship in Figure 7.3. Specifically, when the cycle duration is at its equilibrium value, the average delay for a vehicle that is delayed is always half of the *phase-queuing-dur*. (Details for calculating average delay are provided in this chapter's appendix). All vehicles travelling along critical movements are delayed, and thus, their average delay is half of the *phase-queuing-dur*.

Only some of the vehicles travelling along the non-critical movement are delayed; thus, the average delay for these vehicles is not perfectly linear. However, in our regression, for the examined arrival rates, we find that the R^2 (the coefficient of determination) is greater than 0.99. Similarly, we find that the average delay across all vehicle types also has an R^2 greater than 0.99.

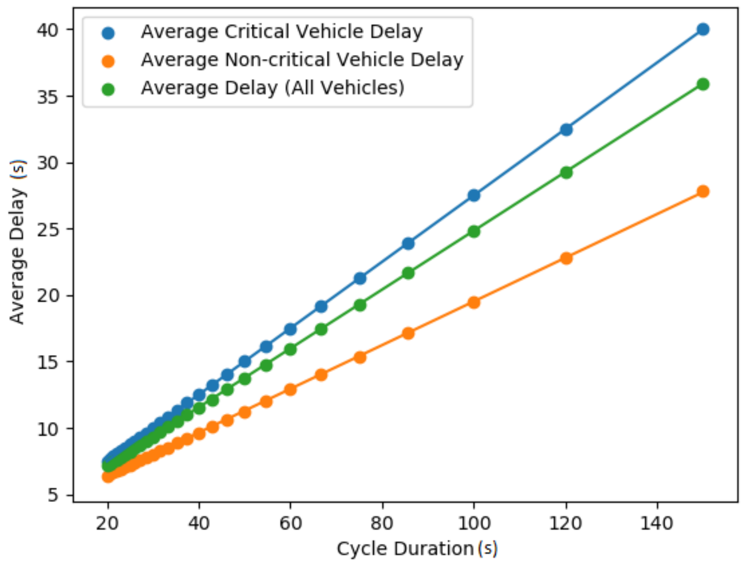


Figure 7.3: Relationship between Cycle Duration and Average Delay

Reparameterizing Analysis Based on Cycle Duration

As can be observed in the three previous figures, when examining cycle duration based on the critical vehicle arrival rate, we notice many more data points along the left side of our figures (corresponding to low arrival rates and low cycle durations). We will reparameterize our further analysis by considering cycle durations 5 seconds apart, between 20 seconds and 150 seconds.

Recall from Section 7.2.2 that $C_e = \frac{2*D*S}{S-2*\lambda_C}$, where C_e is defined as our equilibrium cycle duration. We can rearrange this equation to isolate λ_C: $\lambda_C = \frac{S}{2} - \frac{S*D}{C_e}$. For the following analysis, λ_C will be calculated based on the desired equilibrium cycle duration.

Analysis with $\Delta = 2.0$

We now present our comparison between the two actuated control strategies. We begin by examining the case when $\Delta = 2.0$. Recalling our earlier definition of Δ, two vehicles arrive along the south movement at the instant that the first north/south phase is about to end. Our actuated control strategies will behave differently. We are interested in quantifying the relative impact on delay of each strategy.

In Figure 7.4, we present the results of our model, run for both control strategies, across a range of equilibrium cycle durations in the interval [20, 150]. For each equilibrium cycle duration, we calculate the average delay for both Fully Actuated Control and Critical Movement Actuated Control and plot the percentage change in delay when switching from Fully Actuated Control to Critical Movement Actuated Control. (Negative percentages indicate that Critical Movement Actuated Control outperforms Fully Actuated Control).

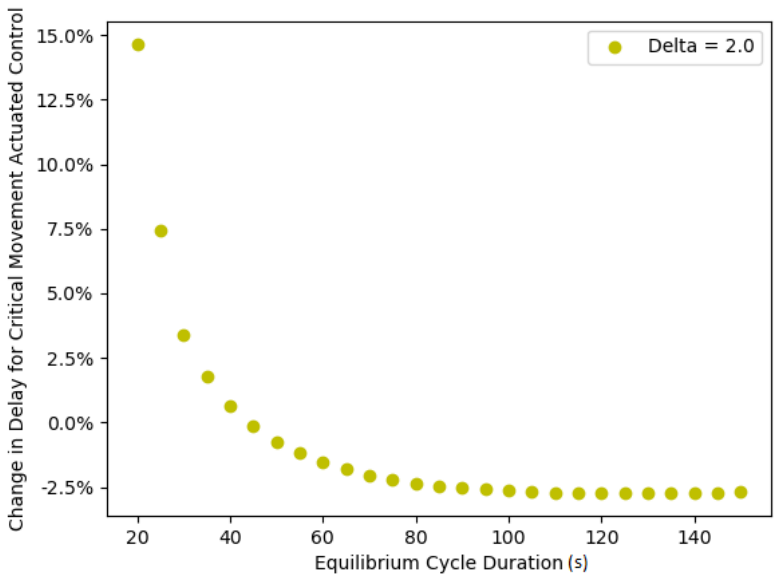


Figure 7.4: Change in Delay for Critical Movement Actuated Control at Various Equilibrium Cycle Durations when $\Delta = 2.0$

We observe that as the equilibrium cycle duration increases, the relative performance of Critical Movement Actuated Control (when compared against Fully Actuated Control) improves. At low equilibrium cycle durations (those 40 seconds and below), prioritizing critical movements with Critical Movement Actuated Control leads to higher average vehicle delay than Fully Actuated Control. Gradually, as the equilibrium cycle duration increases and the associated demand also increases, Critical Movement Actuated Control improves relative to Fully Actuated Control. Critical Movement Actuated Control offers a reduction in delay exceeding 2.0% for the examined equilibrium cycle durations that are 70 seconds and above.

Note that when the equilibrium cycle duration is 70 seconds, the model horizon is $70 * 5 = 350$ seconds ≈ 5.8 minutes. When the equilibrium cycle duration is 160 seconds, the model horizon is $160 * 5 = 800$ seconds ≈ 13.3 minutes. A single decision to extend a phase by 5 seconds can increase the average delay experienced by all vehicles arriving at the intersection over the next 5-13 minutes by over 2.0%.

Analysis with $\Delta = 4.0$

We conduct a similar experiment with $\Delta = 4.0$. Once again, recalling our earlier definition of Δ, four vehicles arrive along the south movement at the instant that the first north/south phase is about to end.

In Figure 7.5, we present the results of our model, run for both control strategies, across a range of equilibrium cycle durations in the interval [20, 150]. For each equilibrium cycle duration, we present the percentage change in average delay for each of the two control strategies. (As above, negative percentages indicate that Critical Movement Actuated Control outperforms Fully Actuated Control).

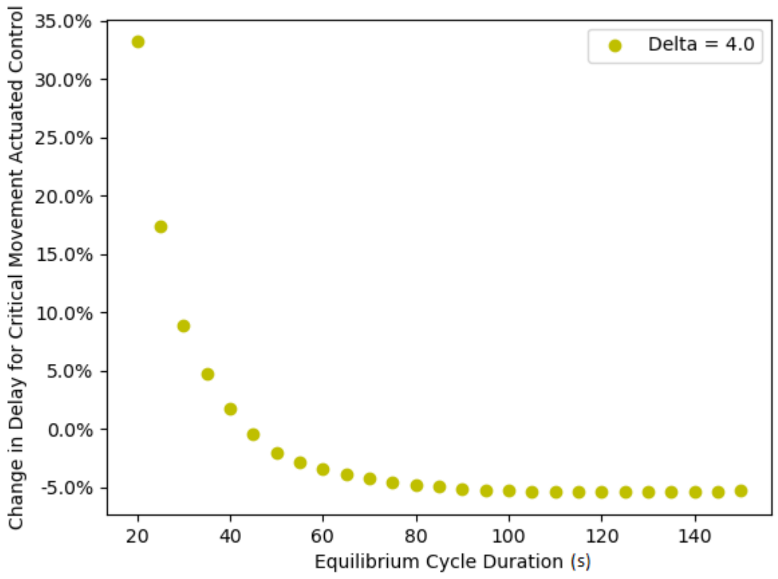


Figure 7.5: Change in Delay for Critical Movement Actuated Control at Various Equilibrium Cycle Durations when $\Delta = 4.0$

Our results are very similar to the previous experiment. As above, we observe that as the equilibrium cycle duration increases, the relative performance of Critical Movement Actuated Control (when compared against Fully Actuated Control) improves. However, the relative differences in delay between the two control strategies are larger in this experiment. For equilibrium cycle durations that are 90 seconds and above, Critical Movement Actuated Control offers a reduction in delay exceeding 5% when compared against Fully Actuated Control. A single decision to extend a phase by 10 seconds can increase the average delay of all vehicles arriving at the intersection over the next 7 - 13 minutes by over 5%.

7.2.6 Discussion

Phase Maximum Times

Our model does not consider phase maximum times; here, we discuss what may occur if our model were to incorporate phase maximum times. First, let us assume that the phase maximum times are at least as large as the equilibrium phase duration (P_e). Without this assumption, our intersection is over-capacitated.

Recall that when $\Delta > 0$, Critical Movement Actuated Control does not increase the phase duration over the equilibrium value, so the addition of phase maximum times would not influence the resulting delay for Critical Movement Actuated Control.

With Fully Actuated Control, during the initial phase, after the $\Delta > 0$ vehicles arrive, the phase maximum time could limit the amount of time that the phase can serve these vehicles. The unserved vehicles along the south movement would then be served during the next north/south phase. By limiting the number of vehicles served, a phase maximum time would force Fully Actuated Control to act more like Critical Movement Actuated Control. At higher values of λ_C (those where Critical Movement Actuated Control outperforms Fully Actuated Control), Fully Actuated Control with a phase maximum time would perform better than Fully Actuated Control without a phase maximum time. The relative improvement of Critical Movement Actuated Control would be smaller. Lower phase maximum times (assuming that the phase maximum is still large enough to serve all vehicles along the critical movement), would lead to smaller differences in delay between the two actuated control strategies.

For schedule-driven control, larger phase maximum times provide the intersection with additional scheduling flexibility, but can result in the scheduler sub-optimally serving vehicles along a non-critical movement (when no vehicles are present along the complementary critical movement). This is one possible explanation for the unintuitive result presented by Hu and Smith (2020) that using longer phase maximum times with a modified version of Surtrac2012 is not ideal.

Model Conclusions

Baseline implementations of schedule-driven control methods, including both ERIS and Surtrac2012, do not differentiate between extending a phase for vehicles travelling along a critical movement and extending a phase for vehicles travelling along a non-critical movement. This analysis has demonstrated (albeit for a different control framework) that a single decision to extend a phase or end a phase can lead to differences in average delay of over 5% for vehicles using the intersection over the next 7 - 13 minutes. Specifically, when demand and the equilibrium cycle duration are low, extending a phase to serve vehicles along a non-critical movement is a better decision. When demand and the equilibrium cycle duration are high, ending a phase and thus delaying vehicles along a non-critical movement is a better decision.

Our analysis considered a single decision point where no critical vehicles were present while non-critical vehicles were present. Schedule-driven control strategies face analogous decisions when deciding whether to extend or end a phase for non-critical vehicles. There will be situations where the critical movement has cleared but several vehicles remain on the non-critical movement (possibly because the initial non-critical movement queue was longer; possibly because several vehicles have just arrived along the non-critical movement). Modifying the decision rules for schedule-driven control strategies to favor critical movements when the intersection is busy has the potential to reduce average delay. We demonstrated this in Chapter 6 on the Eight-Movement Intersection by downgrading the non-critical movements to prioritize the critical movements. In the next section, we continue this exploration and examine prioritizing critical movements for the Halton Hills network.

7.3 Experimental Analysis on Halton Hills Network

In this section, we examine weighting the movements of the busiest intersection of the Halton Hills network. At high levels of demand, we observe a statistically significant reduction in delay from downgrading the non-critical movements.

We provide a brief overview of the network and then present our experiment.

7.3.1 Network Overview

The Halton Hills network is in Halton Hills, Ontario, Canada, roughly 30 miles west of Toronto. The network was provided by Rapid Flow Technologies who has deployed an updated version of Surtrac along the network. This is a corridor network with 3 intersections along the corridor (Steeles Avenue). The southwestern-most intersection of the network, Steeles Avenue and Trafalgar Road, is the busiest intersection in the network as this intersection is adjacent to a freeway exit. As such, we will focus on weighting this intersection. A graphic of the full network is presented in Figure 7.6 and a graphic of the Steeles Avenue and Trafalgar Road intersection, which shows the number of lanes and provided demand for each movement, is presented in Figure 7.7. While the network runs from the southwest to northeast in the real-world, in our figures and further discussion, we rotate the intersection 60° clockwise and model it as running from west to east.

We have based our model of this network on the provided model from Rapid Flow Technologies. Similarly, we obtained measured demand information from Rapid Flow Technologies for the AM rush hour. As with the St. Albert network studied earlier in the thesis, the provided demand information did not match with qualitative descriptions of how busy the network was during the AM rush hour. Specifically, provided demand was roughly 1100 vehicles per hour across the entire network during the AM rush hour. The busiest intersection in the network has 17 inbound lanes. One can reasonably expect more than 18 vehicles to pass through this intersection every minute during the AM rush hour.

As in Chapter 3, we scaled up the provided demand. We were unable to obtain realistic videos of the network. (This analysis was conducted in April 2020 and due to the COVID-19 pandemic, traffic had greatly fallen; as of September 2020, when this chapter was written, demand remained reduced). Thus, we examined a range of possible demand scaling factors. Qualitatively, using a scaling factor of 3.25 seemed to produce roughly equivalent behavior to using a scaling factor of 1.43 for the St. Albert network. Specifically, with these scaling factors, we noticed that at the busiest intersection in each network, queues along the busiest movement were approximately 25 vehicles long and roughly one out of every four cycles, a phase would not serve all of its

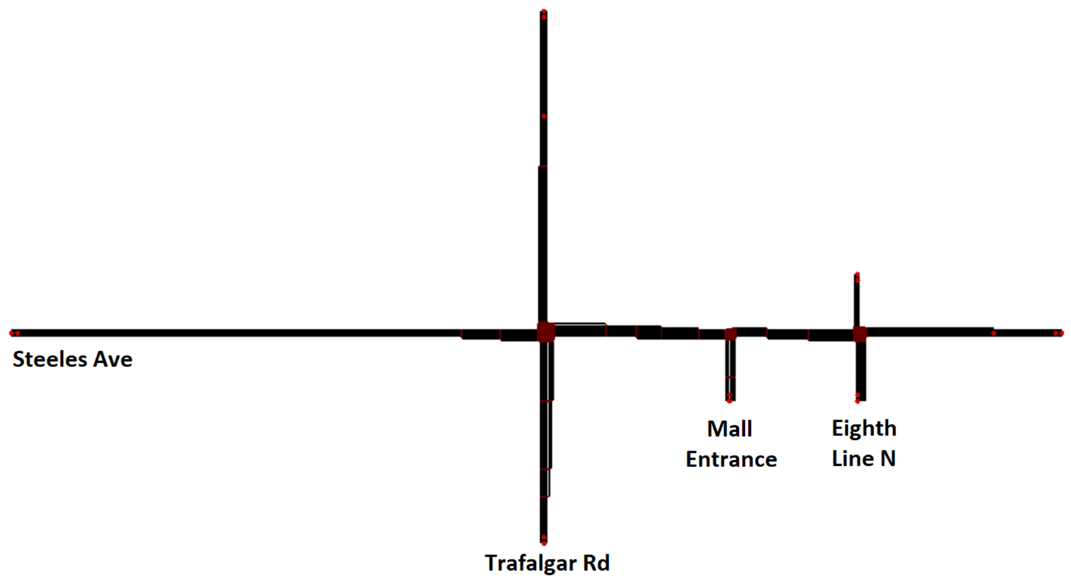


Figure 7.6: Halton Hills Network

vehicles. Using a scaling factor above 3.5 led to the network becoming overcapacitated, so we avoided scaling factors above 3.5. In our experiments, we consider a range of scaling factors between 1.0 and 3.5.

At higher scaling factors in this range, we observed a larger variance in delay as some simulation runs had queue build-up such that phases were unable to serve all vehicles present while other simulation runs did not. As such, we ran 20 trials for each combination of weighting strategy and demand level.

7.3.2 Calculation of Critical Movements

Using the critical movement calculation procedure discussed in Section 6.6 and the provided demand information, we calculated the critical movements of this intersection. Note that to compress the size of the score calculation, we only present the busier of right/straight for each of the origins with non-overlapping right and straight lanes (east, west, and south origins).

1. East left (80) + West right (166) + North straight/right (77.5) + South left (27.5) = 351
2. East left (80) + West right (166) + North left (21) + South right (70) = 337
3. East straight (46) + West left (8) + North straight/right (77.5) + South left (27.5) = 159
4. East straight (46) + West left (8) + North left (21) + South right (70) = 145

Our critical movements are thus the east left movement, west right movement, north straight/right movement, and the south left movement. The demands from these movements are surrounded by a green box in Figure 7.7

In our experiments, these movements will remain unweighted while the non-critical movements will be downgraded by a factor of 0.5. (0.5 was chosen arbitrarily as it is half-way between 0 and 1. We did not run a full parameter sweep as we did in Chapter 6).

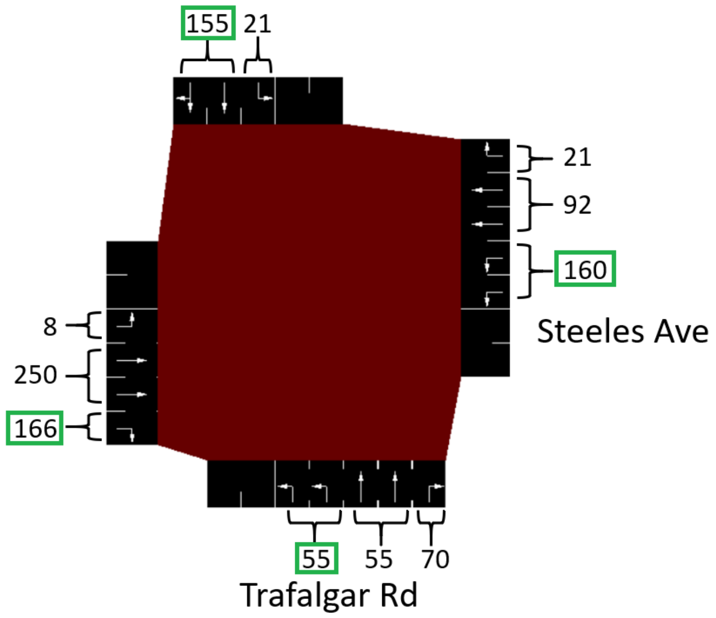


Figure 7.7: Intersection at Steeles Avenue and Trafalgar Road with Provided Demand

7.3.3 Experimental Evaluation

We present average vehicle delay in Table 7.13 and Table 7.14. Table 7.13 presents the results at low and medium levels of demand (which we classify as 3300 vehicles per hour and below). At levels of demand between 1100 and 3025 vehicles/hour, there is an average of a 1% increase in delay as a result of our weighting strategy. We believe this small increase is not simply noise since for all of the comparisons at these levels of demand, baseline ERIS (i.e. the no weighting strategy) outperformed the weighting strategy. Additionally, 3 of these 8 differences are statistically significant when performing two-sided paired t-tests with a 95% confidence level (the differences at demand scaling factors of 1.50, 2.00, and 2.25 are statistically significant).

Demand Scaling Factor	1.00	1.25	1.50	1.75	2.00	2.25	2.50	2.75	3.00
Hourly Demand	1100	1375	1650	1925	2200	2475	2750	3025	3300
No Weighting (Baseline)	37.5	40.7	44.2	47.7	51.2	54.7	59.3	65.8	75.6
Downgrading Non-critical Movements	38.0	41.1	45.2	48.1	51.7	55.4	59.4	66.1	75.3
vs Baseline	1.3%	1.1%	2.3%	0.7%	1.1%	1.2%	0.2%	0.4%	-0.4%

Table 7.13: Average Vehicle Delay (s) across Different Weighting Strategies on the Halton Hills Network - Demands 3300 veh/hr and below

Table 7.14 presents the results at high levels of demand. At high levels of demand, downgrading the non-critical movements leads to a reduction in delay. When performing two-sided paired t-tests with a 95% confidence level with the Holm–Bonferroni correction, each of the 7 comparisons are statistically significant.

Demand Scaling Factor	3.20	3.25	3.30	3.35	3.40	3.45	3.50
Hourly Demand	3520	3575	3630	3685	3740	3795	3850
No Weighting (Baseline)	93.6	102.6	109.1	116.0	145.2	170.8	172.3
Downgrading Non-critical Movements	90.2	96.0	104.3	105.2	129.7	143.8	161.8
vs Baseline	-3.7%	-6.4%	-4.4%	-9.3%	-10.7%	-15.8%	-6.1%

Table 7.14: Average Vehicle Delay (s) across Different Weighting Strategies on the Halton Hills Network - Demands 3520 veh/hr and above

As discussed in Chapter 3, when optimizing one criteria, one should consider whether another criteria may be worsened. For this reason, we also examine the average number of times that each vehicle stops. The average number of stops at high levels of demand are presented in Table 7.15.

Demand Scaling Factor	3.20	3.25	3.30	3.35	3.40	3.45	3.50
Hourly Demand	3520	3575	3630	3685	3740	3795	3850
No Weighting (Baseline)	1.42	1.55	1.67	1.77	2.27	2.72	2.81
Downgrading Non-critical Movements	1.38	1.44	1.59	1.57	1.93	2.17	2.51
vs Baseline	-2.5%	-7.1%	-4.7%	-10.9%	-14.9%	-20.0%	-10.9%

Table 7.15: Average Number of Stops across Different Weighting Strategies on the Halton Hills Network - Demands 3520 veh/hr and above

We observe that downgrading the non-critical movements also leads to a reduction in the average number of times that a vehicle stops. When performing two-sided paired t-tests with a 95% confidence level with the Holm–Bonferroni correction, at the four highest levels of demand, the reductions in the number of stops are statistically significant. This result allows us to conclude that there is no trade-off between average vehicle delay and the average number of stops; rather, average vehicle delay and the average number of stops are correlated.

This experiment demonstrates that downgrading the non-critical movements can offer a significant reduction in both average vehicle delay and the average number of stops when the intersection is running at or near its capacity. Additionally, the results of these experiments support the theoretical model presented in the previous section that prioritizing the critical movements is detrimental at low demands but beneficial at high demands.

7.4 Additional Examination of the Intersection at Steeles Avenue and Trafalgar Road

In the previous section, we demonstrated that downgrading the non-critical movements can reduce average vehicle delay on the Halton Hills network. In this section, we examine the busiest intersection of this network, the intersection at Steeles Avenue and Trafalgar Road, in more detail to determine if other weighting strategies can also be useful. Specifically, in this section, we examine only this intersection (as opposed to the full network). We also generate synthetic demand (as opposed to using the provided demand as we did in the previous section).

In contrast to the Eight-Movement Intersection presented in Chapter 6 where each movement had one lane, the different movements of this intersection do not all have the same number of lanes. Recalling the earlier experiment with the 2x1 Intersection, we may expect other weighting strategies based on lane counts to provide an improvement.

After detailing additional information relating to the setup of this evaluation, we will briefly explain our high-level search process to determine the two most-promising strategies. We then present the results of our validation experiment and discuss these results.

7.4.1 Experimental Setup

For this evaluation, we generate synthetic demand. We specify that each lane along a critical movement will have approximately 400 vehicles per hour. We vary the demand for the non-critical movements; demand for these movements will vary from 50% of the critical demand to 100% of the critical demand. As in the previous section, our critical movements will be the following movement groups: north straight/right, east left, south left, and west right.

For this section, when considering weighting strategies, we will combine right and straight movements into one movement group and treat both as either critical or non-critical. Since we plan to consider all combinations of weight/no-weight as we did in Chapter 6, this simplification allows us to restrict our search space of possible weighting strategies to $2^8 - 1 = 255$. (As mentioned in the previous section, straight and right movements share a lane along the north origin, but not along the other three origins; thus, our exponent would increase by three if we were to individually weight the right and straight movements for these origins. Without this simplification, there would be $2^{11} - 1 = 2047$ possible weighting strategies).

7.4.2 Parameter Sweep

We begin by running a parameter sweep for all 255 possible weight/no-weight combinations, as we did in Section 6.7.3, across three different demand settings (high congestion where non-critical movements have about 70% of the per lane demand of critical movements, high congestion where non-critical movements have about 50% of the per lane demand of critical movements, and medium congestion where non-critical movements have about 50% of the per lane demand of critical movements). Recall that when a movement is selected to receive a weight, it receives a weight below 1, downgrading the importance of the movement.

For each demand setting and weight/no-weight combination, we run four 30-minute trials. For each trial, we calculate average vehicle delay for vehicles arriving between the 6-minute mark and the 27-minute mark.

After running our initial parameter sweep, we select what we consider the 16 best weighting combinations to further explore. Specifically, we select combinations that either provided one of the lowest delays on a single demand or generally performed well across multiple demands.

We then run additional trials evaluating these 16 selected strategies. We modify our demand settings to solely consider high congestion inputs (weighting appeared most-impactful during high congestion) and varied demand along the non-critical movements to be between 50% and 100% of the critical movement demand. For each demand setting and weight/no-weight combination, we run eight 30-minute trials.

We identify the four most-promising strategies:

1. Downgrading the four non-critical movements. (This is the same strategy we have been discussing throughout this chapter and Chapter 6).
2. Downgrading the east straight/right and south straight/right movements. These movements have three lanes and (according to the fixed phase ordering of the dual ring barrier controller) immediately follow a left turn movement with one lane (west left and north left, respectively). Since more vehicles travel along the three-lane movements, baseline ERIS' objective function prioritizes serving these movements. The straight movements with three lanes "overpower" the single-lane left turn movements, so ERIS switches out of left turn movements too quickly.
3. Downgrading all movements except for west left and north left. Similar to the above strategy, this ensures that these one-lane left turn movements are not overpowered.
4. Downgrading the four straight/right movements. Similar to the two above strategies, this discourages the straight movements from overpowering the left turn movements.

Finally, we examined each of these methods with five possible weights (0.3, 0.4, 0.5, 0.6, 0.7) and using the same demand settings as above, ran six 30-minute trials for each combination. From these experiments, we reduced the set of strategies down to two combinations that appeared most-promising:

1. Downgrading the four non-critical movements with a weight of 0.3.
2. Downgrading the east straight/right and south straight/right movements with a weight of 0.4.

In the next subsection, we present the results of validation experiments for both of these strategies.

7.4.3 Validation Experiment

For each weighting strategy and demand combination, we run 20 one-hour trials. (For the validation experiment, we run more trials and longer trials than in the parameter sweeps to reduce the impact of noise). We calculate the average delay for vehicles arriving in the middle 40 minutes of each trial.

We present average vehicle delay in Table 7.16. We refer to the strategy of downgrading the east straight/right and south straight/right movements as "Left Protect" in Table 7.16 and the remainder of this section.

At all levels of congestion, both weighting strategies outperform the baseline. At the lowest scaling factor (0.6), downgrading the non-critical movements is the better of the two methods. At the next two scaling factors (0.7 and 0.8), Left Protect is slightly better than downgrading the non-critical movements. Left Protect is also better at the highest scaling factors (0.9 and 1.0) and this improvement is larger.

We run two-sided paired t-tests with a 95% confidence level with the Holm-Bonferroni correction for the 10 comparisons against the baseline (i.e. Left Protect vs the baseline at each of the five scaling factors and Downgrade Non-Critical Movements vs the baseline at each of the five scaling factors). We find two statistically significant differences: Left Protect is statistically significantly better than the baseline at a demand scaling factor of 0.9 and Left Protect is also

Non-critical Demand Scaling Factor	0.6	0.7	0.8	0.9	1.0
Non-critical Vehs/Lane/Hour	240	280	320	360	400
No Weighting (Baseline)	70.3	69.5	80.2	117.0	185.6
Downgrade Non-Critical Movements	65.5	67.8	75.5	90.3	146.9
Left Protect	67.4	67.0	74.8	86.1	136.2
Downgrade Non-Crits vs Baseline	-6.9%	-2.5%	-5.8%	-22.8%	-20.9%
Left Protect vs Baseline	-4.1%	-3.6%	-6.7%	-26.4%	-26.6%
Downgrade Non-Crits vs Left Protect	-2.8%	1.2%	0.9%	4.9%	7.8%

Table 7.16: Average Vehicle Delay (s) across Different Weighting Strategies for Intersection at Steeles Avenue and Trafalgar Road

statistically significantly better than the baseline at a demand scaling factor of 1.0. (If we were to apply a more lenient statistical cut-off and not apply the Holm-Bonferroni correction, 5 of the 10 comparisons would be statistically significant).

At demand scaling factors of 0.9 and 1.0, we notice long queues forming along the north left and west left movements when running baseline ERIS. As discussed in the previous subsection, baseline ERIS does not keep the north left and west left movements active for very long because it prefers to serve vehicles along the south straight/right and east straight/right movements which have three lanes (as opposed to one lane).

The Left Protect weighting strategy downgrades the importance of these straight movements, thereby making the left turn movements relatively more attractive to ERIS. These weighting modifications which encourage ERIS to spend more time serving these left turn movements successfully prevent vehicles from needing to wait multiple cycles along these movements.

We caution reading too much into the reduction at scaling factors of 0.9 and 1.0 for the strategy which downgrades the non-critical movements. Specifically, at a demand scaling factor of 1.0, there are not any critical movements. For this network, it happens that two of the four movements that this strategy downgrades are the two movements downgraded by Left Protect. At these higher levels of demand, we consider this strategy to be a weaker version of Left Protect that works because it happens to downgrade the same movements as Left Protect.

7.4.4 Discussion

This experiment serves as a good example for two situations in which one may wish to have ERIS weight movements.

First, one may wish to downgrade non-critical movements to encourage ERIS to focus on reducing the cycle duration. In this experiment, we observed that when the demand on the non-critical lanes was 60% of the demand on the critical movements, downgrading the non-critical movements led to a decrease in delay. We noticed a similar result for our Eight-Movement Intersection from Chapter 6 and from our theoretical model.

Second, one may wish to downgrade movements that overpower other movements. For this experiment, we saw that ERIS tended to favor straight movements with three lanes to the detriment of left turn movements with a single lane. We observed a similar effect with our 2x1 Intersection from Chapter 6 where ERIS prioritized the movement with two lanes over the movement with one lane. In both of these situations, when demand was relatively uniform for each lane, the movement with more lanes had a larger impact on the objective function. While serving multiple vehicles at a time is generally better than serving a single vehicle at a time, the baseline ERIS scheduler focuses too much on serving these movements, and as a result, commits too little time towards serving single-lane movements. This leads to a large queue build-up where vehicles along these movements must wait for several cycles for their turn to pass through the intersection, ultimately increasing delay. By downgrading the overpowering lanes, we encourage ERIS to spend more time serving other movements and ultimately reduce congestion.

7.5 Summary of when to Consider Weighting

We present a brief summary of our thought process for when one may want to apply weighting. One may consider the following questions:

1. Is the network near or at its capacity?
2. Does the network have clearly defined critical movements?
3. Are there any movements for which we observe long queues?

In our experiments, we observed that weighting is beneficial when the network is running at or near its capacity. Our queuing model demonstrated that at higher demands, an intersection requires additional time to return to its equilibrium cycle duration and the increase in cycle duration leads to a relatively larger increase in delay for vehicles arriving during the next several cycles.

Prioritizing the critical movements (by downgrading the non-critical movements) is more beneficial when an intersection has clearly defined critical movements. As shown in our model, prioritizing the critical movements for a busy intersection allows the intersection to reduce the average cycle duration to ultimately reduce the total average delay. This requires the intersection to delay vehicles travelling along the non-critical movements, but the increase in delay for these vehicles is less than the decrease in delay from keeping the cycle duration lower.

We observed from the Eight-Movement Intersection experiments presented in Chapter 6 that downgrading the non-critical movements leads to a decrease in delay. Similarly, in our most recent experiment examining the intersection at Steeles Avenue and Trafalgar Road, we observed that when the critical movement demand per lane was much higher than the non-critical movement demand per lane (i.e. at a demand scaling factor of 0.6), downgrading the non-critical movements leads to a decrease in delay compared to the baseline and the other examined weighting strategy.

Finally, when examining the intersection at Steeles Avenue and Trafalgar Road when the relative demands on all movements were relatively similar (i.e. at demand scaling factors of 0.9 and 1.0), we observed that baseline ERIS allows long queues to form along a subset of the movements. ERIS favors movements with more lanes which serve more vehicles and have a higher impact in the objective calculation. By downgrading the importance of the multi-lane movements that compete with the single-lane left turn movements, ERIS becomes more willing to serve these left turn movements.

Note that the act of overpowering does not necessarily take the form of a straight movement overpowering the left turn movement on the same ring. In our experiment on the 2x1 Intersection, we observed that a straight movement with two lanes overpowered a straight movement with one lane. As in the prior experiment, downgrading the importance of the overpowering movement (the two-lane movement) can lead to a reduction in delay.

The above discussion describes some concepts to consider when deciding how to apply weighting to an intersection. However, if these strategies do not appear applicable, one can run a parameter sweep as we have performed both in this chapter and Chapter 6 to determine if there are other classes of weighting strategies that may prove beneficial.

7.6 Conclusion

In Chapter 6, we presented three extensions to improve ERIS and decided that weighting movements was the most promising of our three extensions. In this chapter, we examined weighting movements in additional detail. We began by presenting a deterministic queuing model to confirm our experimental observations and provide theoretical evidence that prioritizing critical movements can be beneficial for reducing delay. Our model demonstrated that a single decision to extend a phase to serve four vehicles travelling along a non-critical movement could lead to an increase in delay for all vehicles travelling along the network over the next $7 - 13$ minutes by as much as 5%.

We presented an experiment where we prioritized critical movements along the Halton Hills network. Similar to the results of the queuing model, we noticed that prioritizing critical movements led to a decrease in delay at higher levels of demand. We then examined the busiest intersection of this network and discussed two beneficial weighting strategies. When demand is high and there are clearly defined critical movements, downgrading the non-critical movements (to prioritize the critical movements) reduces delay when compared against our baseline. On the other hand, when the demand across different movements is relatively equal, we noticed that ERIS tends to favor multi-lane movements while starving competing single-lane movements. By downgrading the importance of multi-lane movements, we can encourage ERIS to more efficiently serve vehicles along these multi-lane movements, ultimately providing more time to serve vehicles along single-lane movements and reducing delay.

Finally, we presented three questions one should consider when deciding whether to augment their scheduling algorithm with weighting. We suggested that weighting strategies are most beneficial when demand is high. Prioritizing the critical movements is beneficial when clearly defined critical movements exist. When one observes a network and notices that long queues form along some movements, downgrading the importance of opposing movements may prove beneficial.

While our experimentation solely examined improving the ERIS scheduler with weighting, we suspect that this extension can an offer an improvement to other schedule-driven methods as well. Schedule-driven methods typically do not differentiate between serving vehicles along a critical movement and serving vehicles along a non-critical movement. A relatively simple change in a scheduler's objective function to prioritize certain movements has the potential to reduce delay by 10% - 20% for other schedule-driven control methods.

7.7 Appendix: Additional Model Calculations

In this appendix, we present additional model calculations that were not presented in Section 7.2.2. We detail how our model calculates the number of vehicles served and average delay.

7.7.1 Additional Formulation, $\Delta = 0$

When $\Delta = 0$, the number of vehicles served along each of the critical and non-critical movements (denoted crit-served and non-crit-served, respectively) is the sum of the initial queue, as well as vehicles that arrive during the phase.

$\text{crit-served}_i = \text{crit-init-queue}_i + \text{phase-dur}_i * \lambda_C$
$\text{non-crit-served}_i = \text{non-crit-init-queue}_i + \text{phase-dur}_i * \lambda_N$

Additionally, at this time, the model calculates an intermediate value, the number of vehicles served along the non-critical movement when it empties (denoted non-crit-served-before-empty) which will be useful when calculating delay:

$\text{non-crit-served-before-empty}_i = \text{non-crit-init-queue}_i + \text{non-crit-empty-dur}_i * \lambda_N$

Next, the model uses the phase duration and number of vehicles served to to calculate the total delay for both the vehicles along the critical movement and the vehicles along the non-critical movement.

The model calculates the average delay for the vehicles along each movement by first calculating the average arrival time and the average departure time. For the critical movement, vehicles arrive uniformly between the time that the phase last ended and the time that the queue empties. Vehicles depart uniformly between the time that the phase began and the time that the queue empties. The average delay for vehicles travelling along the critical movement (denoted crit-av-delay) is the difference between the average critical vehicle departure time (denoted crit-av-dep) and the average critical vehicle arrival time (denoted crit-av-arr). (Recall that unless specified, we use the end time of the previous matching phase as the zero-reference point to simplify our average delay calculations).

$$\text{crit-av-arr}_i = \frac{(0) + (\text{phase-queuing-dur}_i + \text{crit-empty-dur}_i)}{2}$$

$$\text{crit-av-dep}_i = \frac{(\text{phase-queuing-dur}_i) + (\text{phase-queuing-dur}_i + \text{crit-empty-dur}_i)}{2}$$

$$\text{crit-av-delay}_i = \text{crit-av-dep}_i - \text{crit-av-arr}_i = \frac{\text{phase-queuing-dur}_i}{2}$$

The total delay for these vehicles (denoted crit-tot-delay) is the product of the average delay and the number of vehicles served along the critical movement:

$$\text{crit-tot-delay}_i = \frac{\text{phase-queuing-dur}_i}{2} * \text{crit-served}_i$$

Calculating the delay for the non-critical movement is slightly more complicated. First, recall that the queue along the non-critical movement will empty before the queue along the critical movement. Vehicles arriving along the non-critical movement once the queue has emptied are immediately served and do not experience delay. Our model will multiply the number of vehicles delayed along the non-critical movement (which we calculated above) by their average delay to obtain the total delay along the non-critical movement.

Delayed vehicles arrive uniformly between the time that the phase last ended and the time that the non-critical queue empties. Vehicles depart uniformly between the time that the phase began and the time that the non-critical queue empties. The average delay (denoted non-crit-av-delay) is the difference between the average non-critical vehicle departure time (denoted non-crit-av-dep) and the average non-critical vehicle arrival time (denoted non-crit-av-arr):

$$\text{non-crit-av-arr}_i = \frac{(0) + (\text{phase-queuing-dur}_i + \text{non-crit-empty-dur}_i)}{2}$$

$$\text{non-crit-av-dep}_i = \frac{(\text{phase-queuing-dur}_i) + (\text{phase-queuing-dur}_i + \text{non-crit-empty-dur}_i)}{2}$$

$$\text{non-crit-av-delay}_i = \text{non-crit-av-dep}_i - \text{non-crit-av-arr}_i = \frac{\text{phase-queuing-dur}_i}{2}$$

The total delay for these vehicles (denoted non-crit-tot-delay) is the product of the average delay and the number of vehicles that were delayed:

$$\text{non-crit-tot-delay}_i = \frac{\text{phase-queuing-dur}_i}{2} * \text{non-crit-served-before-empty}_i$$

7.7.2 Additional Formulation, $\Delta > 0$, Fully Actuated Control

For Fully Actuated Control, when $\Delta > 0$, our model uses the same equation for the total number of vehicles arriving along the critical movement (using the new value for phase duration). However, the model updates the number of vehicles served along the non-critical movement during the initial phase to account for the Δ vehicles.

$$\text{non-crit-served}_{0,FAC} = \text{non-crit-init-queue}_0 + \text{phase-dur}_{0,FAC} * \lambda_N + \Delta$$

Our model employs a different formulation to calculate delay for the initial phase of our model. We begin by presenting the modified delay calculation for the critical (north) movement during this initial phase.

When the Δ vehicles arrive, the north queue remains empty and vehicles arriving along the north movement are instantaneously served by the intersection, and thus experience no delay. Thus, our model will calculate the average delay only for the vehicles along the north movement that arrive prior to this point in time and multiply by the number of such vehicles to obtain total delay. First, recall from earlier the amount of time required to clear the critical movement queue.

$$\text{crit-empty-dur}_0 = \frac{\text{crit-init-queue}_0}{S - \lambda_C}$$

The number of critical vehicles served before the critical queue empties (denoted crit-served-before-empty) is thus:

$$\text{crit-served-before-empty}_{0,FAC} = \text{crit-init-queue}_0 + \text{crit-empty-dur}_0 * \lambda_C$$

The model calculates the average delay for vehicles along the critical movement arriving before the critical queue empties (denoted crit-av-delay-before-empty) as the difference between their average departure and arrival times (denoted crit-av-dep-before-empty and crit-av-arr-before-empty, respectively).

$$\text{crit-av-arr-before-empty}_{0,FAC} = \frac{(0) + (\text{phase-queuing-dur}_0 + \text{crit-empty-dur}_0)}{2}$$

$$\text{crit-av-dep-before-empty}_{0,FAC} = \frac{(\text{phase-queuing-dur}_0) + (\text{phase-queuing-dur}_0 + \text{crit-empty-dur}_0)}{2}$$

$$\text{crit-av-delay-before-empty}_{0,FAC} = \text{crit-av-dep-before-empty}_{0,FAC} - \text{crit-av-arr-before-empty}_{0,FAC} = \frac{\text{phase-queuing-dur}_0}{2}$$

This is the same average delay as calculated above. However, here, we only multiply by the number of vehicles arriving before the critical queue empties to obtain the total delay for critical vehicles:

$$\text{crit-tot-delay}_{0,FAC} = \frac{\text{phase-queuing-dur}_0}{2} * \text{crit-served-before-empty}_{0,FAC}$$

We now present the formula for delay for the non-critical vehicles arriving during the initial phase. Observe that the non-critical queue empties twice: first for the initial queue and then a second time for the queue created by the Δ vehicles. We calculate the total delay for each group separately and then combine them to obtain the total delay for the non-critical movement.

For the first group, vehicles arrive uniformly between the time that the phase last ended and the time that the non-critical queue first empties (which we defined earlier as non-crit-empty-dur$_i$). Vehicles depart uniformly between the time that the phase began and the time that the non-critical queue first empties. The average delay (denoted group-1-av-delay) for these vehicles is the difference between their average departure and arrival times (denoted group-1-av-dep and group-1-av-arr, respectively).

$$\text{group-1-av-arr}_{0,FAC} = \frac{(0) + (\text{phase-queuing-dur}_0 + \text{non-crit-empty-dur}_0)}{2}$$

$$\text{group-1-av-dep}_{0,FAC} = \frac{(\text{phase-queuing-dur}_0) + (\text{phase-queuing-dur}_0 + \text{non-crit-empty-dur}_0)}{2}$$

$$\text{group-1-av-delay}_{0,FAC} = \text{group-1-av-dep}_{0,FAC} - \text{group-1-av-arr}_{0,FAC} = \frac{\text{phase-queuing-dur}_0}{2}$$

The total delay for these vehicles (denoted group-1-tot-delay) is the product of the average delay and the number of vehicles that were delayed:

$$\text{group-1-tot-delay}_{0,FAC} = \frac{\text{phase-queuing-dur}_0}{2} * \text{non-crit-served-before-empty}_0$$

We must also calculate the delay for vehicles in the second group of delayed vehicles; as before, we will calculate average arrival time and average departure time to calculate average delay. We will then multiply this value by the number of vehicles in the second group of delayed vehicles. (Variable names for averages and totals will be denoted as above, but using “group-2” instead of “group-1” in the variable names).

Recall that

$$\Delta\text{-dur}_{0,FAC} = \frac{\Delta}{S - \lambda_N}$$

Using known arrival rates, our model calculates the number of vehicles in group 2 (denoted group-2-vehs):

$$\text{group-2-vehs}_{0,FAC} = \frac{\lambda_N * \Delta}{S - \lambda_N} + \Delta$$

Using the point at which the Δ vehicles arrive as the zero-reference point, we note that the vehicles depart uniformly between 0 and $\Delta\text{-dur}_{0,FAC}$. The average departure time is thus:

$$\text{group-2-av-dep}_{0,FAC} = \frac{\Delta\text{-dur}_{0,FAC}}{2}$$

Δ vehicles arrive at reference time 0. The remaining $\frac{\lambda_N * \Delta}{S - \lambda_N}$ vehicles arrive uniformly between 0 and $\Delta\text{-dur}_{0,FAC}$. The average arrival time is thus:

$$\text{group-2-av-arr}_{0,FAC} = \left(0 * \Delta + \frac{\Delta\text{-dur}_{0,FAC}}{2} * \frac{\lambda_N * \Delta}{S - \lambda_N}\right) \Big/ \left(\text{group-2-vehs}_{0,FAC}\right)$$

Taking the difference provides the average delay for a vehicle in group 2:

group-2-av-delay$_{0,FAC}$ $=$ group-2-av-dep$_{0,FAC}$ $-$ group-2-av-arr$_{0,FAC}$

The total delay for these vehicles is thus:

group-2-tot-delay$_{0,FAC}$ $=$ group-2-av-delay$_{0,FAC}$ $*$ group-2-vehs$_{0,FAC}$

It follows that:

non-crit-tot-delay$_{0,FAC}$ $=$ group-1-tot-delay$_{0,FAC}$ $+$ group-2-tot-delay$_{0,FAC}$

For all phases after the initial phase, delay calculations are unchanged from the original example.

7.7.3 Additional Formulation, $\Delta > 0$, Critical Movement Actuated Control

For Critical Movement Actuated Control, when $\Delta > 0$, our model updates the formula for the number of vehicles served along the south movement for the second and later south phases by checking that there is sufficient time to serve all vehicles along the south movement. The first term in the minimization is the sum of the vehicles queued along the south movement before the phase begins and the number of vehicles arriving along this movement during the phase. The second term is the maximum number of vehicles that can be served along the movement during the phase (due to capacity).

non-crit-served$_{i,CMAC}$ $=$ min(non-crit-init-queue$_{i,CMAC}$ $+$ phase-dur$_i$ $* \lambda_N, S *$ phase-dur$_i$)
for $i \geq 2$ and i is even

If the first term of the minimization is higher than the second term, there are unserved vehicles (denoted south-unserved) that must be remembered for the following north/south phase.

south-unserved$_{i,CMAC}$ $=$ max(non-crit-init-queue$_{i,CMAC}$ $+$ phase-dur$_i$ $* \lambda_N - S *$ phase-dur$_i$, 0)
for $i \geq 2$ and i is even

The delay calculations for the south phases are modified to account for vehicles that are not served during the first north/south phase that they are present. First, the model must keep track of the average arrival time of these vehicles. Arrival times are relative to the end of the first north/south phase that they are present. (This places arrival times in the same reference frame that will be used when calculating delays during the next north/south phase). For the Δ vehicles that arrive as the first phase ends, their average arrival time (denoted unserved-av-arr) is 0.

unserved-av-arr$_{0,CMAC}$ $= 0$

As mentioned above, there may be unserved vehicles along the south movement for future north/south phases as well. We must calculate the average arrival time of these vehicles for our delay calculation. (Our analysis assumes that the Δ vehicles are all served during the second north/south phase; this simplifies the calculation of the average arrival time for unserved vehicles). The average arrival time for the unserved vehicles is:

$$\text{unserved-av-arr}_{i,CMAC} = -\frac{\text{south-unserved}_{i,CMAC}}{2 * \lambda_N} \text{ for } i \geq 2 \text{ and } i \text{ is even}$$

Once we have these preliminary values, we can calculate the delay for the south phase. We break the calculation into two cases. In Case I, the south movement serves all vehicles (south-unserved$_{i,CMAC} = 0$). For Case II, the south movement does not serve all vehicles (south-unserved$_{i,CMAC} > 0$). We begin by presenting Case I.

For Case I, as before, we calculate the average arrival time and the average departure time and take their difference. Vehicles depart uniformly between the start of the phase and the time that the non-critical queue empties. Using the end point of the previous north/south phase as our zero-reference point:

$$\text{non-crit-av-dep}_{i,CMAC} = \frac{(\text{phase-queuing-dur}_i) + (\text{phase-queuing-dur}_i + \text{non-crit-empty-dur}_i)}{2}$$
$$\text{for } i \geq 2 \text{ and } i \text{ is even}$$

Calculating the average arrival time is more complicated. south-unserved$_{i-2,CMAC}$ vehicles arrive at an average time of unserved-av-arr$_{i-2,CMAC}$. The remaining vehicles arrive uniformly between 0 and the time that the non-critical queue empties. The average arrival time is thus:

$$\text{non-crit-av-arr}_{i,CMAC} = [\text{unserved-av-arr}_{i-2,CMAC} * \text{south-unserved}_{i-2,CMAC} +$$
$$\frac{(0) + (\text{phase-queuing-dur}_i + \text{non-crit-empty-dur}_i)}{2} *$$
$$(\text{non-crit-served-before-empty}_i - \text{south-unserved}_{i-2,CMAC})]/\text{non-crit-served-before-empty}_i$$
$$\text{for } i \geq 2 \text{ and } i \text{ is even}$$

The average delay for these vehicles is the difference between the average departure time and average arrival time:

$$\text{non-crit-av-delay}_{i,CMAC} = \text{non-crit-av-dep}_{i,CMAC} - \text{non-crit-av-arr}_{i,CMAC}$$
$$\text{for } i \geq 2 \text{ and } i \text{ is even}$$

Recall that not all non-critical vehicles will be delayed. The total delay is thus:

$$\text{non-crit-tot-delay}_{i,CMAC} = \text{non-crit-av-delay}_{i,CMAC} * \text{non-crit-served-before-empty}_{i,CMAC}$$
$$\text{for } i \geq 2 \text{ and } i \text{ is even}$$

For Case II, not all vehicles are served along the south movement. This further complicates our calculation. As before, the model calculates the average departure time, the average arrival time and takes the difference to calculate average delay. Since the queue does not empty, we know that vehicles depart for the entire duration of the phase. Thus, the average departure time (using the end point of the previous north/south phase as our zero-reference point) is:

$$\text{non-crit-av-dep}_{i,CMAC} = \frac{(\text{phase-queuing-dur}_i) + (\text{phase-queuing-dur}_i + \text{phase-dur}_i)}{2}$$
$$\text{for } i \geq 2 \text{ and } i \text{ is even}$$

Before calculating the average arrival time for the vehicles that are served, the model calculates the final served vehicle's arrival time (denoted final-arr-time):

$$\text{final-arr-time}_{i,CMAC} = \text{phase-queuing-dur}_i + \text{phase-dur}_i - \frac{\text{south-unserved}_{i,CMAC}}{\lambda_N}$$
$$\text{for } i \geq 2 \text{ and } i \text{ is even}$$

The model then calculates the average arrival time. south-unserved$_{i-2,CMAC}$ vehicles arrive at an average time of unserved-av-arr$_{i-2,CMAC}$. The remaining vehicles arrive uniformly between 0 and final-arr-time$_{i,CMAC}$.

$$\text{non-crit-av-arr}_{i,CMAC} = [\text{unserved-av-arr}_{i-2,CMAC} * \text{south-unserved}_{i-2,CMAC} + \frac{\text{final-arr-time}_{i,CMAC}}{2} * (\text{non-crit-served}_{i,CMAC} - \text{south-unserved}_{i-2,CMAC})] / \text{non-crit-served}_{i,CMAC}$$
$$\text{for } i \geq 2 \text{ and } i \text{ is even}$$

The average delay for these vehicles is:

$$\text{non-crit-av-delay}_{i,CMAC} = \text{non-crit-av-dep}_{i,CMAC} - \text{non-crit-av-arr}_{i,CMAC}$$
$$\text{for } i \geq 2 \text{ and } i \text{ is even}$$

It follows that the total delay is:

$$\text{non-crit-tot-delay}_{i,CMAC} = \text{non-crit-av-delay}_{i,CMAC} * \text{non-crit-served}_{i,CMAC}$$
$$\text{for } i \geq 2 \text{ and } i \text{ is even}$$

Chapter 8

Conclusion

In this thesis, we proposed several methods for adaptive traffic signal control. In this chapter, we first recall the challenges presented in Chapter 1. We summarize our contributions while noting how they address these challenges. We then propose several areas for future work.

8.1 Contribution Overview

Before overviewing our contributions, we recall the six challenges for real-time schedule-driven control that we presented in Chapter 1.

1. **Requirement of Real-time Solvability** - Schedule-driven methods typically make decisions every time step, which may be as often as every second. The time a model has to run its calculations is limited by the time step. Thus, schedule-driven methods must make modelling and optimization compromises to ensure that they can successfully return a solution within this time limit.
2. **Complexity of Traffic Dynamics** - If search states maintain information about all vehicles separately, search state updates will be too slow. Instead, schedule-driven methods, including the original Surtrac system, tend to group vehicles into clusters and usually do not allow splitting clusters [69]. As a result, there may be good schedules that split clusters that are not considered.
3. **Exponentially Large Search Space** - Comparing all possible signal timing strategies is exponentially large in the length of the scheduling horizon. Some methods require all steps to be exactly 5 seconds in duration, reducing the number of branching points of the search [24, 28, 56]. There may be good schedules with steps of different durations that are not considered.
4. **Global Coordination** - Scheduling more than one intersection at once tends to increase runtime exponentially. (For example, McCluskey and Vallati (2017) proposes a search method that requires roughly 30 seconds to generate a joint schedule for 7 intersections). As a result, many schedule-driven methods are decentralized, meaning the scheduler at each intersection optimizes a local objective. These local objectives do not always align with global objectives, and as a result, schedulers make sub-optimal decisions.

5. **Finite Sensing Horizon** - A scheduler only schedules the vehicles it has sensed through its detection system, which has limited range, or from shared information between intersections. We refer to this as the finite sensing horizon. Vehicles outside of the finite sensing horizon will continue to arrive at the intersection and most methods fail to consider the impact of their decisions on these vehicles.
6. **Inadequate Objective Function** - A scheduler that minimizes delay every time step will not necessarily minimize delay during the entire evaluation period (even if the evaluation period is as short as 5 minutes). A decision that minimizes delay over the scheduling horizon may cause queues to form along some movements, ultimately increasing delay during future time steps.

8.1.1 Part I: Expressive Real-time Intersection Scheduling

In the first part of this thesis, we introduced ERIS to address the first three challenges. ERIS' internal representation of traffic conditions models the vehicles along each lane of an intersection allowing ERIS to more accurately estimate delay than past schedule-driven control methods (Challenge #2). Employing an A* search with an efficient heuristic function calculation, ERIS is able to calculate schedules in real-time (Challenges #1 & #3).

In Chapter 3, we presented ERIS' search state formulation and discussed the heuristic function pipeline. We evaluated ERIS on the St. Albert network and demonstrated that ERIS offers an improvement when compared against both SynSurtrac2012 and a fixed timing plan. At the highest level of demand, ERIS reduced average vehicle delay by 10.9% when compared against SynSurtrac2012 and by 39.7% when compared against the fixed timing plan. We highlighted the four major model differences between ERIS and SynSurtrac2012 and found that ERIS' internal representation of traffic conditions that models the vehicles along each lane was the most important factor for reducing delay.

8.1.2 Part II: Green Wave Coordination

In the second part of this thesis, we introduced CARIC to address the fourth challenge. CARIC generates green wave coordination plans so that vehicles can travel through the network without stopping (Challenge #4).

In Chapter 4, we presented CARIC's green wave planning pipeline and discussed necessary modifications to ERIS. We evaluated CARIC on a corridor network, the Portland-Franklin network, and demonstrated that when compared to ERIS, CARIC reduces average vehicle delay by 7.2%, average fuel usage by 3.8%, and the average number of stops by 10.4%.

In Chapter 5, we first illustrated that CARIC is most beneficial on networks with intersections that are tightly spaced. We then presented Adapative-CARIC to demonstrate that adaptively triggering green wave coordination plans based on recent congestion outperforms always triggering green wave coordination plans and never triggering green wave coordination plans.

8.1.3 Part III: Addressing Inefficiencies of the ERIS Scheduler

In the third part of this thesis, we introduced three extensions to ERIS to address the fifth and sixth challenges. First, we modified ERIS to generate phantom vehicle clusters so that schedules can better predict their impact on vehicles outside of the sensing horizon (Challenge #5). Second, we adapted ERIS' objective function to incorporate a term for makespan to prioritize shorter schedules (Challenge #6). Third, we altered ERIS' objective function to prioritize the vehicles travelling along selected movements (also Challenge #6).

In Chapter 6, we detailed each of these extensions and the necessary alterations to ERIS. We evaluated these extensions on several intersections and observed the largest improvement when modifying ERIS' objective function to prioritize the vehicles travelling along selected movements. For the 2x1 Intersection, we observed that downgrading the importance of the two-lane movement could reduce delay by as much as 33% (when compared to baseline ERIS). For the Eight-Movement Intersection, we observed that downgrading the importance of the non-critical movements (i.e. prioritizing the critical movements) could reduce delay by 4.8%.

In Chapter 7, we first proposed a deterministic queuing model to support our earlier result that prioritizing the critical movements reduces delay. We found that under an actuated control framework, a single decision to extend a non-critical movement by 10 seconds could increase average vehicle delay for the vehicles arriving over the next 10 minutes by more than 5%. We then evaluated the weighting extension on the Halton Hills network and demonstrated that prioritizing the critical movements could reduce average vehicle delay by up to 15.8%. We modified the demand at the busiest intersection of the Halton Hills network and observed that downgrading the importance of two of the multi-lane movements could reduce average vehicle delay by up to 26.6%. We concluded this chapter by presenting several questions one should consider when selecting a weighting strategy.

8.2 Future Work

In this section, we propose several potential areas of future work.

8.2.1 Additional Model Granularity

As mentioned in Section 8.1.1, ERIS employs a lane-based framework allowing it to more accurately estimate delay than past schedule-driven control methods. We are curious whether further increasing model granularity can lead to more accurate delay estimations, better schedule selection, and ultimately, lower average vehicle delay. Additionally, we must ensure that such methods continue to run in real-time.

To increase model granularity, one could discard the clustering simplification and instead maintain a separate arrival time for every vehicle. This would allow one to calculate schedules that immediately terminate a phase after serving a selected vehicle.

Recalling Challenges #1 & #3, a schedule must still be calculated in real-time, but this increase in granularity would increase the size of our search space. One could explore whether further modifications to ERIS or a different algorithm could efficiently solve such a problem. We propose several possible ideas to improve ERIS.

First, one could consider modifications to the heuristic pipeline to improve the accuracy of estimates of the remaining delay. For example, the second-to-last step of our heuristic pipeline runs a relaxation that does not require change-over time between phases, causing it to greatly underestimate remaining delay in some cases.

Second, one could consider variants of A* search. Weighted A* is not guaranteed to return an optimal solution, but in practice, it often returns solutions that are very close to optimal (or even optimal). Alternatively, D* (or D* Lite) would allow ERIS to reuse computation from previous time steps.

Third, refactoring our implementation of ERIS may be sufficient to achieve speed-up. In our current implementation, each search state keeps track of the underlying partial schedule and ERIS' A* search copies the underlying partial schedule from parent to children search states. It is possible that simply keeping track of the parent state's identity and then walking backwards once a goal state is reached to find the completed schedule would be more efficient.

8.2.2 Model Selection

In Chapter 5, we presented Adaptive-CARIC which adaptively decided whether to run ERIS or to run CARIC. We demonstrated that Adaptive-CARIC offers an improvement when compared against both ERIS and CARIC.

One could consider generalizing this method to train a model which adaptively selects between any two (or more) scheduling methods. For a given network with its demand profile, it may be the case that no scheduling method is always the best strategy. A model that selects between multiple scheduling strategies has the potential to outperform individual strategies.

8.2.3 Additional Prioritization Strategies

In Chapters 4 & 5, we presented CARIC which prioritized vehicles travelling along the green wave route and reduced average vehicle delay. In Chapters 6 & 7, we demonstrated that prioritizing vehicles at an intersection by weighting selected movements also reduces average vehicle delay. We are curious whether other prioritization strategies can be generated to reduce average vehicle delay.

For example, it is often the case that a network will have a single busiest intersection. Perhaps optimizing the delay at this intersection is considerably more important than optimizing delay at other intersections. We are curious whether running ERIS (while prioritizing critical movements through weighting) at the busiest intersection while neighboring intersections run a different algorithm that primarily focuses on ensuring that the critical movements at the busiest intersection are never starved can reduce overall network delay.

Bibliography

[1] Monireh Abdoos, Nasser Mozayani, and Ana LC Bazzan. Traffic light control in non-stationary environments based on multi agent q-learning. In *2011 14th International IEEE conference on intelligent transportation systems (ITSC)*, pages 1580–1585. IEEE, 2011. 2.3.1

[2] SMA Bin Al Islam and Ali Hajbabaie. Distributed coordinated signal timing optimization in connected transportation networks. *Transportation Research Part C: Emerging Technologies*, 80:272–285, 2017. 1.3, 2.2.1, 3.3

[3] Zahra Amini, Ramtin Pedarsani, Alexander Skabardonis, and Pravin Varaiya. Queue-length estimation using real-time traffic data. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1476–1481. IEEE, 2016. 3.6.1

[4] Audi of America. Audi launches first Vehicle-to-Infrastructure (V2I) technology in the U.S. starting in Las Vegas. URL `https://www.audiusa.com/newsroom/news/press-releases/2016/12/audi-launches-vehicle-to-infrastructure-tech-in-vegas`. Accessed: 2017-05-06. 1.3.1

[5] Shumeet Baluja, Michele Covell, and Rahul Sukthankar. Traffic lights with auction-based controllers: Algorithms and real-world data. *arXiv preprint arXiv:1702.01205*, 2017. 2.3.1

[6] Ana LC Bazzan. A distributed approach for coordination of traffic signal agents. *Autonomous Agents and Multi-Agent Systems*, 10(2):131–164, 2005. 2.3.1

[7] Rachel Binder. 40 startups transforming the future of traffic. cb insights research, Apr 2019. URL `https://app.cbinsights.com/research/traffic-tech-startups-market-map-expert-intelligence/`. 1.1

[8] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013. 5.3.2

[9] Cadillac. V2V safety technology now standard on cadillac cts sedans, Mar 2017. URL `https://media.cadillac.com/media/us/en/cadillac/news.detail.html/content/Pages/news/us/en/2017/mar/0309-v2v.html`. 1.3.1

[10] Noe Casas. Deep deterministic policy gradient for urban traffic light control. *arXiv preprint arXiv:1703.09035*, 2017. 1.3, 2.3.1

[11] Pew Research Center. Demographics of mobile device ownership and adoption in the united states, Jun 2019. URL `https://www.pewinternet.org/fact-sheet/mobile/`. 1.3.1

[12] Reggie Chandra, James Bley, Stephen Penrod, and Arthur Parker. Adaptive Control Systems and Methods. U.S. Patent 8,050,854, issued November 1, 2011, 2011. 2.3.3, 4.2.2

[13] Michael AG Clark. Induction loop vehicle detector, February 4 1986. US Patent 4,568,937. 1.3.1

[14] Graham Cookson and Bob Pishue. Inrix global traffic scorecard. `http://inrix.com/scorecard/`, 2017. Accessed: 2017-09-17. 1.1

[15] Michele Covell, Shumeet Baluja, and Rahul Sukthankar. Micro-auction-based traffic-light control: Responsive, local decision making. In *Intelligent Transportation Systems (ITSC), 2015 IEEE 18th International Conference on*, pages 558–565. IEEE, 2015. 1.3, 2.3.1

[16] Christopher M. Day and Darcy M. Bullock. Computational efficiency of alternative algorithms for arterial offset optimization. *Transportation Research Record*, 2259(1):37–47, 2011. doi: 10.3141/2259-04. URL `https://doi.org/10.3141/2259-04`. 2.1

[17] Paolo Dell'Olmo and Pitu Mirchandani. Realband: An approach for real-time coordination of traffic flows on networks. 1995. 4.2.2

[18] Kurt Dresner and Peter Stone. Multiagent traffic management: A reservation-based intersection control mechanism. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 530–537. IEEE Computer Society, 2004. 2.4.2

[19] Econolite. Centracs® edaptive, Jul 2019. URL `https://www.econolite.com/products/software/centracs-edaptive/`. 2.1

[20] S. El-Tantawy and B. Abdulhai. Multi-agent reinforcement learning for integrated network of adaptive traffic signal controllers (marlin-atsc). In *2012 15th International IEEE Conference on Intelligent Transportation Systems*, pages 319–326, 2012. doi: 10.1109/ITSC.2012.6338707. 2.3.1

[21] Rhythm Engineering. Hawkeye pricing. URL `https://hawkeyeradar.com/pricing/`. 1.3.1

[22] Yiheng Feng, K Larry Head, Shayan Khoshmagham, and Mehdi Zamanipour. A real-time adaptive signal control in a connected vehicle environment. *Transportation Research Part C: Emerging Technologies*, 55:460–473, 2015. 3.6.1

[23] Michel Ferreira, Ricardo Fernandes, Hugo Conceição, Wantanee Viriyasitavat, and Ozan K Tonguz. Self-organized traffic control. In *Proceedings of the seventh ACM international workshop on VehiculAr InterNETworking*, pages 85–90. ACM, 2010. 2.4.2

[24] Nathan H Gartner. *OPAC: A demand-responsive strategy for traffic signal control*. Number 906. 1983. 1.3, 1.3.2, 3, 2.2.1, 3, 3

[25] Wade Genders and Saiedeh Razavi. Using a deep reinforcement learning agent for traffic signal control. *arXiv preprint arXiv:1611.01142*, 2016. 1.3, 2.3.1

[26] Jean Gregoire, Xiangjun Qian, Emilio Frazzoli, Arnaud De La Fortelle, and Tichakorn Wongpiromsarn. Capacity-aware backpressure traffic signal control. *IEEE Transactions on Control of Network Systems*, 2(2):164–173, 2014. 6.4

[27] S. Ilgin Guler, Monica Menendez, and Linus Meier. Using connected vehicle technology to improve the efficiency of intersections. *Transportation Research Part C: Emerging Technologies*, 46:121 – 131, 2014. ISSN 0968-090X. doi: https://doi.org/10.1016/j.trc.2014.05.008. URL `http://www.sciencedirect.com/science/article/pii/S0968090X14001211`. 6.4

[28] Jean-Jacques Henry, Jean-Loup Farges, and J Tuffal. The prodyn real time traffic algorithm. In *IFACIFIPIFORS conference on control in*, 1984. 1.3, 1.3.2, 3, 2.2.1, 3, 3

[29] Patrick M Hodge and Raymond J Lipan. Vehicle presence loop detector, September 18 1984. US Patent 4,472,706. 1.3.1

[30] Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, pages 65–70, 1979. 3.6.6, 6.7.5

[31] Hsu-Chieh Hu and Stephen F. Smith. Coping with large traffic volumes in schedule-driven traffic signal control. In *Twenty-Seventh International Conference on Automated Planning and Scheduling*, 2017. 2.2.4, 6.4

[32] Hsu-Chieh Hu and Stephen F. Smith. Softpressure: A schedule-driven backpressure algorithm for coping with network congestion. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 4324–4330, 2017. doi: 10.24963/ijcai.2017/604. URL `https://doi.org/10.24963/ijcai.2017/604`. 2.2.4, 6.4

[33] Hsu-Chieh Hu and Stephen F. Smith. Using bi-directional information exchange to improve decentralized schedule-driven traffic control. *CoRR*, abs/1907.01978, 2019. URL `http://arxiv.org/abs/1907.01978`. 2.2.2, 2.2.4, 4.2.2

[34] Hsu-Chieh Hu and Stephen F Smith. Learning model parameters for decentralized schedule-driven traffic control. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 531–539, 2020. 2.2.4, 7.2.6

[35] Hsu-Chieh Hu, Stephen F. Smith, and Rick Goldstein. Cooperative schedule-driven intersection control with connected and autonomous vehicles. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2019, Macau, SAR, China, November 3-8, 2019*, pages 1668–1673. IEEE, 2019. doi: 10.1109/IROS40897.2019.8967975. URL `https://doi.org/10.1109/IROS40897.2019.8967975`. 2.4.1

[36] Peter Koonce, Lee Rodegerdts, Kevin Lee, Shaun Quayle, Scott Beaird, Cade Braud, Jim Bonneson, Phil Tarnoff, and Tom Urbanik. Traffic signal timing manual. Technical report, 2008. 2.3.4, 6.6

[37] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker. Recent development and applications of SUMO - Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements*, 5(3&4):128–138, December 2012. 2.5

[38] Daniel Krajzewicz, Stefan Hausberger, Peter Wagner, Michael Behrisch, and Mario Krum-

now. Second generation of pollutant emission models for sumo. In *SUMO2014 - Second SUMO User Conference*, Reports of the DLR-Institute of Transportation Systems, May 2014. URL `https://elib.dlr.de/89398/`. 4.5.4

[39] Stefan Krauß. *Microscopic modeling of traffic flow: Investigation of collision free vehicle dynamics*. PhD thesis, 1998. 3.6.1

[40] Xiao Liang, S Ilgin Guler, and Vikash V Gayah. Joint optimization of signal phasing and timing and vehicle speed guidance in a connected and autonomous vehicle environment. *Transportation Research Record*, page 0361198119841285, 2019. 2.4.1

[41] Jennie Lioris, Alex Kurzhanskiy, and Pravin Varaiya. Adaptive max pressure control of network of signalized intersections. *IFAC-PapersOnLine*, 49(22):19–24, 2016. 2.3.2

[42] PR Lowrie. Scats, sydney co-ordinated adaptive traffic system: A traffic responsive method of controlling urban traffic. *Roads and Traffic Authority of New South Wales- Traffic Control Section*, 1990. 1.2.2, 2.1, 4.2.2

[43] Felipe Luyanda, Douglas Gettman, Larry Head, Steven Shelby, Darcy Bullock, and Pitu Mirchandani. Acs-lite algorithmic architecture: applying adaptive control system technology to closed-loop traffic signal control systems. *Transportation Research Record: Journal of the Transportation Research Board*, 1856:175–184, 2003. 1.2.2, 2.1, 4.2.2

[44] Highway Capacity Manual. Highway capacity manual. *Washington, DC*, 2, 2000. 1.2.2, 6.6.4

[45] T. L. McCluskey and Mauro Vallati. Embedding Automated Planning within Urban Traffic Management Operations. In *Proceedings of The 27th International Conference on Automated Planning and Scheduling*, 2017. 4, 2.2.1, 4, 4

[46] Pitu Mirchandani and Fei-Yue Wang. Rhodes to intelligent transportation systems. *IEEE Intelligent Systems*, 20(1):10–15, 2005. 1.3, 1.3.2, 2.3.3, 4.2.2

[47] National Academies of Sciences, Engineering, and Medicine. Sub-problem 1a: Maxwell drive pm peak hour - existing conditions. URL `https://www.hcmguide.com/Case2/techprob01/subproblem01a/prob1a6.htm`. (document), 3.2

[48] Rhythm Engineering, Jul 2015. URL `https://www.youtube.com/watch?v=LJYTMJ6VnnI&list=PLvjIWZt-BlL08QNW1nKQU26chYtXyhwCI&index=10`. 4.2.2

[49] Dennis I Robertson. Transyt: a traffic network study tool. *Road Research Laboratory Report, LR 253*, 1969. 2.1

[50] Dennis I Robertson and R David Bretherton. Optimizing networks of traffic signals in real time-the scoot method. *IEEE Transactions on vehicular technology*, 40(1):11–15, 1991. 1.2.2, 2.1, 4.2.2

[51] J Rogelj et al. Special report: Global warming of 1.5 ^{o}c (eds flato, g. et al.), 2018. 1.1

[52] David Schrank, Bill Eisele, and Tim Lomax. 2019 urban mobility report. Technical report, Texas A&M Transportation Institute, 2019. 1.1

[53] Seattle Department of Transportation. Mercer scoot. URL `https://www.`

seattle.gov/transportation/projects-and-programs/programs/technology-program/mercer-scoot. 2.1

[54] Suvrajeet Sen and K Larry Head. Controlled optimization of phases at an intersection. *Transportation science*, 31(1):5–17, 1997. 2.2.1, 4.2.2

[55] Guni Sharon and Peter Stone. A protocol for mixed autonomous and human-operated vehicles at intersections. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 151–167. Springer, 2017. 2.4.2

[56] Steven Gebhart Shelby. *Design and evaluation of real-time adaptive traffic signal control algorithms*. PhD thesis, The University of Arizona, 2001. 1.3, 1.3.2, 3, 2, 2.2.1, 3, 3.3, 3

[57] Siemens. Scoot adaptive traffic control: Traffic management: Siemens. URL https://new.siemens.com/us/en/products/mobility/road-solutions/traffic-management/scoot-adaptive-traffic-control.html. 2.1

[58] Stephen Smith, Gregory Barlow, Xiao-Feng Xie, and Zack Rubinstein. Surtrac: Scalable urban traffic control. In *Transportation Research Board 92nd Annual Meeting Compendium of Papers*. Transportation Research Board, 2013. 2.2.2, 3.3

[59] Stephen F Smith, Gregory J Barlow, Xiao-Feng Xie, and Zachary B Rubinstein. Smart urban signal networks: Initial application of the surtrac adaptive traffic signal control system. In *Twenty-Third International Conference on Automated Planning and Scheduling*, 2013. 1.3.2

[60] Aleksandar Stevanovic. *Adaptive traffic control systems: domestic and foreign state of practice*. 2010. Project 20-5 (Topic 40-03). National Cooperative Highway Research Program. 2

[61] Remi Tachet, Paolo Santi, Stanislav Sobolevsky, Luis Ignacio Reyes-Castro, Emilio Frazzoli, Dirk Helbing, and Carlo Ratti. Revisiting street intersections using slot-based systems. *PloS one*, 11(3):e0149607, 2016. 2.4.2

[62] Philip J Tarnoff and Peter S Parsonson. Selecting traffic signal control at individual intersections. *NCHRP Report*, 233, 1981. 2.3.4

[63] Trafficware. Synchro studio. URL https://www.trafficware.com/synchro.html. 1.2.2, 2.1

[64] United States Department of Transportation. Connected vehicle pilot deployment program. URL https://www.its.dot.gov/pilots/. 1.3.1

[65] US Department of Defense. Global positioning system standard positioning service performance standard. 2008.

[66] Pravin Varaiya. Max pressure control of a network of signalized intersections. *Transportation Research Part C: Emerging Technologies*, 36:177–195, 2013. 1.3, 2.3.2

[67] Axel Wegener, Michal Piórkowski, Maxim Raya, Horst Hellbrück, Stefan Fischer, and Jean-Pierre Hubaux. Traci: An interface for coupling road traffic and network simulators. In *Proceedings of the 11th Communications and Networking Simulation Symposium*, CNS '08, pages 155–163, New York, NY, USA, 2008. ACM. ISBN 1-56555-318-7. doi: 10.1145/1400713.1400740. URL http://doi.acm.org/10.1145/

1400713.1400740. 2.5

[68] Kerry Wu. 40 corporations working on autonomous vehicles. cb insights research, Aug 2019. URL https://www.cbinsights.com/research/autonomous-driverless-vehicles-corporations-list/. 1.1

[69] Xiao-Feng Xie, Stephen F Smith, Liang Lu, and Gregory J Barlow. Schedule-driven intersection control. *Transportation Research Part C: Emerging Technologies*, 24:168–189, 2012. 1.3, 1.3.2, 2, 2.2.1, 2.2.2, 2, 3.3, 3.6.2, 4.2.2, 2

[70] Bo Yang and Christopher Monterola. Efficient intersection control for minimally guided vehicles: A self-organised and decentralised approach. *Transportation Research Part C: Emerging Technologies*, 72:283–305, 2016. 2.4.2

[71] Weiran Yao and Zhen Sean Qian. Real-time traffic monitoring and prediction for cranberry township. 2019. 1.2.2, 2.1

[72] Juyuan Yin, Jian Sun, and Keshuang Tang. A kalman filter-based queue length estimation method with low-penetration mobile sensor data at signalized intersections. *Transportation Research Record*, 2672(45):253–264, 2018. 3.6.1