The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Expressive Real-Time Intersection Scheduling

Rick Goldstein
The Robotics Institute
Carnegie Mellon University
rgoldste@cs.cmu.edu

Stephen F. Smith
The Robotics Institute
Carnegie Mellon University
sfs@cs.cmu.edu

Abstract

We present Expressive Real-time Intersection Scheduling (ERIS), a schedule-driven control strategy for adaptive intersection control to reduce traffic congestion. ERIS maintains separate estimates for 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 environments.

Introduction

Traffic congestion is a widespread annoyance throughout global metropolitan areas. It causes increases in travel time, increases in emissions, inefficient usage of gasoline, and driver frustration. It is estimated that in 2014, congestion in urban America caused an additional 6.9 billion hours of travel time and that an additional 3.1 billion gallons of fuel were used as a result of congestion, costing roughly $160 billion (Schrank et al. 2015). Inefficient signal patterns at intersections are one major cause of such congestion. 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 reduce congestion and its negative impacts. However, three factors make real-time intersection scheduling difficult. Firstly, current data about traffic is incomplete and noisy; point sensors such as induction loop detectors and cameras miss vehicles, double count vehicles, and only provide an inexact estimate of arrival times. Secondly, even with perfect data, modelling the dynamics of a multi-agent system of heterogeneous vehicles and pedestrians is challenging. Thirdly, given perfect data and perfect agent modelling, the problem of comparing all possible signal timing strategies is exponentially large in the length of the scheduling horizon. The second and third challenges are linked by a trade-off between model expressiveness and the requirement of real-time solvability.

Inspired by many automakers' (including Ford, Audi, and Tesla) decisions to integrate connected devices into vehicles, our real-time scheduling work adopts a connected vehicle framework as a solution to the first proposed challenge of noisy sensors. In the connected vehicle framework, we assume all vehicles are equipped with a cellphone or dedicated short-range communication (DSRC) radio which continuously transmits real-time location, heading, and speed. Receivers connected to a computer at each intersection can collate this data into a real-time estimate of local traffic conditions.

In response to the second challenge of modelling traffic, we adopt a lane-based framework; we maintain separate estimates of the vehicles present in every lane approaching an intersection. Given the above information and learned vehicle dynamics, we estimate 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 which combines compatible movements (e.g. east and west) into a single movement (as in Xie et al. (2012)). The increased size of our traffic state representation ultimately makes the third challenge of calculating an optimal schedule of signal timings more difficult. We propose a heuristic function that exploits problem structure to quickly calculate meaningful lower bounds on the delay associated with potential signal patterns. This heuristic is employed by an A* search to efficiently calculate optimal schedules in real-time (every second). 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 paper is organized as follows. We begin by explaining the real-time intersection control problem and provide an overview of intersection control research. Next, we present ERIS, our strategy for the real-time intersection control problem, and discuss our application of A* search to solve single instances of the problem. We then present several experiments demonstrating that ERIS outperforms other real-time approaches in a fully connected vehicle framework. We conclude with a discussion of future work.

Problem Formulation

We start by presenting the real-time intersection control problem, introducing relevant traffic terminology, and highlighting several assumptions that underlie our model.

Real-time Intersection Control

In the real-time intersection control problem, we consider an intersection, such as the one presented in Figure 1, where traffic signals have been installed to improve mobility or increase safety. A traffic light controller, located at the intersection, is responsible for controlling the individual traffic signals. In the real-time intersection control problem, we dictate to the controller how to act at every time-step. We may prescribe that the controller extend an active green signal or end an active green signal and activate another as long as we do not violate traffic light controller constraints (which are detailed throughout this section).

We are interested in generating a feasible plan for allocating green time to different approaches that will minimize a measure of disutility. We focus on minimizing average vehicle delay which is how long a vehicle must wait for its turn to pass through the intersection. This value is the difference between a vehicle's intended arrival time at the intersection stop line (assuming no congestion) and when the vehicle actually passes through the intersection.

An intersection has inbound lanes from various directions. An intersection may associate a subset of lanes, referred to 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 is currently providing two green movements, namely the North-straight movement and the South-straight movement. (We group right turn movements into straight movements as they typically occur simultaneously). 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. The current state of the controller, which consists of green movements or yellow or red clearances, is referred to as the controller's phase.

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. This paper specifically examines the perfect information connected vehicle environment; we assume that vehicles transmit noiseless data regarding their position, speed, and heading to the intersection every second.

Once data is gathered, it is collated into an estimate of vehicle locations. Some schedule-driven approaches, ours included, combine vehicle positions into clusters (or groups) of vehicles traveling in the same direction within a specified timing gap (Xie et al. 2012). Our schedule-driven approach uses connected vehicle data as well as projected outflows from neighboring intersections to generate these clusters.

Real-time schedule-driven methods are one type of model predictive control. A schedule-driven method calculates a schedule that allows most or all vehicle clusters to pass through the intersection while minimizing the cost function. Once a schedule has been calculated, the first step of the schedule is passed to the controller to implement, and projected outflows are communicated to downstream neighbors. This process repeats every time-step.

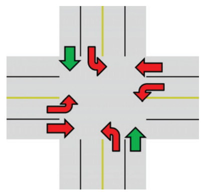


Figure 1: Example Intersection

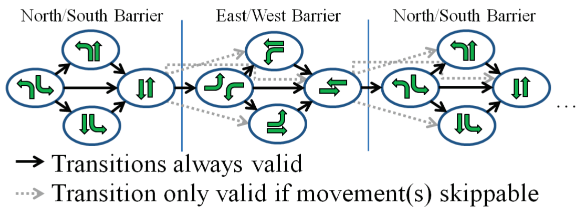


Figure 2: Compatible Green Phase Orderings

Traffic Light Controller Assumptions

We assume that all intersections are four-sided intersections controlled by a dual ring barrier controller. The dual ring barrier controller may provide two compatible movements with a green signal. Compatible movements and legal transitions between compatible pairs are presented below in Figure 2. (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. Straight movements cannot be skipped. Currently, we allow only protected movements at our intersections. Vehicles that desire to turn left may only do so when they receive a green turn arrow; similarly, right turns on red are forbidden in our model. We discuss relaxations of this assumption at the end of this paper.

Related Work

A general treatment of all past intersection scheduling strategies is beyond the scope of this paper; we refer the reader to the works by Shelby (2001) and Stevanovic (2010) for more detailed overviews of various intersection control strategies. Here, we briefly summarize several control strategies that adapt by setting high-level parameters. We then introduce real-time intersection scheduling and highlight three real-time scheduling techniques that are most similar to our work.

High-Level Intersection Control

High-level strategies set timing parameters, typically *cycle length* for an entire network of intersections, *splits* between green phases at each traffic signal within the network, and *offsets* which 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 a hill-climbing algorithm to minimize an objective such as delay or number of stops on historical data (Robertson 1969). Several more recent procedures, such as SCOOT, SCATS, and ACS Lite perform similar optimizations that adapt timing parameters during execution based on observed data (Robertson and Bretherton 1991; Lowrie 1990; Luyanda et al. 2003).

SCOOT, SCATS, and ACS Lite adapt gradually to changing traffic conditions. They 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.

Real-time Intersection Scheduling

Real-time intersection scheduling methods improve upon high-level strategies by making decisions in direct response to sensed traffic. Real-time methods are not constrained by fixed timing requirements and can immediately adapt when one side of the intersection is unexpectedly much busier. The majority of these methods (including ERIS) are schedule-driven methods. They use real-time vehicle information to generate a schedule of phases to minimize a cost function. The first step of this schedule is then sent to the traffic light controller and executed. This process repeats as frequently as once per second, depending on the specific method. The majority of schedule-driven methods are decentralized, meaning each intersection of a network is controlled by a single computer making scheduling decisions primarily based on local traffic.

Our work most closely resembles the original SURTRAC system (Xie et al. 2012; Smith et al. 2013). SURTRAC is a decentralized schedule-driven method which 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. To ensure that the forward dynamic program can run in real-time, SURTRAC makes simplifications to the underlying scheduling problem. SURTRAC schedules a simpler controller that requires left turn movements to end simultaneously which reduces the branching factor of the dynamic programming search. Additionally, SURTRAC combines vehicle clusters travelling in compatible directions (e.g. east and west) into larger clusters. 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.

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.

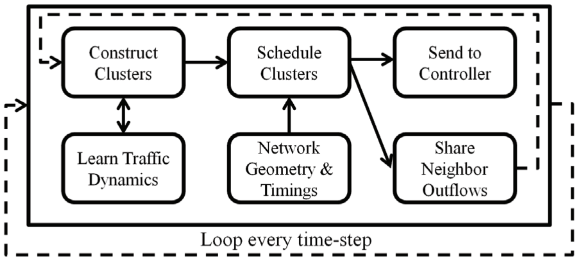


Figure 3: ERIS Control Flow

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

There are several real-time branch and bound schedule-driven methods. 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 over-estimate and under-estimate 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 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 SURTRAC, is able to efficiently calculate schedules that are several minutes in duration without compromising granularity.

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.

Expressive Real-time Intersection Scheduling

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 controller to execute. This section summarizes the overall control flow of an ERIS controlled intersection. The control flow is depicted in Figure 3 below. Every second, the ERIS Executor runs an outer loop that executes the necessary modules to make a scheduling decision. The Executor first cal-

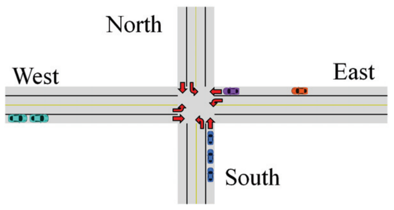


Figure 4: Intersection for Clustering

🚗	Count	%
↰	100	25%
➡	300	75%

Figure 5: West-Origin Turn Ratios

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

Once clusters are obtained, they are passed to the Scheduling Module to obtain an optimal schedule of phases that allows all vehicle clusters to pass through the intersection while minimizing total delay. Clusters are scheduled in accordance with programmed traffic dynamics including the fixed phase orderings, 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 controller to execute. Similarly, scheduled cluster outflows are broadcast to and received from neighboring intersections for use during the next time-step when building vehicle clusters.

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 intended arrival time at the intersection, a size, and a flow rate describing how quickly we expect the vehicles in the cluster to pass through the intersection. An example intersection and the resulting clusters are presented below in Figure 4 and Figure 6, respectively.

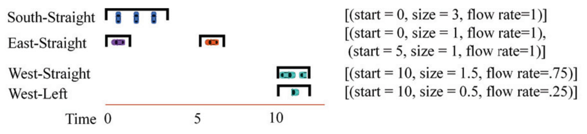

South-Straight
East-Straight
West-Straight
West-Left
Time
0
5
10
[(start = 0, size = 3, flow rate=1)]
[(start = 0, size = 1, flow rate=1),
(start = 5, size = 1, flow rate=1)]
[(start = 10, size = 1.5, flow rate=.75)]
[(start = 10, size = 0.5, flow rate=.25)]

Figure 6: Arrival Time Line and Clusters

The intersection presented in Figure 4 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 splits (as shown in Figure 5). 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). Flow rates are also appropriately weighted (0.75 and 0.25, respectively).

Scheduling Module

The Scheduling Module runs an A* search to calculate an optimal schedule with respect to 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 which allows us to efficiently calculate this optimal schedule. We conclude this section with a brief discussion of our dominance check algorithm.

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. 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 intended arrival time, a size, and a flow rate. We define A_i as the sequence of ordered clusters in 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 real-world 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 $\vec{t^a} = (t_1^a, t_2^a)$.

Search States

We define a search state as a tuple: $(\vec{c}, \vec{m}, \vec{t}, T, g, h, f, PS)$. $\vec{c} = (c_1, c_2, ..., c_L)$, 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 defines the end time of the most recent scheduling decision. 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 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.

Start State

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

Goal State

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

State Transitions

Our state transition algorithm is presented below 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 $(\vec{m}, \vec{t}, \Delta T)$ ∈ 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:         $\Delta\vec{c}$, ΔDelay = 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 ∪ {$(\vec{c}, \vec{m}, \vec{t}, T, g, h, f, PS)$}
14:    return(children)
```

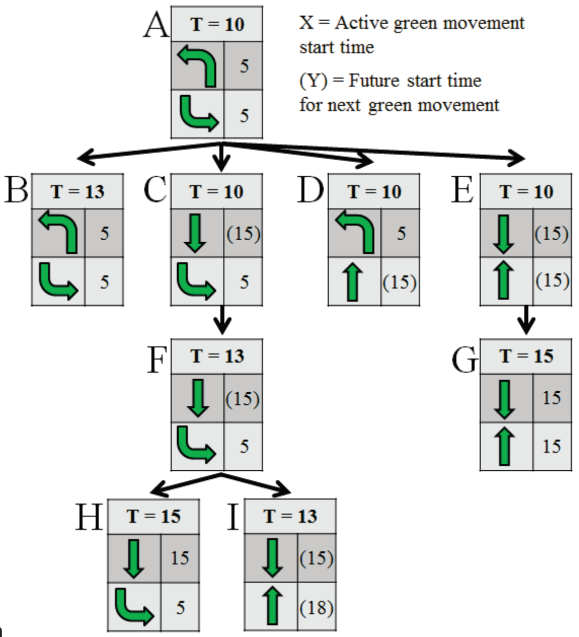
Our algorithm expands a parent state (whose elements are denoted with the subscript p), calculating all possible child states to ultimately add to our search's open list. In addition to the parent state, our algorithm also requires arrival information, namely $\vec{A}$, to generate child states.

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 7. If both movements are left turn green movements (Figure 7, 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 2). 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.

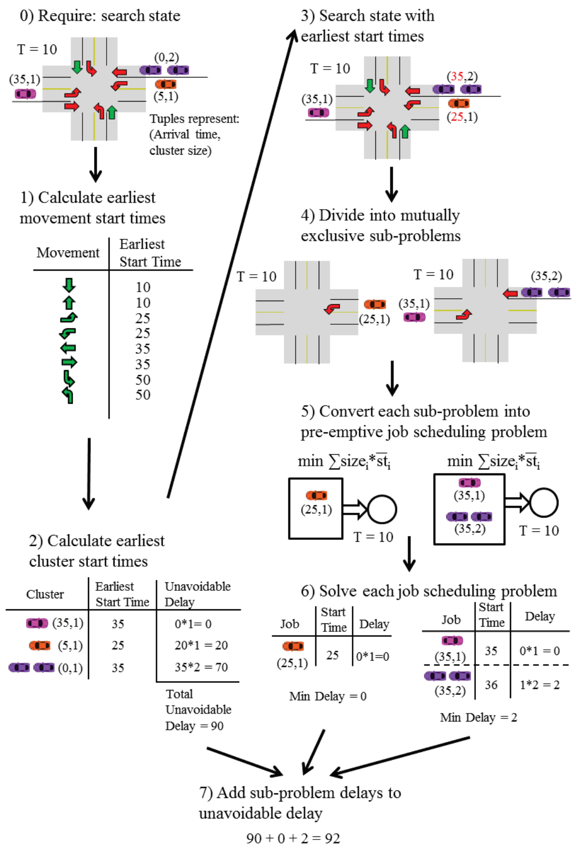
When in a state with two green movements that are waiting to begin (state E), we step forward to the earlier start time (state G). However, if in a state where one green movement is active and the other is waiting to begin (state C), we cannot automatically jump forward. We must allow the current green 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).

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



hh

Figure 7: Phase Transitions Example


0) Require: search state
T = 10
(35,1)
(0,2)
(5,1)
Tuples represent: (Arrival time, cluster size)
1) Calculate earliest movement start times
Movement
Earliest Start Time
10
10
25
25
35
35
50
50
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*st_i
(25,1)
T = 10
min ∑size_i*st_i
(35,1)
(35,2)
T = 10
6) Solve each job scheduling problem
Min Delay = 0
Min Delay = 2
7) Add sub-problem delays to unavoidable delay
90 + 0 + 2 = 92

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

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

Figure 8: Heuristic Calculation Pipeline

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 far 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).

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 8.

Our heuristic function operates on the current search state. In the example presented in Figure 8, 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 phase, and the current world time, we first calculate the earliest possible start time for all green movements. This involves using the fixed phase orderings, phase minimum times, and forced yellow and red clearances. 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

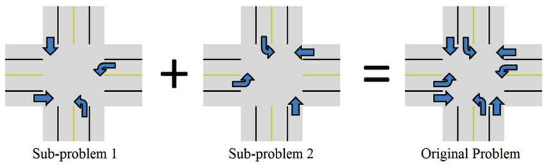


Figure 9: Sub-problem splits

pink cluster arrives at time 35, 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 9. Other sub-problem divisions are possible, but splitting into two sub-problems leads to tighter bounds. 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 penalties. 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, 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.

Closed List & Dominance Checks

A* search typically maintains a closed list to ensure that search states are not expanded more than once. Our search states are defined by two discrete features, namely the two movements of the controller, and several continuous features, including the fractional 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. By hashing states based on $\vec{m}$ and discretizations of T, the scheduler can obtain a set of states that might dominate the candidate to expand. If the search identifies an equivalently hashed state having served more clusters in every direction, having a lower cost, and with more timing flexibility, the candidate state is not expanded.

Experimental Evaluation

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 less-expressive real-time scheduling method based on the SURTRAC model. Second, we compare to a variant of fully-actuated control, which we refer to as connected fully-actuated control. In traditional fully-actuated control, a traffic light controller will terminate a green movement when it has been several seconds (e.g. 4 seconds) since the most recent vehicle passed over a detector (Tarnoff and Parsonson 1981). 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 4 seconds) to end earlier. In connected fully-actuated control, we assume that vehicles only send a message to an intersection when they are within 4 seconds of the intersection. If a message is received, the movement continues; otherwise, the movement ends. Using these models as comparisons presents a more competitive comparison than using simpler algorithms such as traditional fully-actuated control or fixed timing plans that have previously been shown to significantly underperform SURTRAC (Xie et al. 2012).

We ran evaluations of the three methods on the Simulation of Urban MObility (SUMO) (Krajzewicz et al. 2012). 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) (Wegener et al. 2008). TraCI provides real-time data about vehicle locations and allows external control of traffic decisions.

All simulations run for an hour of simulated time. For each simulation, we report average time loss, per vehicle.

Time loss varies from our earlier definition of delay as time loss also includes other factors that we do not aim to improve, such as time loss from driving behind a vehicle with a lower cruising speed. To eliminate the effects of simulation start up and termination, we only report the time loss 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.

Single Intersection

We first examine a network consisting of 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 1). Average delay per vehicle (in seconds) and standard errors across a range of vehicle volumes are presented in Table 1 below. We also present ERIS' improvement when measured against the better of SURTRAC and 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
SURTRAC	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 1: Single Intersection Comparisons

ERIS outperforms SURTRAC 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 SURTRAC'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. SURTRAC'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 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. We present our results in Table 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
SURTRAC	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 2: Network Comparisons

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.

Statistical Significance

We ran 20 two-sided paired t-tests comparing ERIS' performance with each of the other methods. Each test compared ERIS' average vehicle time loss to a competing method's average vehicle time loss at a single congestion level on a single network. (We did not compare SURTRAC to 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% (Holm 1979). In 19 of the 20 comparisons (all except ERIS vs SURTRAC for 5300 vehicles/hour in the 3 x 3 grid setting), ERIS' average vehicle time loss is statistically significantly different than the comparison method's average vehicle time loss.

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 presents run-time performance percentiles in the most congested (and hardest) case for the single intersection model. All timings are presented in milliseconds.

Method	Mean	25%tile	50%tile	75%tile	90%tile	95%tile
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: Run-time Comparisons, 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 run-time to find the optimal schedule. This increase in run-time is larger for more complicated problems.

Conclusion

This paper 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 demonstrated that ERIS outperforms other real-time intersection scheduling methods and can reduce vehicle delay by as much as 20% when compared to other methods. The underlying model of ERIS is general, making it easy to extend, and the A* search is well-suited to solve more complicated problem representations than other intersection scheduling techniques.

This paper made several modelling assumptions that we intend to relax in the future. First, we assumed that 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 timestep. While this was a useful assumption for demonstrating the improvement of ERIS over other scheduling methods, it is not completely 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. In future work, we will relax the perfect information assumption and examine performance on stale, noisy data, possibly by incorporating Bayesian state estimation techniques (e.g. Kalman filtering).

We also assumed all intersections were four sided intersections with protected movements. Allowing unprotected left turns and right turns on red present another source of uncertainty to our model. Similarly, we intend to look at intersections with single lanes in each direction, namely lanes that jointly serve left turn, straight, and right turn movements. If a vehicle in front of the lane is unable to turn left (due to oncoming traffic), all traffic behind this vehicle is blocked. We hypothesize that our expressive model will perform better at estimating such blockages and know if it should provide one movement with a delayed green signal.

Finally, we intend to examine weighting different types of traffic. Policy makers may have a hierarchy on traffic priorities. Perhaps emergency vehicles should always receive priority even if it requires delaying an arterial road. On the other hand, buses should receive some priority, but not to the extent of emergency vehicles. We intend to measure the impacts of such policies on each priority group and the overall system as a whole.

References

Al Islam, S. B., and Hajbabaie, A. 2017. Distributed coordinated signal timing optimization in connected transportation networks. *Transportation Research Part C: Emerging Technologies* 80:272–285.

Holm, S. 1979. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics* 65–70.

Krajzewicz, D.; Erdmann, J.; Behrisch, M.; and Bieker, L. 2012. Recent development and applications of SUMO - Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements* 5(3&4):128–138.

Lowrie, P. 1990. 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*.

Luyanda, F.; Gettman, D.; Head, L.; Shelby, S.; Bullock, D.; and Mirchandani, P. 2003. 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.

Robertson, D. I., and Bretherton, R. D. 1991. Optimizing networks of traffic signals in real time-the scoot method. *IEEE Transactions on vehicular technology* 40(1):11–15.

Robertson, D. I. 1969. Transyt: a traffic network study tool. *Road Research Laboratory Report, LR 253*.

Schrank, D.; Eisele, B.; Lomax, T.; and Bak, J. 2015. 2015 urban mobility scorecard. Technical report, Texas A&M Transportation Institute.

Shelby, S. G. 2001. *Design and evaluation of real-time adaptive traffic signal control algorithms*. Ph.D. Dissertation, The University of Arizona.

Smith, S.; Barlow, G.; Xie, X.-F.; and Rubinstein, Z. 2013. Surtrac: Scalable urban traffic control. In *Transportation Research Board 92nd Annual Meeting Compendium of Papers*. Transportation Research Board.

Stevanovic, A. 2010. *Adaptive traffic control systems: domestic and foreign state of practice*. Project 20-5 (Topic 40-03). National Cooperative Highway Research Program.

Tarnoff, P. J., and Parsonson, P. S. 1981. Selecting traffic signal control at individual intersections. *NCHRP Report* 233.

US Department of Defense. 2008. Global positioning system standard positioning service performance standard.

Wegener, A.; Piórkowski, M.; Raya, M.; Hellbrück, H.; Fischer, S.; and Hubaux, J.-P. 2008. Traci: An interface for coupling road traffic and network simulators. In *Proceedings of the 11th Communications and Networking Simulation Symposium*, CNS '08, 155–163. New York, NY, USA: ACM.

Xie, X.-F.; Smith, S. F.; Lu, L.; and Barlow, G. J. 2012. Schedule-driven intersection control. *Transportation Research Part C: Emerging Technologies* 24:168–189.