Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)

Learning Model Parameters for Decentralized Schedule-Driven Traffic Control

Hsu-Chieh Hu, Stephen F. Smith
Carnegie Mellon University
Pittsburgh, PA, USA
hsuchieh@andrew.cmu.edu, sfs@cs.cmu.edu

Abstract

Model-based intersection optimization strategies that produce signal timings over a specified optimization horizon have been widely investigated for urban traffic signal control, and recent work in this area produced a scalable approach to real-time traffic control based on a decentralized schedule-driven optimization model. In this approach, a scheduling agent is associated with each intersection. Each agent senses the traffic approaching its intersection through local sensors and in real-time constructs a schedule that minimizes the cumulative wait time of vehicles approaching the intersection over the current look-ahead horizon. Intersections then exchange schedule information with their neighbors to achieve network level coordination. Although the approach is general and has demonstrated substantial success, its effectiveness in a given road network depends on the extent to which various parameters of the model, e.g, maximum green time, are adjusted to match that network's actual flow conditions over time. To address this problem, we propose a two-stage hierarchical structure that combines online planning and reinforcement learning. Reinforcement learning is applied to adjust the parameters of the model over a longer time-scale. On the other hand, online planning is used to compute the schedule for managing the traffic signals in the shorter term. We demonstrate how this hybrid approach outperforms the original approach in real-time traffic signal control problems.

Introduction

Urban mobility is becoming an increasingly critical problem, and it is commonly recognized that better optimization of traffic signals could lead to substantial reduction in traffic congestion. Recent development of a decentralized online planning approach to the traffic signal control problem has achieved significant traffic flow efficiency improvements through real-time, distributed generation of long-horizon, signal timing plans. (Xie, Smith, and Barlow 2012; Smith et al. 2013) The key idea behind this schedule-driven approach is to formulate the intersection scheduling problem as a single machine scheduling problem, where input jobs are represented as sequences of spatially proximate vehicle clusters (approaching platoons, queues).This aggregate representation enables efficient generation of plans with longer horizons that incorporate multi-hop traffic flow information, and thus network-wide coordination is achieved through exchange of schedule information.

One potential limitation of this approach, however, is its reliance on a scheduling model whose parameters must be configured to match actual traffic conditions. Although the online planning approach is able to compute schedules in real-time (i.e., on a second-by-second basis), the settings of model parameters such as minimum and maximum green times for each phase, vehicle free-flow speed and intersection turning percentages must be fixed in advance to reflect expected traffic flow patterns, and the extent to which these settings match actual traffic conditions will ultimately determine the accuracy of the signal timing schedules that are generated over time. Hence, we would expect to be able to further boost the performance of the traffic signal control system by introducing the ability to adjust these parameters over longer time scales in response to observed traffic behavior.

In this paper, we explore this hypothesis. We propose a hierarchical framework that integrates reinforcement learning (RL) with schedule-driven traffic control, where the upper-level module learns a policy to configure parameters, and the lower-level module generates signal timing plans that minimize cumulative vehicle delay according to current model parameter settings. More specifically, the model controls traffic flow at an intersection over two levels of hierarchy:

1. The lower-level module regenerates a signal timing plan every second and interacts with the hardware controller at the intersection and the intersection's direct neighbor intersections to execute it in rolling horizon fashion. Meanwhile, the traffic data produced by execution (e.g, queue length data), as well as projected demand (e.g., number of arriving vehicles) extracted from incoming messages received from upstream intersections is relayed to the upper layer for learning a control policy.
2. The upper-level module computes statistics (e.g., average queue length and average number of arriving vehicles) over a larger time-scale based on the received data from the lower-level module and updates model parameters.

The work flow is described in Figure 1. We are interested in a cooperative setting where all agents tend to smooth traffic flow, which is to jointly minimize the globally averaged delay over all intersections. Basically, two types of information are exchanged horizontally between intersections: a) *schedule*

information in the lower-level module and b) *traffic statistics in the upper-level module*. Thus, the ability to utilize information received from neighbors is especially essential for acting in such cooperative environments.

In addition, we adopt a framework of *decentralized training* with *decentralized execution* for RL (Zhang et al. 2018) instead of the more traditional centralized training approach (Lowe et al. 2017; Foerster et al. 2018), mirroring the philosophy of the original decentralized online planning approach to traffic signal control that we are proposing to extend. Fully decentralized learning greatly increases its scalability, as the number of joint states and actions grows exponentially with the size of the network in a centralized training context. A decentralized training approach also preserves the advantage of allowing the system to be incrementally deployed to urban areas over time. However, the challenge of decentralized training is that the intersections are coupled together and the environments become non-stationary from the perspective of any individual intersection. Thus, we propose a simple extension of actor-critic policy gradient methods where the critic and actor are augmented with extra information about other intersections. By exchanging information (i.e., schedule information and traffic statistics), the agents are able to discover complex communicative coordination strategies to adjust parameters and apply online planning (Wu, Zilberstein, and Chen 2009). To ensure scalability, we assume that scheduling messages continue to be exchanged only between direct neighbors (although information can travel multiple hops over multiple planning and learning episodes).

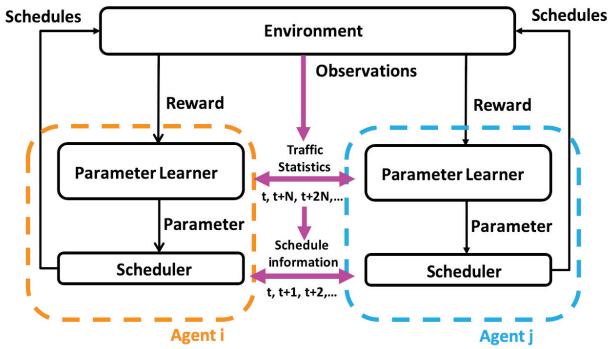


Figure 1: The hierarchical structure of traffic control systems with the upper-level reinforcement learning module and the lower-level planning module. An agent is able to share schedule information and traffic statistics with its neighbor agents.

Related Work

Real-Time Traffic Signal Control

In urban environments, *centralized* traffic signal control that adjust signal timing plan parameters (e.g., cycle time, green time split) according to actual sensed traffic data (Lowrie 1992; Heung, Ho, and Fung 2005; Gettman et al. 2007) have been proposed to overcome the inefficiency of relying on conventional fixed signal timing plans. However, these approaches are designed to accommodate continuous gradual change in traffic patterns (typically adjusting parameters after integrating information for several minutes), and are not responsive to real-time traffic events and disruptions. Alternatively, *decentralized* online planning approaches have been proposed (Sen and Head 1997; Gartner, Pooran, and Andrews 2002; Shelby 2001; Cai, Wong, and Heydecker 2009; Jonsson and Rovatsos 2011). These approaches solve the problem of scalability in principle, but have historically had difficulty computing plans in real-time with a sufficiently long horizon to achieve network-level coordination.

RL-Based Approach

Real-time traffic signal control is a complicated control problem because of its responsiveness, scalability and especially the non-linearity of queueing dynamics. In machine learning, RL formalizes such control problems as finding a policy that maximizes expected future rewards. Although several works in RL has already made progress in traffic signal control problems (Wiering 2000; Kuyer et al. 2008; Bazzan and Klügl 2014), they are often slow to converge and difficult to apply under the real-time setting if traffic flows vary frequently. Moreover, if we consider a more realistic setting in which external information is provided solely through local sensors (the most common being physically nearby in-roadway induction loop detectors or cameras) rather than relying on global state or vehicle-based representation, the online planning approach (Xie, Smith, and Barlow 2012) is viewed as a recent state of the art (Covell, Baluja, and Sukthankar 2015). In this sense, learning model parameters for planning is a more reasonable solution for realistic traffic signal control systems.

It is well established that agent-based approaches suit the decentralized traffic management well given newly developed sensing technologies and historical temporal data, as well as the frequent and flexible interaction between the agents and their environment (Dresner and Stone 2008; Bazzan and Klügl 2014). A common approach related to control of traffic signals is to let multiple agents learn a policy for mapping states to actions by monitoring traffic flow and selecting actions (Wiering 2000; Kuyer et al. 2008). However, recent development in RL has placed greater emphasis on the framework of centralized training and decentralized execution (Lowe et al. 2017; Foerster et al. 2018), which is not practical for such real-world applications as intelligent transportation systems or sensor networks. For those applications, a central controller does not exist or may be costly to install. The closest to our problem setting is that of (Zhang et al. 2018), who also considers a fully decentralized multi-agent RL.

Communication in RL and Planning

To decentralize multi-agent RL problems, a recent trend is to let agents learn independently but allow them to interact with each other and combine their policies or plans. This provides a new trade-off between total centralization and total independence. Exchange of information between a group of agents may increase accuracy and learning speed at the expense

of communication (Nunes and Oliveira 2004). The work in (Kuyer et al. 2008) also focuses on exchanging information to benefit reinforcement learning and explicit coordination among agents through a coordination graph. However, this leads to an increase in complexity as the graph becomes larger. In the field of planning, exchanging information to extend the horizon is considered in (Sen and Head 1997; Gartner, Pooran, and Andrews 2002; Xie, Smith, and Barlow 2012) as a way to accommodate non-local information.

Preliminaries

Schedule-Driven Traffic Control As indicated above, the key to the single machine scheduling problem formulation of the schedule-driven approach of (Xie, Smith, and Barlow 2012) is an aggregate representation of traffic flows as sequences of clusters c over the planning (or prediction) horizon. Each *cluster* c is defined as $(|c|, arr, dep)$, where $|c|$, arr and dep are number of vehicles, arrival time and departure time respectively. Vehicles entering an intersection are clustered together if they are traveling within a pre-specified interval of one another. The clusters become the jobs that must be sequenced through the intersection (the single machine). Once a vehicle moves through the intersection, it is sensed and grouped into a new cluster by the downstream intersection.The sequences of clusters provide short-term variability of traffic flows at each intersection and preserve the non-uniform nature of real-time flows. Specifically, the *road cluster sequence* $C_{Road,m}$ is a sequence of $(|c|, arr, dep)$ triples reflecting each approaching or queued vehicle on entry road segment m and ordered by increasing arr. Since it is possible for more than one entry road to share the intersection in a given *phase* (a phase is a compatible traffic movement pattern, e.g., East-West traffic flow), the *input cluster sequence* C can be obtained through combining the road cluster sequences $C_{Road,m}$ that can proceed concurrently through the intersection. The travel time on entry road m defines a finite horizon (H_m), and the prediction horizon H is the maximum over all roads.

Every time the cluster sequences along each approaching road segment are determined, each cluster is viewed as a non-divisible job and a forward-recursion dynamic programming search is executed in a rolling horizon fashion to continually generate a phase schedule that minimizes the cumulative delay of all clusters. The frequency of invoking scheduling is once a second for reducing uncertainty associated with clusters and queues. The process constructs an optimal sequence of clusters that maintains the ordering of clusters along each road segment, and each time a phase change is implied by the sequence, then a delay corresponding to the intersection's yellow/all-red changeover time constraints is inserted. If the resulting schedule is found to violate the maximum green time constraints for any phase (introduced to ensure fairness), then the first offending cluster in the schedule is split, and the problem is re-solved.

Formally, the resulting *control flow* can be represented as a tuple (S, C_{CF}) shown in Figure 2, where S is a sequence of phase indices, i.e., $(s_1, \cdots, s_{|S|})$, C_{CF} contains the sequence of clusters $(c_1, \cdots, c_{|S|})$ and the corresponding starting time after being scheduled. More precisely, the delay that each cluster contributes to the cumulative delay $\sum_{k=1}^{|S|} d(c_k)$ is defined as $d(c_k) = |c_k| \cdot (ast - arr(c_k))$, where ast is the actual start time that the vehicle is allowed to pass through, which is determined by the optimization process. The optimal sequence (schedule) C^*_{CF} is the one that incurs minimal delay for all vehicles.

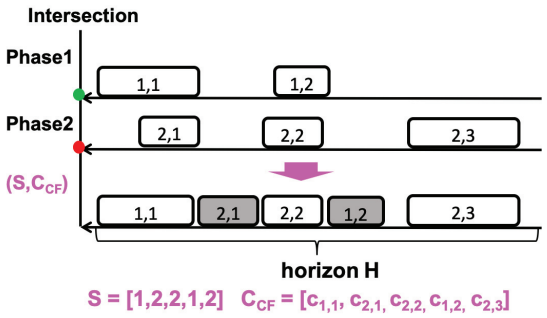


Figure 2: The resulting control flow (S, C_{CF}) calculated by scheduling agents: each block represents a vehicular cluster. The shaded blocks represent the delayed clusters.

To collaborate with neighbor intersections, each intersection receives a projection of expected outflows from its upstream neighbors and plugs it into its local computation. After starting to execute its schedule, the resulting flows are communicated to its downstream neighbors. Since a vehicle may enter into/leave from intersection via different road segments, the clusters that are propagated to neighbors over extended look-ahead horizon H are split and weighted by turning movement proportion. Thus, the weight $|c|$ of the non-local cluster will be a fractional number to reflect the uncertainty of movement. The turning movement proportion data is estimated by taking average of traffic flow rates for different phases. All approaching vehicles are sensed through the intersection's lane detectors.

Learned Parameter: Maximum Green Time Schedule-driven traffic control has several parameters in its model and some of them need to be updated or configured according to the traffic condition. In this section, we introduce maximum green time G_{max}, a parameter that upper bounds the duration of phase and has impact on average delay performance. It can be understood as deciding when to cut off vehicular clusters (in the worst case).

The phase duration of each phase i is constrained to be within $[G_{i,min}, G_{i,max}]$, in which $G_{i,min}$ and $G_{i,max}$ are respectively the minimum and maximum green times for phase i. $G_{i,min}$ is usually configured for pedestrian safety reasons, i.e., how long a pedestrian should take to cross street. $G_{i,max}$, alternatively, is designed to ensure fairness, but can lead to significant inefficiencies if set improperly. For instance, a too small $G_{i,max}$ may cause traffic signal to switch too frequently and lower the service rate. On the other hand, a too long maximum green time could cause queues of the competing phase to grow without bound. Note that it is not

always advantageous to have longer G_{max}. Correctly setting maximum green time will assist the lower-level planner to perform better under different traffic conditions.

Background

Deterministic Policy Gradient (DPG) Algorithm Policy gradient methods are widely used in a variety of reinforcement learning problems. The main idea is to directly optimize the parameters θ of policy $\pi_\theta(a|s)$: $\mathcal{S} \times \mathcal{A} \mapsto [0, 1]$, where state space $\mathcal{S}$ could be discrete or continuous and action space $\mathcal{A}$ is discrete, in order to maximize the objective $J(\theta) = \mathbb{E}_{s\sim p^\pi, a\sim\pi_\theta}[R]$ by taking steps in the direction of gradient $\nabla_\theta J(\theta)$, where $R = \sum_{t=0}^{T} \gamma^t r^t$. Using the Q function, the gradient of the policy can be written as

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim p^\pi, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s, a)], \quad (1)$$

where p^π is $p(s)$. It is also possible to extend policy gradient to deterministic policies $\mu_\theta(s) : \mathcal{S} \mapsto \mathcal{A}$ (Silver et al. 2014), where state space $\mathcal{S}$ could be discrete or continuous and action space $\mathcal{A}$ is continuous. It is especially useful for configuring parameters of the model. We can write the gradient of the objective $J(\theta) = \mathbb{E}_{s\sim p^\mu}[R(s, a)]$ as

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\mathcal{D}}[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)}], \quad (2)$$

where the state s is drawn from experience replay buffer $\mathcal{D}$. Deep deterministic policy gradient (DDPG) is a variant of DPG where the policy μ and Q function Q^μ are approximated with deep neural networks.

Coordinated Reinforcement Learning Coordinated reinforcement learning is a method for agents to select cooperative actions under multi-agent setting. In this approach, agents make coordinated decisions based on structured communication between agents and share information to achieve a principled learning strategy. We can show that if the global utility function Q is approximated by the sum of local utilities Q_j , then it is possible to use the coordination graph (Guestrin, Lagoudakis, and Parr 2002) to compute the maximizing joint action efficiently. The coordination is achieved using the max-plus algorithm (Kuyer et al. 2008), which estimates the optimal joint action by sending locally optimized message among neighbor agents.

Temporal Abstraction Learning and operating over different levels of temporal abstraction is a key challenge in tasks involving long-horizon planning (Dietterich 2000; Kulkarni et al. 2016). In hierarchical deep reinforcement learning (HDRL), the agent uses a two-stage hierarchy consisting of a controller and a meta-controller. The meta-controller receives a state and chooses a goal. The controller selects an action using state and goal. The reward function for the controller is to maximize cumulative intrinsic reward that depends on the goal set by the meta-controller. Similarly, the meta-controller attempts to maximize the extrinsic reward received from the environment. The time-scales for the two controllers are different. For certain applications, combining planning and learning is more efficient than the pure hierarchical learning solution. For instance, planning could replace the controller in HDRL to minimize objective directly without training overhead.

A Fully Decentralized Hierarchical Algorithm

System Model The system of interest is an urban road network. The connectivity of the network is represented by a graph $G = (V, E)$ as shown in Figure 3, where V is the set of nodes and E is the set of links. We consider a network consisting of $|V|$ nodes (intersections) and $|E|$ links (road segments). Each node has a scheduling agent to serve jobs (clusters) belonging to specific classes. On the link (i, j), the node j has a corresponding queue to buffer approaching jobs. The scheduling agent can only serve one queue at a time. Given the observations of all agents, each agent is able to learn a parameterized policy μ_θ to configure $\mathbf{G} = [G_{1,max}, \cdots, G_{P,max}]$, where P is the number of input queues. In the following sections, we assume the number of input queues is equal to the number of phases and and we assume that all agents have the same number of queues to simplify the formulation.

From Centralized to Decentralized Critic Since the problem of scalability is a critical issue of urban traffic control systems, we adopt the framework of decentralized training over the recent trend of centralized training with decentralized execution. Moreover, we allow the policies to use exchanged neighbor information. Before introducing the fully decentralized algorithm, a globally optimal formulation is provided as an initial step for deriving decentralized method. Consider a queueing network with N scheduling agents with policies parameterized by $\boldsymbol{\theta} = \{\theta_1, \cdots, \theta_N\}$ and let $\boldsymbol{\mu} = \{\mu_{\theta_1}, \cdots, \mu_{\theta_N}\}$ be the set of all agents' continuous policies used for setting $\mathbf{G}$. Then, we can write the gradient of the global expected delay to θ_i as:

$$\begin{aligned} \nabla_{\theta_i} J(\boldsymbol{\mu}) &= \mathbb{E}_{s\sim\rho^{\boldsymbol{\mu}}}\left[\nabla_{\theta_i} Q^{\boldsymbol{\mu}}(s, \boldsymbol{\mu})\right] \\ &= \mathbb{E}_{s\sim\rho^{\boldsymbol{\mu}}}\left[\nabla_{\theta_i} \mu_{\theta_i}(s) \nabla_{a_i} Q^{\boldsymbol{\mu}}(s, \mathbf{a})|_{a_i=\mu_{\theta_i}(s)}\right], \end{aligned} \quad (3)$$

where s is a true network state and $Q^{\boldsymbol{\mu}}(s, \mathbf{a})$ is the centralized critic. The multi-agent environment we consider is a queueing network, so that the centralized critic can be decomposed into terms that involve the connecting link of two adjacent agents (intersections).

$$Q^{\boldsymbol{\mu}}(s, \mathbf{a}) = \sum_{(n,m)\in E} Q^{\boldsymbol{\mu}}_{n,m}(s, a_n, a_m). \quad (4)$$

We replace the centralized critic with right term of (4) and get

$$\begin{aligned} &\nabla_{\theta_i} J(\boldsymbol{\mu}) = \\ &\mathbb{E}_{s\sim\rho^{\mu}}\left[\nabla_{\theta_i} \mu_{\theta_i}(s) \nabla_{a_i} \sum_{(n,m)\in E} Q^{\boldsymbol{\mu}}_{n,m}(s, a_n, a_m)|_{a_i=\mu_{\theta_i}(s)}\right]. \end{aligned} \quad (5)$$

With (5), the gradient can be simplified further by the following proposition.

Proposition 1. *With $G = (V, E)$, the gradient of centralized critic in terms of a_i can be simplified to*

$$\nabla_{a_i} \sum_{(n,m)\in E} Q^{\boldsymbol{\mu}}_{n,m}(s, a_n, a_m) = \nabla_{a_i} \sum_{j\in\mathcal{N}_i} Q^{\boldsymbol{\mu}}_{i,j}(s, a_i, a_j), \tag{6}$$

where $\mathcal{N}_i$ is a set of neighbor agents of the local agent i.

Proof. The proof is straightforward since all irrelevant terms to a_i can be eliminated from the gradient due to the structure of the graph G. □

Approximated Network State and Loss In the simplest case, network state s could consist of the observations of all agents, however, for a decentralized real-time application it is not practical to hold such assumption. In the decentralized systems, since the $Q^{\boldsymbol{\mu}}(s, \mathbf{a})$ is learned separately, agents are allowed to access different state information and arbitrary reward structures, including a shared reward in a cooperative setting. Given that, we assume that each agent i is able to access the local observation of *queue length*:

$$\mathbf{q}_i(\tau) = [q_{1,i}(\tau), \cdots, q_{P,i}(\tau)], \quad t \le \tau < t + T,$$

and *the number of arriving vehicles* sent by neighbor intersections:

$$\mathbf{c}_i(\tau) = [c_{1,i}(\tau), \cdots, c_{P,i}(\tau)], \quad t \le \tau < t + T,$$

where P is the number of input queues (i.e., phases), t is the last time step when $\mathbf{G}_i$ is updated, and T denotes the number of time steps until $\mathbf{G}_i$ is updated. The network state can thus be expressed by $s = \{\mathbf{q}_1, \cdots, \mathbf{q}_N, \mathbf{c}_1, \cdots, \mathbf{c}_N\}$. Furthermore, each agent i obtains average delay (i.e., $d_i(\tau) = 1/|S| \sum_{k=1}^{|S|} d(c_k)$) at each time step as a loss sample (i.e., it is equivalent to a negative reward $-r_i(\tau)$) of the state and agent's action. Every time when the replanning is executed, the online planner will forward: a) *current queue length*, b) *neighbor arrival information*, c) *average delay* $d_j(\tau)$ and d) *maximum green time* $\mathbf{G}_j$ from neighbor scheduling agents to upper reinforcement learner. After obtaining the observations within T steps, the policy μ_θ outputs maximum green time $\mathbf{G}_i$ that will be applied for next T time steps. The learned policies minimize the expected shared delay for each agent respectively.

To be efficient in terms of both data size and computation for longer time-scale, we compact the samples that constitute the state and loss by taking averages. $\hat{\mathbf{q}}_i$ and $\hat{\mathbf{c}}_i \in \mathbb{R}^P$ denote the sample mean vectors with P dimension (i.e., P queues) within T time steps such that

$$\hat{\mathbf{q}}_i = [\hat{q}_{1,i}, \cdots, \hat{q}_{P,i}], \tag{7}$$

where $\hat{q}_{p,i} = \frac{1}{T}(q_{p,i}(t) + \cdots + q_{p,i}(t + T - 1)$. It is similar for the case of $\hat{\mathbf{c}}_i$. On the other hand, *loss function* (shared delay) is obtained through averaging among neighbors and local agent: $\hat{l}_i = \frac{1}{T\times(|\mathcal{N}_i|+1)} \sum_{j\in\mathcal{N}_i\cup i} \sum_{\tau=t}^{t+T} d_j(\tau)$.

Since each agent learns the policy independently, each agent needs to maintain an approximation of centralized critic, i.e., $\sum_{j\in\mathcal{N}_i} Q^{\boldsymbol{\mu}}_{i,j}(s, a_i, a_j)$, by itself. To enable the decentralized learning, we define the following approximated network state for agent i:

Definition 1 (Approximated Network State of Agent i)**.** *The concatenation of local queue information and neighbor arrival information, which is $[\hat{\mathbf{q}}_i, \hat{\mathbf{c}}_i]$, is defined as approximated network state for agent i.*

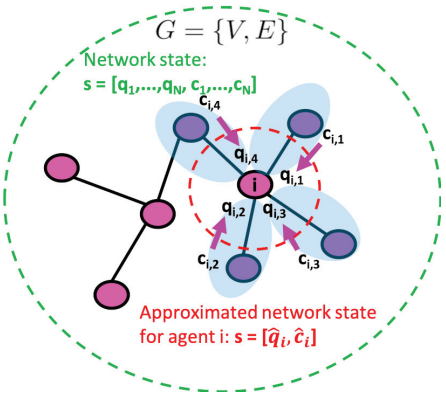


Figure 3: The approximated network state for agent i is composed of $\hat{\mathbf{q}}_i$ and $\hat{\mathbf{c}}_i$, where $\hat{\mathbf{q}}_i$ is the time average of local queue length $\mathbf{q}_i(t)$ and $\hat{\mathbf{c}}_i$ is the time average of the communicated number of arriving vehicles $\mathbf{c}_i(t)$.

Hence, to approximate the gradient of the centralized critic, each agent takes approximated network state $s \equiv [\hat{\mathbf{q}}_i, \hat{\mathbf{c}}_i]$, $a_i \equiv \mathbf{G}_i$ and $a_j \equiv \mathbf{G}_j$ as the inputs of action-value function:

$$\sum_{j\in\mathcal{N}_i} Q^{\boldsymbol{\mu}}_{i,j}(s, a_i, a_j) = \sum_{j\in\mathcal{N}_i} Q^{\boldsymbol{\mu}}_{i,j}([\hat{\mathbf{q}}_i, \hat{\mathbf{c}}_i], \mathbf{G}_i, \mathbf{G}_j), \tag{8}$$

The approximated network state for agent i in a graph G is illustrated in Figure 3.

Counterfactual Deterministic Baseline Since we consider the problem of searching for policies to maximize an estimated network-level utility function, the gradient computed for each intersection does not explicitly reason about how the actions made by a particular agent contribute to the utility function and becomes noisy for multiple neighbors. Therefore, we adopt the concept of *counterfactual baseline*, in which each agent learns from a shaped reward. Compared to the stochastic case, the deterministic case cannot marginalize out $\mathbf{G}_i$ through averaging over actions. Hence, we apply *time-averaged* value function over the historical actions of agent i as the baseline to compute the advantage function:

$$A^{\boldsymbol{\mu}}_i(s, a_i, a_{\mathcal{N}_i}) = \sum_{j\in\mathcal{N}_i} Q^{\boldsymbol{\mu}}_{i,j}([\hat{\mathbf{q}}_i, \hat{\mathbf{c}}_i], \mathbf{G}_i, \mathbf{G}_j) - \frac{1}{KT} \sum_{k=1}^{K} \sum_{j\in\mathcal{N}_i} Q^{\boldsymbol{\mu}}_{i,j}([\hat{\mathbf{q}}_i, \hat{\mathbf{c}}_i], \mathbf{G}^{t-kT}_i, \mathbf{G}_j) \tag{9}$$

, where $a_{\mathcal{N}_i}$ represents the set of neighbor actions and K is the number of saved historical actions. If we also replace s of $\mu_{\theta_i}(s)$ with $[\hat{\mathbf{q}}_i, \hat{\mathbf{c}}_i]$, the gradient of the advantage function with respect to agent i can be written as:

$$\nabla_{\theta_i} J(\boldsymbol{\mu}) = \mathbb{E}_{s\sim\mathcal{D}_i}\left[\nabla_{\theta_i}\mu_{\theta_i}(s) \nabla_{\mathbf{G}_i} A^{\boldsymbol{\mu}}_i(s, a_i, a_{\mathcal{N}_i})|_{a_i=\mu_{\theta_i}(s)}\right] \tag{10}$$

The following lemma establishes the convergence of this gradient to a local optimal value of (8). The proof follows directly from the deterministic gradient policy theorem (Silver et al. 2014).

Proposition 2. *For a full decentralized actor-critic algorithm following (10) at each iteration, the global performance converges to a local optimal value.*

Proof. Since the deterministic gradient of the time-averaged baseline function does not depend on current $\mathbf{G}_i^t$ that is a function of state $[\hat{\mathbf{q}}_i, \hat{\mathbf{c}}_i]$, then we get

$$\mathbb{E}_{s\sim\mathcal{D}_i}\big[\nabla_{\theta_i}\frac{1}{KT}\sum_{k=1\cdots K}\sum_{j\in\mathcal{N}_i} Q^{\boldsymbol{\mu}}_{i,j}([\hat{\mathbf{q}}_i,\hat{\mathbf{c}}_i],\mathbf{G}_i^{t-kT},\mathbf{G}_j)\big]=0.$$

Thus, (10) is equivalent to the DPG of (8). □

Here the experience replay buffer $\mathcal{D}_i$ contains the tuples $(s', s, a_i, a_{\mathcal{N}_i}, \hat{l}_i) \equiv ([\hat{\mathbf{q}}'_i, \hat{\mathbf{c}}'_i], [\hat{\mathbf{q}}_i, \hat{\mathbf{c}}_i], \mathbf{G}_i, \mathbf{G}_{\mathcal{N}_i}, \hat{l}_i)$ for the agent i. The decentralized action-value function $A_i^{\boldsymbol{\mu}}$ is updated as:

$$\mathcal{L}(\theta_i) = E\big[(A_i^{\boldsymbol{\mu}}(s, a_i, a_{\mathcal{N}_i}) - y)^2\big], \tag{11}$$
$$y = -\hat{l}_i + \gamma A_i^{\boldsymbol{\mu}}(s', a'_i, a'_{\mathcal{N}_i})|_{a'_i=\mu_{\theta_i}(s)}$$

Inferring Policies of Neighbor Agents To remove the assumption of knowing neighbors' configured $\mathbf{G}_{\mathcal{N}_i}$ required in (11), each agent can adopt an online regression algorithm to estimate neighbors' actions, e.g., online passive-aggresive regression algorithm (Crammer et al. 2006). We take the current action, i.e., $\mathbf{G}_j$, to predict $\hat{\mathbf{G}}'_j = \mathbf{w}_t \cdot \mathbf{G}_j$ and update the weight vector after receiving the actual $\mathbf{G}'_j$ according to: $\mathbf{w}_{t+1} = \mathbf{w}_t + sign(\mathbf{G}'_j - \hat{\mathbf{G}}'_j)\tau_t \mathbf{G}_j$, where $\tau_t = l_t/\|\mathbf{G}_j\|^2$, $l_t = \max(|\mathbf{w}_t \cdot \mathbf{G}_j - \mathbf{G}'_j| - \epsilon, 0)$and ϵ is a positive parameter that controls the sensitivity of predictions.

We learn the weights of actor-critic network and apply planning at different time-scales. Local queue information and cluster information from neighbor intersections are collected at every time step after replanning, but the parameters of scheduling model are only updated after T time steps. The new $\mathbf{G}_i$ is drawn in an $\epsilon-$greedy fashion with the exploration variance v annealing as learning proceeds. In the planner, the delay and observation of replanning are collected at every time step.

Experimental Evaluation

To evaluate our approach, we simulate performance on a two-intersection model and a real world network. The two-intersection model is for studying how different parameters affect performance. We compare parameters that is configured on-line by a learned policy with fixed parameters. The real world network is for evaluating the performance of hierarchical approach in a larger complex real network. The simulation model was developed in VISSIM, a commercial microscopic traffic simulation software package. We assume that each vehicle has its own route as it passes through the network and measure how long a vehicle must wait for its turn to pass through the intersections (the delay). In our experiments, since one traffic phase may be composed of multiple lanes (queues), we use a composite queue that contains multiple queues corresponding to a specific traffic phase as a feature of the state. Tested traffic volume is averaged over sources at network boundaries. To assess the performance boost provided by the proposed algorithm, we measure the average waiting time of all vehicles over ten runs. All simulations run for 3.5 hour of simulated time. Results for a given experiment are averaged across all simulation runs with different random seeds.

Two-Intersection Model We consider a simple two-intersection model with 2-way, multiple lanes, and multi-directional traffic flow as controlled experiments. The source of traffic are assumed to be stationary and set to 2800 cars/hour. In this simple model, there is only one connecting road segment.

Two different RL algorithms are implemented. First, we base our DDPG for continuous actions, where both actor and critic network have two 30-unit hidden layers, and the chosen activation unit is rectified linear unit (ReLU). A single network is used to approximate critic (8) by incorporating neighbor actions. For deterministic policy, we apply $tanh$ as output function to constrain output within $[-1, 1]$. We set $\gamma = 0.99$, learning rate $\alpha = 0.001$ for actor and $\alpha = 0.002$ for critic. Experience replay is applied with the size = 500 and the replace frequency of updating the two networks are calculated using exponential moving average (EMA) over the time steps necessary for each traffic phase. For both algorithms, we train the model for 10 simulation hours and test the performance for 1 simulation hour over 10 runs with different random seeds. Here two phase cycles are referred to be the long time-scale T, and length of history $K = 2$ for computing advantage function. Second, we apply the double Q-network with experience replay and frozen target. Both action-value functions Q_1 and Q_2 share same network structure, which has three hidden layers with $15, 20, 10$ units for each. The actions comprises three outputs corresponding to $[60s, 90s, 120s]$. Other setting is same as DDPG.

In Table 1, we can observe that for a fixed traffic condition (2800 vehicles/hour), choosing different parameters could provide a substantial reduction on average delay. For instance, the delay of benchmark parameter *(50s, 90s)* is lower than *(50s, 60s)* by nearly 25%. The table shows how the delay is affected by choosing different parameters, and there exists an optimal parameter values. DQN and DDPG approaches generally outperform fixed parameter settings we manually set. Furthermore, applying deterministic policy is able to have better performance compared to discrete actions (DQN).

Urban Network Model We compare our hierarchical algorithm to two other real-time traffic control methods: 1) First, we take the performance of the original *schedule-driven traffic control system* (Xie, Smith, and Barlow 2012) as our baseline system. 2) Second, we compare to a variant of *cycle-based adaptive control* that optimizes cycle time, phase split and timing offset of successive signals every cycle. The basic concept of cycle-based adaptive control is to calcu-

Table 1: Summary of two intersection model results

	Average Delay (second)		No. of stops
	mean	std.	mean
DQN	63.78	53.35	1.75
DDPG	**57.80**	**47.86**	**1.59**
Bench (50s, 60s)	89.62	78.64	2.91
Bench (50s, 90s)	67.66	55.50	1.91
Bench (50s, 120s)	73.49	62.18	2.34

late cycle time based on estimation of saturation flow rate (Webster 1958) and allocate green time according to flow ratio on each phase. A well known of this type of adaptive control scheme is SCATS system (DAIZONG 2003; Wongpiromsarn et al. 2012). [1]

Figure 4: Map of the 24 intersections in the Baum-Centre neighborhood of Pittsburgh, Pennsylvania

The network model is based on the Baum-Centre neighborhood of Pittsburgh, Pennsylvania as shown in Figure 4. The network consists of 24 intersections that are mainly 2-phased. It can be seen as a two-way grid network. To explore how the proposed algorithm performs under different demand, we categorize traffic demand into three different groups. We use highest and lowest averaged traffic volume from actual traffic data observed for the Baum-Centre traffic network as our high and low demand levels, and designate the middle point of these two values as the medium demand, which are: low (472 cars/hour), medium (708 cars/hour), and high (1056 cars/hour). Two phase cycles are still referred to be T, and $K = 2$. Episodic RL is applied in this model to tackle irreducible state transition since high demand scenario could cause a "terminate" state in which queuing stability is lost, queue length increases without bound, and the system is not able to return the stable states. With this situation, we need to reset our training by starting a new episode. Here one episode is defined as 3.5 hours corresponding to a period of peak hour. Similarly, we train the model for 35 simulation hours (10 episodes) and test the performance for 3.5 simulation hours over 10 runs with different random seeds. The delay corresponding to different episodes are plotted in Figure 5. It shows a decreasing trend along with increasing episode number.

[1]Note also that previous research with the baseline schedule-driven approach has shown its comparative advantage over other online planning approaches (Xie, Smith, and Barlow 2012).

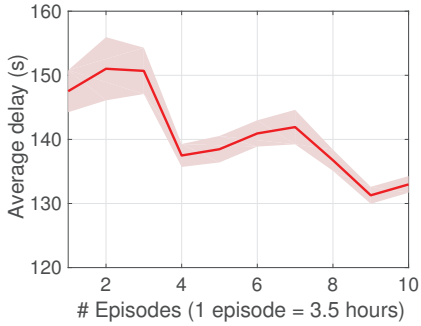
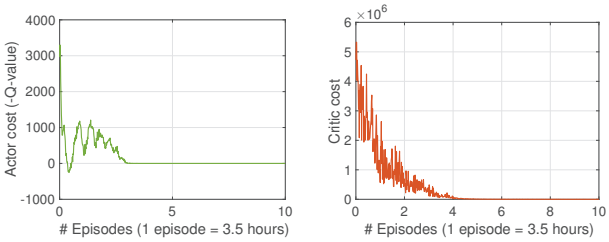


Figure 5: The average delay for each episode



(a) Actor cost (b) Critic cost

Figure 6: Costs of actor and critic under *high traffic demand*.

The model architecture and setting are the same as two-intersection model other than an enlarged experience buffer size $= 1000$. Table 2 shows the hierarchical approach to yield an improvement over the schedule-driven approach of about 30% and the cycle-based adaptive control of about 35% for the high traffic demand case. For low and medium traffic, the average delay of the three approaches are comparable. Since we use the queue length and the number of arrival vehicles as state to configure parameters, low and medium may not have noticeable state transition to adjust policy. Especially, the parameter we choose to optimize is maximum green time. Under low and medium demand, the scheduler is already able to allocate green time efficiently. If we choose different parameter to optimize, it may express different behavior under low and medium demand. In addition, the learning costs of two neural networks are shown in Figure 6. We can observe that after 4th episode the costs already converge. A visualization of testing policy function is also plotted in Figure 7 and shows a non-linear relationship between the queue length and the parameters.

We also test the hierarchical algorithm under non-stationary traffic demand, which is closer to real-world scenarios. All simulation runs were carried out according to a realistic traffic pattern from late afternoon through "PM rush" (4-6 PM). The traffic pattern ramps up volumes over the simulation interval as follows: (0-30mins: 472 cars/hour, 30min-1hour: 708 cars/hour, 1hour-2.5hours: 1056 cars/hour, 2.5hours-3hours: 708 cars/hour, 3hours-3.5hours: 472 cars/hour,). This simulation model presents a complex practical application to verify the effectiveness of the pro-

	Average Delay (second) and Number of Stops								
	Benchmark			Hierachical			Cycle-based Adaptive		
	mean	std.	stop no.	mean	std.	stop no.	mean	std.	stop no.
High demand	212.14	361.41	9.55	**132.98**	**92.95**	**6.76**	230.26	279.19	12.34
Medium demand	84.22	61.90	6.34	**82.56**	**55.84**	**4.56**	86.46	61.40	8.78
Low demand	71.84	54.25	6.12	**72.10**	**49.11**	**4.23**	73.89	56.77	8.11
PM rush hour	147.00	177.94	8.27	**113.89**	**88.24**	**5.10**	169.23	265.91	10.81

Table 2: Average delay under different scenarios.

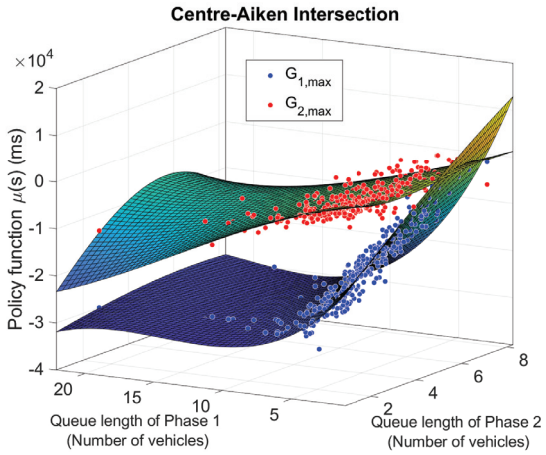


Figure 7: Visualization of policy $\mu(s)$ at two-phased Centre-Aiken intersection

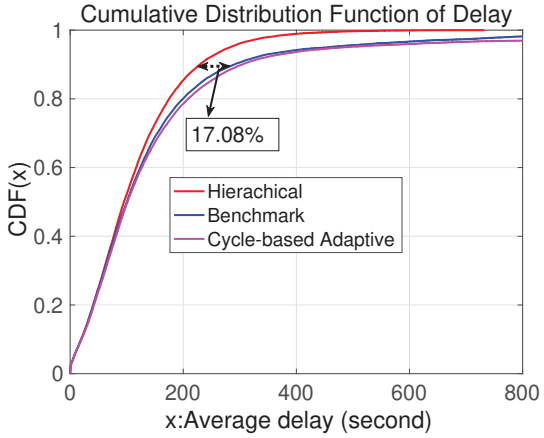


Figure 8: The cumulative distribution function of delay.

posed approach. The simulation time, T and K are same as the categorized cases.

Table 2 also shows the results of hierarchical approach under PM rush, compared to cycle-based adaptive control approach and the baseline schedule-driven approach. As can be seen, delay is reduced by 23.13% and 33.14%, compared to the schedule-driven and adaptive control approaches respectively. The use of learned policy reduces delay by balancing the maximum green time among phases. The learned policy is also able to coordinate intersections through incorporating neighbor schedule information. In addition to delay, the number of stops of hierarchical approach is nearly half of cycle-based adaptive control. Note that using learned policy to configure G_{max} is actually similar to the effect of clearing queues of waiting vehicles and reducing the deleterious effects of spillback (Daganzo 1998) by stopping vehicles further away from entry into a road segment with insufficient capacity.

Since traffic conditions are dynamically changing, knowing the distribution of delay to vehicles helps us verify the effectiveness of the proposed hierarchical algorithm. As shown in Figure 8, using learned policy shifts the cumulative distribution function (CDF) leftward and provides a 17.08% improvement over the schedule-driven approach for 90% of the vehicles. Note also that while the proposed approach reduces average delay by $40s$, the reduction is more than $100s$ for the congested vehicles. In other words, configuring maximum green time correctly is especially effective for high congestion scenarios. In comparison to adaptive control, it provides a 18.26% delay reduction for 90% of the vehicles.

Conclusion

In this work, we considered the limitations of prior approaches to schedule-driven traffic control that rely on a fixed model without regard to potential consequences due to incorrect configuration of parameters. A fully decentralized hybrid algorithm is proposed to achieve better network-level performance in circumstances of high traffic demand. In this algorithm, each agent has two hierarchical modules, where the lower-level module is responsible for generating continuously generating schedules to control the traffic signal, and the upper-level module learns a policy for configuring model parameters, based on the samples collected from the lower-level module. Briefly, the planning module focuses on short-term traffic variation, while the learning module focuses on long-term performance. Moreover, learning and planning achieve better performance through exchanging information, i.e., online planning utilizes neighbor schedule information to extend the look-ahead horizon; reinforcement learning generates better coordinating policies by including neighbor information as an explicit state feature. Experimental results showed that the hierarchical model improves cumulative delay overall in comparison to both the baseline schedule-driven traffic control approach and a cycle-based adaptive traffic signal control approach, and that solutions

provide substantial gain in highly congested scenarios.

References

Bazzan, A. L., and Klügl, F. 2014. A review on agent-based technology for traffic and transportation. *The Knowledge Engineering Review* 29(3):375–403.

Cai, C.; Wong, C. K.; and Heydecker, B. G. 2009. Adaptive traffic signal control using approximate dynamic programming. *Transportation Research Part C: Emerging Technologies* 17(5):456–474.

Covell, M.; Baluja, S.; and Sukthankar, R. 2015. Micro-auction-based traffic-light control: Responsive, local decision making. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, 558–565. IEEE.

Crammer, K.; Dekel, O.; Keshet, J.; Shalev-Shwartz, S.; and Singer, Y. 2006. Online passive-aggressive algorithms. *Journal of Machine Learning Research* 7(Mar):551–585.

Daganzo, C. F. 1998. Queue spillovers in transportation networks with a route choice. *Transportation Science* 32(1): 3–11.

DAIZONG, L. 2003. *Comparative evaluation of dynamic TRANSYT and SCATS-based signal control systems using Paramics simulation*. Ph.D. Dissertation.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research* 13:227–303.

Dresner, K., and Stone, P. 2008. A multiagent approach to autonomous intersection management. *Journal of artificial intelligence research* 31:591–656.

Foerster, J. N.; Farquhar, G.; Afouras, T.; Nardelli, N.; and Whiteson, S. 2018. Counterfactual multi-agent policy gradients. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Gartner, N.; Pooran, F.; and Andrews, C. 2002. Optimized policies for adaptive control strategy in real-time traffic adaptive control systems: Implementation and field testing. *Transportation Research Record: Journal of the Transportation Research Board* (1811):148–156.

Gettman, D.; Shelby, S.; Head, L.; Bullock, D.; and Soyke, N. 2007. Data-driven algorithms for real-time adaptive tuning of offsets in coordinated traffic signal systems. *Transportation Research Record: Journal of the Transportation Research Board* (2035):1–9.

Guestrin, C.; Lagoudakis, M.; and Parr, R. 2002. Coordinated reinforcement learning. In *ICML*, volume 2, 227–234. Citeseer.

Heung, T. H.; Ho, T. K.; and Fung, Y. F. 2005. Coordinated road-junction traffic control by dynamic programming. *IEEE Transactions on Intelligent Transportation Systems* 6(3):341–350.

Jonsson, A., and Rovatsos, M. 2011. Scaling up multiagent planning: A best-response approach. In *ICAPS*.

Kulkarni, T. D.; Narasimhan, K.; Saeedi, A.; and Tenenbaum, J. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, 3675–3683.

Kuyer, L.; Whiteson, S.; Bakker, B.; and Vlassis, N. 2008. Multiagent reinforcement learning for urban traffic control using coordination graphs. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 656–671. Springer.

Lowe, R.; Wu, Y.; Tamar, A.; Harb, J.; Abbeel, O. P.; and Mordatch, I. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, 6379–6390.

Lowrie, P. 1992. Scats - sydney co-ordinated adaptive traffic system - a traffic responsive method of controlling urban traffic. Roads and traffic authority, sydney, nsw, australia.

Nunes, L., and Oliveira, E. 2004. Learning from multiple sources. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 3*, 1106–1113. IEEE Computer Society.

Sen, S., and Head, K. L. 1997. Controlled optimization of phases at an intersection. *Transportation science* 31(1):5–17.

Shelby, S. G. 2001. Design and evaluation of real-time adaptive traffic signal control algorithms.

Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; and Riedmiller, M. 2014. Deterministic policy gradient algorithms. In *ICML*.

Smith, S. F.; Barlow, G. J.; Xie, X.-F.; and Rubinstein, Z. B. 2013. Smart urban signal networks: Initial application of the surtrac adaptive traffic signal control system. In *ICAPS*. Citeseer.

Webster, F. V. 1958. Traffic signal settings. Technical report.

Wiering, M. 2000. Multi-agent reinforcement learning for traffic light control. In *ICML*, 1151–1158.

Wongpiromsarn, T.; Uthaicharoenpong, T.; Wang, Y.; Frazzoli, E.; and Wang, D. 2012. Distributed traffic signal control for maximum network throughput. In *2012 15th International IEEE Conference on Intelligent Transportation Systems*, 588–595. IEEE.

Wu, F.; Zilberstein, S.; and Chen, X. 2009. Multi-agent online planning with communication. In *ICAPS*.

Xie, X.-F.; Smith, S. F.; and Barlow, G. J. 2012. Schedule-driven coordination for real-time traffic network control. In *ICAPS*.

Zhang, K.; Yang, Z.; Liu, H.; Zhang, T.; and Başar, T. 2018. Fully decentralized multi-agent reinforcement learning with networked agents. *arXiv preprint arXiv:1802.08757*.