Adaptive Traffic Signal Control With Deep Reinforcement Learning and High Dimensional Sensory Inputs: Case Study and Comprehensive Sensitivity Analyses

Soheil Mohamad Alizadeh Shabestary and Baher Abdulhai, *Member, IEEE*,

***Abstract*— Despite the constant rise in global urban populations and subsequent rise in transportation demand, significant expansion of infrastructure has been hampered by the constraints of space, cost, and environmental concerns. Therefore, optimizing the efficiency of existing infrastructure is becoming increasingly important. Adaptive traffic signal controllers aim to provide demand-responsive strategies to minimize motorists' delay and achieve higher throughput at signalized intersections. With the advent of new sensory technologies and more intelligent control methods, the contribution of this paper is an adaptive traffic signal controller able to receive un-preprocessed high-dimensional sensory information such as GPS traces from connected vehicles and self-learn to minimize intersection delays. We use deep neural networks to operate directly on detailed sensory inputs and feed them into a reinforcement learning-based optimal control agent. The integration of these two components is known as deep learning. Using deep learning, we achieve two goals: (1) We eliminate the need for handcrafting a feature extraction process such as determining queue lengths, which is challenging and location-specific, and (2) we achieve better performance and faster training times compared to conventional tabular reinforcement learning approaches. We test our proposed controller against a tabular reinforcement learning agent, a reinforcement learning agent with a fully-connected Neural Network as a function approximator, and a state-of-practice, actuated traffic signal controller.**

***Index Terms*— Deep learning, reinforcement learning, adaptive traffic signal control.**

I. Introduction

WITH growing urbanization, the demand for transportation is constantly rising. Congestion not only costs people valuable time in traffic, but also results in fuel overconsumption and pollution. Constructing new infrastructure to offset these issues is often impractical due to monetary and space limitations as well as environmental and sustainability concerns. Therefore, a viable option to increase the capacity of urban transportation networks is to use technology that maximizes the performance of existing infrastructure.

Manuscript received 22 April 2019; revised 29 October 2019, 21 April 2020, 25 November 2020, and 25 April 2021; accepted 9 May 2022. Date of publication 14 June 2022; date of current version 7 November 2022. The Associate Editor for this article was H. Dia. *(Corresponding author: Soheil Mohamad Alizadeh Shabestary.)*

Soheil Mohamad Alizadeh Shabestary was with the Civil and Mineral Engineering Department, University of Toronto, Toronto, ON M5S 1A4, Canada. He is now with Huawei Technologies Canada, Markham, ON L3R 5A4, Canada (e-mail: soheil.shabestary@huawei.com).

Baher Abdulhai is with the Civil and Mineral Engineering Department, University of Toronto, Toronto, ON M5S 1A4, Canada (e-mail: baher.abdulhai@utoronto.ca).

Digital Object Identifier 10.1109/TITS.2022.3179893

Optimizing traffic signals decreases delays for drivers in urban networks. The most common approaches for signal control are fixed-time and actuated. Fixed-time controllers use historical traffic data to optimize traffic signals offline and do not change when deployed [1]–[5]. Actuated signal controllers, on the other hand, are more responsive to traffic flows, as they receive feedback from sensors. However, they do not explicitly optimize delay.

Adaptive traffic signal controllers (ATSC) are more advanced and can outperform other controllers. Their superior performance is based on their ability to constantly modify the signal timings to optimize their objective [6]. Some ATSCs, such as SCOOT [7], SCATS [8], PRODYN [9], OPAC [10], UTOPIA [11] and RHODES [12], optimize signals by using an internal model of the environment. These models are often simplistic and rarely up-to-date with current conditions, and the optimization algorithms are mostly heuristic and sub-optimal.

Due to the stochastic nature of traffic and driver behavior, it is difficult to devise a precise traffic model. The models that are more realistic are also more sophisticated and harder to control. Hence, there is a trade-off between the complexity and practicality of the controllers. Furthermore, traffic control is a sequential decision-making problem, meaning that every decision affects future traffic conditions. The complexity of the problem rules out a straightforward computation of the optimal solution. With the advent of Reinforcement Learning (RL), however, there have been some major improvements in this area, as RL is best suited for this kind of problem [13]–[20]. Specifically, RL algorithms can learn an optimal control strategy while interacting with the environment and evaluating their own performance [21].

Conventional RL methods are designed in tabular format and for relatively small state-action space problems. When the state-action combination is very large it is possible that some state-action cells in the Q-table might not be visited enough or at all. In such a case, the controller may encounter unknown states and has to explore. Therefore,

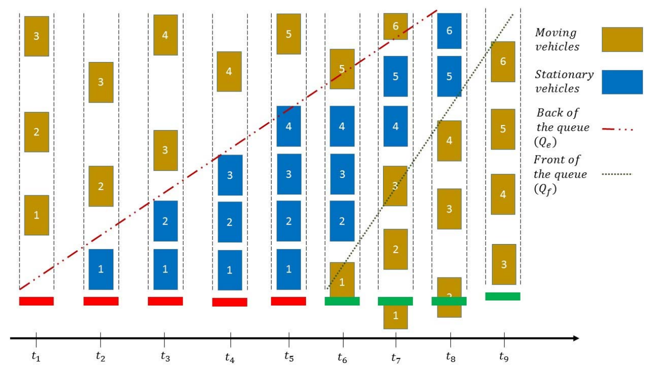


Fig. 1. Aerial snapshots of one street with one lane leading to the intersection at different time steps. The blue vehicles are stopped in the queue and the orange vehicles are moving. From time step t_1 to t_9, the vehicles approach the intersection facing the red signal; the queue starts to build up; the signal turns green and vehicles start to move; the queue starts to disappear. The red dashed line indicates the back of queue and the green dashed line shows the front of the queue. The distance between the two lines is the queue length.

RL with function approximation such as DQN would be more appropriate for such large state-action space problems. The function approximator or the neural network would enable better interpolation and generalization.

In other words, tabular RL methods are highly sensitive to the size of their state space, since they use discretization techniques for continuous state variables and present them in a tabular format. They also suffer from a phenomenon known as the curse-of-dimensionality in environments with a large state space. Recently, there have been major advances in RL with function approximation [22]. Function approximation techniques allow us to use RL methods in continuous and large state spaces. However, they have to be treated carefully, since they cause instability when integrated with RL methods [23].

Most RL-based traffic signal controllers use simple and pre-processed information as input to optimize a traffic signal. Although the most common state space used in ATSC controllers is the queue length at the intersection, there are many promising RL-based ATSCs that used different state space definition [24]–[26], which shows the lack one ultimate pre-processed state space definition in ATSC. In addition, in Fig. 1 we see the buildup and discharge of the queue on a generic street, where the queue lengths at both t_3 and t_8 are the same, but the condition of traffic is completely different. Fig. 1. shows how using queue length as the state space might produce confusion for the controller. Meanwhile, state-of-the-practice traffic sensors are rapidly developing to provide high-resolution, more accurate, and more detailed information at the level of tracking individual motorists and pedestrians. It is important to design an agent capable of processing new high-dimensional traffic information without the help of an expert to generate pre-processed inputs and without getting computationally overwhelmed. Detailed information gives the controller better knowledge of intersections, enabling it to make better decisions and optimize travel times more efficiently.

A subset of RL with function approximation known as Deep Reinforcement Learning has only recently matured. It has the ability to process large state space problems and achieve better performance compared to other RLs with function approximation methods. In [18] and [19], the surface of the street is discretized into small cells and all the cells are put together to create a matrix of positions and speeds of the vehicles approaching the intersection. The authors in [18] propose a deep Q-learning algorithm to control the traffic signal at a generic intersection. They use two CNN networks: one to process a position, and the other for the speeds of the approaching vehicles. However, analyzing the position and speed matrices separately ignores the correlation between the two. Moreover, the testbed in [18] is relatively simple, with an unrealistic configuration for the intersection and no minimum green time for signal phases. The study in [19] addresses some of the shortcomings in [18] by using an experience replay memory mechanism and processing the position and speed matrix together.

Another promising approach to take advantage of the deep reinforcement learning in literature is in form of controlling large-scale traffic networks [24]–[26]. These papers show the potential of DRL in controlling multiple intersections; however, they use complicated preprocessed state spaces. The focus of this research is on eliminating the need for handcrafting features as the state space, being mindful of the soon-to-be-available vast sensory information with increasing popularity of connected and autonomous vehicles.

In the present work, we build on and expand the promising deep reinforcement learning approach to traffic signal control on a simulation testbed of a real traffic network. We also propose a more mature system for practical use and conduct comprehensive experimental analyses. We develop our system using a specific deep learning agent called Deep Q-Network (DQN) [27]. It combines Deep Neural Networks (DNNs) with Q-learning algorithms, which is one of the most common RL techniques. DQN is initially designed to receive images as input and learn the optimal policy that optimizes its objective function. To apply DQN to a traffic signal control problem, we rearrange the structure of traffic sensory inputs to be presentable in the form of a two-dimensional array of pixel-like values (such as in an image) and tune the details of DQN to be more suitable for our traffic signal control problem. We call our controller the Deep Q Traffic Signal Controller (DQTSC).

We test our novel DQTSC in Paramics traffic micro-simulator [28] for a small neighborhood in the City of Burlington, ON, Canada. The proposed controller is tested under various practical limitations facing traffic signal controllers. We then present the improvements achieved by our deep controller and compare the advantages of this new technique to a Q-learning traffic signal controller with a shallow fully-connected Neural Network function approximator (SQTSC), a tabular Q-learning traffic signal controller (TQTSC), a state-of-practice, actuated traffic signal controller, and a fixed-time traffic signal controller.

In the rest of this paper, we first introduce the details of our proposed controller in section II. In section III, we present our testbed and provide a set of experiments to test the traffic signal controllers in section IV. We present the simulation results in section V, and in section VI, we conclude the results and observations from the experiments.

II. Methodology

In the following section we explain the controllers used in this study, which includes DQTSC, SQTSC, Actuated, and fixed time. It should be noted that the main difference between the DQTSC and SQTSC is the resolution of the state space and the type of the neural network. In DQTSC we use high resolution image like state representation which is processed by a deep neural network comprising a CNN followed by a fully connected neural network, while the SQTSC takes as input simple queue lengths and uses a shallow fully connected neural network.

A. Deep Q Traffic Signal Controller (DQTSC)

To build the controller, we use a Deep Q-Network (DQN) [27], which combines Q-learning algorithm with a deep convolutional network. Deep convolutional networks are known for their ability to handle large-sized, high-dimensional inputs like images. We chose DQN considering that the state-of-practice traffic sensors, e.g., radars and connected vehicles, are able to provide the detailed information for each individual vehicle approaching the intersection [29], [30], and we want to avoid any information loss during the state definition pre-processing. More specifically, we want to avoid the issues pertinent to defining, measuring, and dynamically tracking queue lengths as inputs to the controller. If the traffic controller can handle and process the available detailed high-dimensional input, we eliminate the need for an expert to compress this information to low-dimensional state features such as queue lengths. We also eliminate the possibility of losing information during this process.

DQN is suitable for traffic signal control problems, as it does not require a model of the environment and learns the optimal strategy to control the signal by interacting with the environment. Like any other RL agent, DQN observes the states (s_t) of the environment provided by the sensors and decides to take an action (a_t). After applying the action to the environment, the states of the system change, and the agent receives the new states (s_{t+1}) and the reward (r_t) – a scalar signal indicating the goodness of the last action. Since the traffic signal control problem is a sequential decision-making process, the goal of the controller is to select an action at each time step that maximizes cumulative future rewards. We approximate the cumulative future reward with the state-action value function (Q function). In so doing, we can estimate the expected cumulative future reward with state-action value function, known as the Q function.

$$Q^*(s_t, a_t) = \max_{\pi} E\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots | \pi\right]. \quad (1)$$

Equation 1 shows the optimal action-value function, in which γ is the discount factor, indicating the difference between the values of the immediate reward and future reward. Here, $\pi = P(a|s)$ is the policy of the agent and shows the probability of each action to be chosen in given states. In the Q-learning algorithm, we assume that, except for the current action, all future actions are chosen to maximize the expected return:

$$Q^*(s_t, a_t) = E\left[r_t + \gamma \max_{a'} Q^*(s_{t+1}, a')\right]. \quad (2)$$

This equation is known as the Bellman equation for the Q-learning algorithm. Our goal is to estimate the action-value function by interacting with the environment. In conventional RL, the action-value function is a discrete table. In DQN, the approximation of the action-value function is a deep neural network with parameters θ, $Q(s, a; \theta) \approx Q^*(s, a)$. A neural network can be trained to estimate the Q-function by minimizing a loss function L(θ),

$$L(\theta) = E\left[(y_t - Q(s_t, a_t; \theta))^2\right], \quad (3)$$

where $y_t = (r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta)$ is the target of the network. To decrease the instability of the algorithm, we incorporate two strategies: experience replay and periodic update of the target network. In the former, we store the interactions of the agent with the environment in a replay memory and during training randomly draw samples from the memory to train the agent, thereby decreasing the correlation between the training samples. In the latter, we use a different network for estimating the target value (target network), in which the weights of the target network (θ^-) are updated with a lower frequency with respect to the actual Q-network weights (θ).

For more information regarding the details of Deep Reinforcement learning, please refer to [27], [31].

1) State Space: We use the position and speed of the vehicles in the vicinity of the intersection as inputs. We want our state to be presentable in the form of an image-like structure to utilize the unique properties of the convolutional networks. In order to present this information in a form similar to an image, we 'pixelate' the surface of the street into small partitions or cells, discretizing the length of each lane segment starting from the stop line into small cells of length d. We choose d to strike a balance between the precision and the size of the state space. A smaller d provides more detailed information about the traffic condition but increases the size of the state space. Reasonable values for d can be the average length of vehicles or their minimum length.

By placing the discretized lanes next to one another, we can create a table in which each cell represents a small segment of the road ending at the intersection. Each cell is filled with the number of vehicles in its corresponding segment. If there are n vehicle(s) on the street, the specific cell corresponding to that part of the street is filled with n; otherwise, there is 0. Therefore, we have a matrix filled with Whole Number ($W = \mathbb{N} \bigcup \{0\}$) for each leg of the intersection. In putting together these matrices for all the streets ending at the intersection, we have an image-like representation of the position of the approaching vehicles at the intersection. We then apply the same idea to speed, but instead of filling the cells with the number of vehicles, we fill the cells with the average speed of the vehicles inside the segment.

From this, we have two same-sized matrices containing all the information provided by the sensors. Similar to RGB

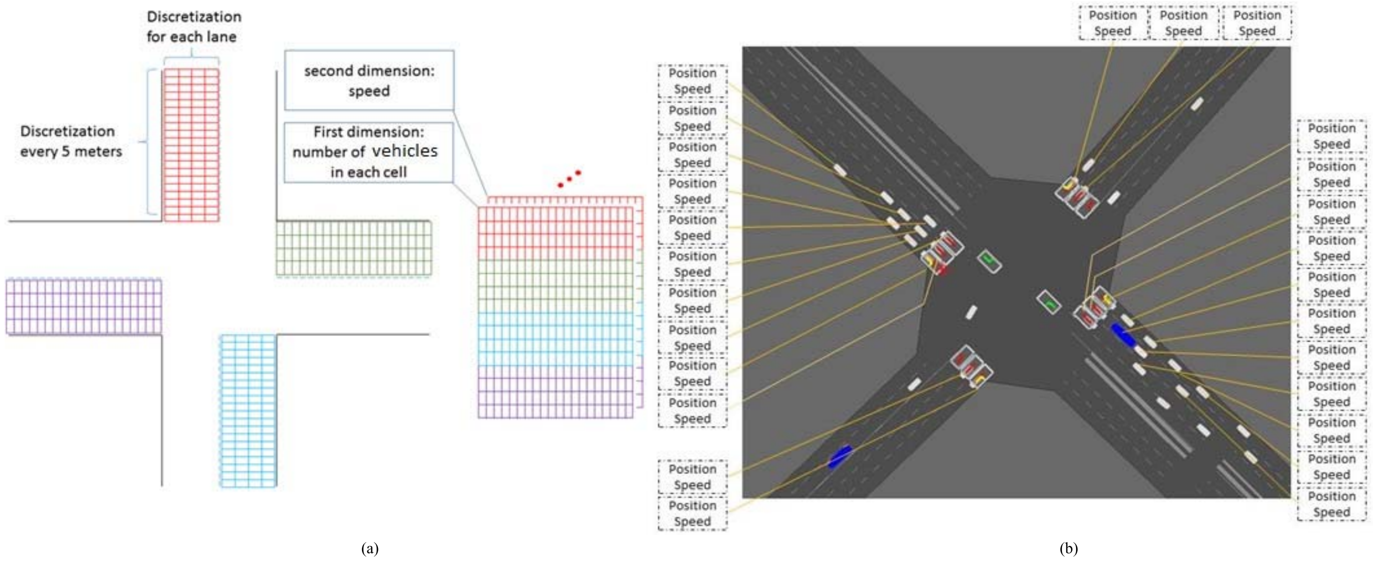

Discretization for each lane
Discretization every 5 meters
second dimension: speed
First dimension: number of vehicles in each cell
(a)
Position Speed
(b)

Fig. 2. (a) Illustration of pixelated streets showing pixels forming an image-like structure. (b) Screenshot of intersection in micro-simulator and information provided by the sensor.

images (where each image consists of three matrices, such that each cell presents the intensity of the basic colors — red, green, and blue — in a pixel), we fuse the two matrices to create 2-layer image-like input data and feed these data to our traffic signal controller (Fig. 2). It is important to note that with such input structure, there is no need to measure queue lengths, or to worry about sub-queues forming and breaking, or to worry about specifying a speed threshold to define when a vehicle is considered queued. Dealing with queues is challenging in the field, but DQN offers a valuable practical advantage over conventional tabular RL signal controllers that require queue length as input. In addition to the position and speed of the vehicles, it is useful for the controller to know the current green phase and the duration that the current phase has been green, which is referred to as elapsed time.

Next, we feed these two values (current green phase and elapsed time) along with the output of the convolutional neural network to a 2-layer fully connected neural network, which is a part of our DQN agent. For an intersection with n lanes, $S \in (W \times \mathbb{R})^{\frac{l}{d}} \times n \times P \times \mathbb{N}$, where l is the detection range, P is the set of possible phases, $\mathbb{R}$ and W denotes the set of all Real Numbers and the set of Whole Numbers, respectively, and N is the set of Natural Numbers representing elapsed time. In this state space definition, the data come directly from the sensors to the controller without any input pre-processing. Accessing the position and speed of vehicles is currently possible using radar technology [29] and will be even easier with the proliferation of connected vehicles [30].

However, it is possible that, in reality, a vehicle covers more than one cell on the surface of the street. This is not an issue for our state representation, because regardless of the sensory technology, each vehicle would be detected and associated with a single coordinate tuple (x, y, z), which places the vehicle in only one cell in the state space. In a situation where the street is shorter than the detection range, we would fill the cells exceeding the length of the street with zeros. Likewise, if some lanes were shorter than others (e.g., left-turn lanes shorter than through lanes), we would also fill the cells exceeding the length of the lane with zeros.

In Convolutional Neural Networks (CNNs), it is common to use square filters and equal striding in both directions, since CNNs are expected to process the input image no matter how the image is oriented [27]. In our case, however, the height and width of the input data represent different meanings. In Fig. 2, the width of the input represents the distance of the cell to the stop line, while the height of the input indicates the lane of the street that the cell is on. Each lane serves a different purpose; for example, two horizontally adjacent cells are different only in terms of their distances to the intersection and only by a value of d, while two vertically adjacent cells might represent substantially different meanings. For example, one might show a car waiting on the northbound movement, and the next cell might show a car waiting on the eastbound movement. This is essential information for the controller. Therefore, in the case of our problem, we prefer to analyze each and every lane separately and not to combine or skip the information in a vertical direction. We modify the standard DQN agent to feature rectangular filters that are narrow in height but longer in width, and we forbid striding on the vertical axis [31].

2) Action Space: The goal of the controller is to find the optimal strategy that maximizes its cumulative reward over time. In our case study, there are eight traffic movements: N, NL, E, EL, W, WL, S, and SL (Northbound, Northbound Left-turn, Eastbound, Eastbound Left-turn, Westbound, Westbound Left-turn, Southbound, and Southbound Left-turn, respectively). The action space includes all possible phases for the intersection, with each phase being a combination of non-conflicting movements A = {{NL, SL}, {N, NL}, {S, SL}, {S, N}, {EL, WL}, {E, EL}, {W, WL}, {E, W}}. At each time step, the controller can choose one of the possible actions in the action space ($a_t \in A$), at which point the movements indicated for each action will be given a green signal, while all the other movements will be set to red. Right turns are permitted in all phases, but with caution and a lower right of way. If the next action is the same as the current action, the green time of the current phase is extended by 1 second. However, if the next action is different from the current action, the signal has to go through three periods of yellow time for the current phase (all red time) and a minimum green time for the next phase until it would be able to take another action.

During this time, the controller is on hold while the information is being processed in the background, e.g., reward calculations. Note that the minimum green times might be different for each phase, as is the case in reality. After the hold time, the controller receives the reward signal and the new state of the environment to update its action value function and to decide its next action. If the current action and next action are different but share a movement, e.g., {NL, SL} and {N, NL}, the shared movement remains green while the other movement passes through yellow and red times.

We apply the actions of the controller to Paramics simulation in the following way: every time there is a phase change, we use Paramics API to set the green time of the chosen phase to a pre-defined maximum value, which usually never occurs in the optimal performance, e.g., 200 seconds. We then set the time for the other phases to zero. If the agent decides to extend the current phase at the next decision point (after the minimum green), there is no need to change the simulation because the current phase will stay green. However, if the agent decides to change the green signal to another phase, we cut the current green phase short, apply the yellow and all red time, and set the green time of the next chosen phase to the maximum green value, while ensuring all the other phase durations are set to zero.

3) Reward Function: In RL algorithms, the reward function is one of the most essential features. The controller learns to maximize its cumulative reward over time, so we have to define a reward function that is correlated to the objective of the problem. It usually takes multiple experiments to find a reward function that leads to the better performance of the controller.

In traffic signal control problems, many reward functions have been studied, including negative number of vehicles in the queue, vehicle throughput, and negative cumulative delay of the intersection. In this research, we use a modification of the reward function from [17], which is the average reduction (savings) in the cumulative delay of the intersection. The cumulative delay of the intersection is the aggregation of the total time spent in queue of all vehicles that are currently in the intersection control area, which includes all the streets ending at the intersection. Calculating the reward signal depends on whether the controller is on hold or not, as the reduction in the cumulative delay is averaged over the time that the controller is on hold. However, we show that this reward definition has performance and learning issues that we will discuss later in this paper. We also demonstrate that, by employing a seemingly minor modification of using reduction in the cumulative delay of the intersection (rather than averaged over time between decisions), as shown in equations 4 and 5 below, we resolve the learning issues and achieve better performance:

$$CD^k = \sum_{u \in VL^k} d_u^k, \tag{4}$$

$$r^k = CD^{k-1} - CD^k, \tag{5}$$

where r^k is the reward signal, CD^k is the cumulative delay of intersection, VL^k is the list of vehicles on streets leading to the intersection, and d_u^k is the delay of vehicle u, all at time step k. We call the agent and modified reward function DQTSC-M.

In this reward function, we need to track all the vehicles approaching the intersection and store their delays, i.e., the time each vehicle spends in the queue. However, this assumption is not practical for field implementation, as delay is not directly measurable. We propose a method to approximate the delay of each approach without having access to the actual delay of the vehicles. To use the approximation method, we only need the queue lengths $\left(q_m^t\right)$ and the output flows of the intersection $\left(O_m^t\right) m \in M$, where t is the time step of the environment, m represents the movement, and M is the set of movements at the intersection: $M = \{N, NL, E, EL, W, WL, S, SL\}$. For the approximation, we introduce an auxiliary variable z_m^t that represents the vehicles contributing to the delay of a movement:

$$z_m^t = \begin{cases} q_m^t, & if signal is red \\ & for movement m \\ & \\ z_m^{t-1} - O_m^t, & if signal is green \\ & for movement m \end{cases} \tag{6}$$

In this method, we track the number of vehicles in the queue when the traffic light is red. The delay of the movement has been built up because of these vehicles in the queue. When the signal turns green, we focus only on the vehicles that were in the queue during the red time and assume that the delay of the movement is divided among them equally. If O_m^t vehicles in the movement exit the intersection, there are still $z_m^{t-1} - O_m^t$ vehicles that are delayed. Consequently, the delay of the approach drops with the proportion of the vehicles remaining in the intersection to all the vehicles initially contributing to the movement delay. Hence, when O_m^t vehicles leave the intersection, the delay of the movement CD_m^t decreases by $\frac{O_m^t}{z_m^{t-1}}$.

$$\widehat{CD_m^t} = \begin{cases} \widehat{CD_m^{t-1}} + q_m^t, & if the signal is red \\ & for movement m \\ \left(1 - \frac{O_m^t}{z_m^{t-1}}\right).CD_m^{t-1} & if the signal is green \\ & for movement m \end{cases} \tag{7}$$

B. Shallow Q Traffic Signal Controller (SQTSC)

In SQTSC, instead of using high dimensional sensory information for individual vehicles, we use queue lengths of each approach {N, NL, E, EL, W, WL, S, SL} as input to the

TABLE I
SIGNAL PHASING CONFIGURATION

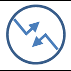
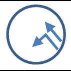
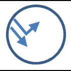
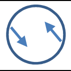
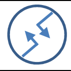
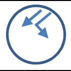
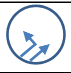
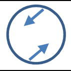
Phase 1	**Phase 2**	**Phase 3**	**Phase 4**	**Phase 5**	**Phase 6**	**Phase 7**	**Phase 8**
NL – SL	N – NL	S – SL	S – N	EL – WL	E – EL	W – WL	E – W
							

TABLE II
FIXED-TIME SIGNAL TIMINGS

Phase No.	Green Movements	Permitted Movements	Duration [sec]
1	NL, SL	Right-turns	7
2	N, S	NL, SL, Right-turns	39
3	EL, WL	Right-turns	23
4	E, W	EL, WL, Right-turns	33

controller. Consequently, there is no need for a deep convolutional neural network, and we simply use a 2-layer multi-layer perceptron as a function approximator. Like DQTSC, in SQTSC, in addition to the queue lengths, the current green phase and its elapsed time is used as inputs to the controller. The action space and the reward function is similar to DQTSC.

C. Tabular Q Traffic Signal Controller (TQTSC)

TQTSC and SQTSC agents share the main components of their RL algorithms: action space, reward function, and state space. The only difference is that the SQTSC agent uses an NN as a function approximator for the action-value function, while the TQTSC agent uses a Q-table as its action-value function. The parameters specific to the TQTSC method, including the discretization intervals, learning rate and exploration rate functions, are adopted from the comprehensive work on TQTSC on the same test case [32].

D. Actuated and Fixed Time

The intersection studied in section III (below) is controlled by an actuated traffic signal controller in real life. In a fully-actuated NEMA signal control system, which extends or terminates phase based on the feedback that it receives from loop detectors near the stop line. Actuated controllers manipulates the phase timings between a minimum and a maximum allowable green time. While the minimum green times are set by safety concerns, the maximum green time is set similar to an optimal fixed time controller.

Signal timings for the fixed time controller are presented in Table II. We use the Webster method [1] to design a fixed time traffic signal controller. In the Webster method, the cycle time is calculated based on the critical movement concept, after which the splits are defined based on the relative flows of each movement.

III. TESTBED

The subject study area of this research includes six intersections in the City of Burlington, Ontario, Canada. However, the primary focus is on the intersection at Harvester Road and Walkers Line. This intersection connects the arterial streets and service roads in Burlington to the main freeway that runs through the city, the Queen Elizabeth Way (QEW). This part of the city is designated for mostly commercial and office use. In the morning, the majority of the traffic comes from the freeway (North-West side of the intersection) and spreads into the arterials, while in the afternoon the majority of the traffic comes from the arterials and joins the freeway. The orientation of the main street is Northwest-Southeast, but for the sake of simplicity we refer to it as North-South. For the same reason, we refer to Northeast-Southwest streets as East-West. The North-South streets have three lanes for through and right-turning movements plus one dedicated left-turn lane, while the East-West streets have two lanes for through and right-turning movements plus one dedicated left-turn lane (Fig. 2 (b)).

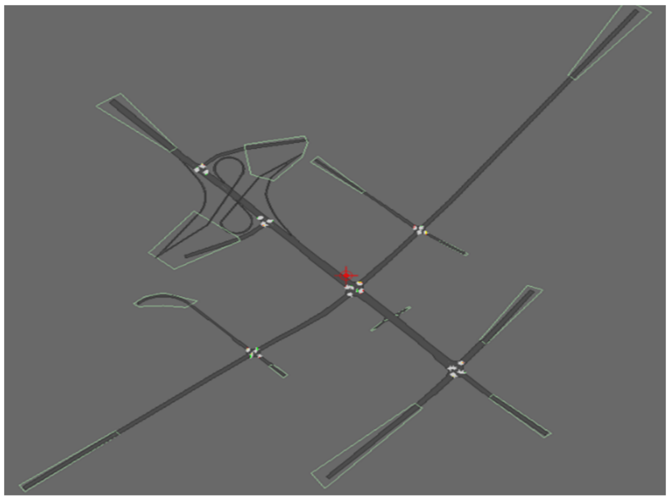


Fig. 3. Testbed traffic network in paramics, with the main intersection highlighted by a red circle.

For this research, we focus on the afternoon peak hour, since the traffic is heavy in all directions and emerges from within the network as opposed to coming from outside. We include the neighboring intersections to create a more realistic arrival pattern at the intersection [32](Fig. 3). This part of the city has mostly commercial use, in the afternoon majority of traffic comes from the arterial and joins the freeway. In reality, the queues frequently reach to the upstream intersections and block them.

To train and test our traffic signal controller, we use Paramics, a microscopic traffic simulator that models the dynamic and stochastic behavior of individual vehicles to create scenarios that are close to real-world cases. We extract sensory information from the simulator and pass it to the traffic signal controller, which is coded in Python. After calculating the controller's response, we apply the selected action to the simulator. The parameters of the micro-simulator and the traffic demand patterns are calibrated to reflect the existing conditions in the field [32].

Each simulation run has a 30-minute warm-up period that is excluded from the calculations and results. The main intersection has four legs, three through lanes and one dedicated left turn on both the North and Southbound movements, and two through lanes plus one dedicated left turn for each East and Westbound street.

IV. Comprehensive Experimental Analysis

The traffic network in this research is simulated in Paramics micro-simulation for the afternoon peak hour (4 PM to 5 PM), with a 30-minute warm-up period. The information from the micro-simulator is extracted using an Application Programming Interface (API) in Paramics and sent to a controller that is coded in Python. We test our deep traffic controller against all the other controllers mentioned in section II.

The detection range for DQTSC, SQTSC, and TQTSC is set to 300 meters from the stop line for each movement. Note that to enhance readability, we refer to DQTSC, SQTSC, and TQTSC as Deep agent, Shallow agent, and Tabular agent, respectively. The size of the experience replay memory used in Deep and Shallow agents is set to 100,000 interactions; if the memory is full, the information of the new interaction replaces the oldest stored interaction. We use randomized batch training with batch sizes of 32, an Adam (adaptive moment estimation) optimizer, and a stochastic gradient descent optimizer with a learning rate of 0.0001 to train the weight of the Deep and Shallow agents. The action selection policy is ε-greedy, with ε decreasing exponentially every 20 simulation runs for the Deep and Shallow agents. For the Tabular agent (i.e., TQTSC), we use an exponentially decreasing learning rate and ε as a function of number of visits for each discretized sate-action cell [32], [33]. The discount factor is set to 0.95.

In our study, the Deep agent consists of a standard form of CNN with three convolutional layers and two layers of FNN. Filter shapes and striding patterns for the three layers of the CNN are as follows: filter 1 is 2×4 with striding of 1×2; filter 2 is 2×4 with striding of 1×2; and filter 3 is 2×2 with striding of 1×3. The FNN is the same network in the Shallow agent — an FNN with 128 and 68 neurons in the hidden layers. The output of the CNN is re-arranged in a 1-dimensional format to match the following FNN, concatenated with the current phase and its elapsed time, and then fed to the FNN. The rectangular filter configuration not only matches the input shape better with less inconsistency, but also results in fewer trainable weights, which ultimately makes the training process faster and the trained model smaller.

In our first experiment, we test the performance of our proposed Deep agent (DQTSC) with the modified reward function (DQTSC-M) against all the other controllers. Next, we test the performance of the Deep agent with the older reward function from [32], [33]. Then we test the effect of discretization length (d) in the input on the Deep agent. In the fourth experiment, we show why we modified the reward function and how the Deep agent behaves with the old reward function. We also test the effect of the approximation method proposed in section II. The approximation method is crucial for practical implementation of the Deep agent. Later, we investigate the performance of this agent under various practical limitations, including limited detection range and low penetration rate of connected vehicles. For the next experiment, we investigate the effect of using multiple sources of traffic sensors and fusing the information from connected vehicles and cameras together. Finally, we investigate the generalization and adaptability of our traffic signal controller in case of a change in traffic flows at the intersection.

For adaptive controllers, the minimum and maximum allowable green times are 7 and 90 seconds, respectively. Also, the phase sequence, phase lengths, and cycle length are not predetermined but are rather the outcome of the agent's decision-making process. Therefore, they vary from one cycle to the next in order to adapt to the demand.

The height of the image-like input is 14, as there are 14 lanes in total on the streets ending at the intersection. The width is 60, as the detection is 300 meters and each cell extends to 5 meters ($d = 5$). The depth is 2: one for position and one for speed. The simulations are run on a 3.40 GHz i7-2600 CPU, 8GB of RAM with Windows 8.1. Because the micro-simulator is slower than the controller, we force the controller to update its weight only once at each time step of the micro-simulator.

V. Results

A. Deep vs Shallow vs Tabular Q-Learning Traffic Signal Controllers

In this section, we test the performance of the modified Deep agent (DQTSC-M), modified Shallow agent (SQTSC-M), Tabular agent (TQTSC), actuated, and fixed-time traffic signal controllers on the Burlington network. We do not use the modified reward function for the Tabular agent (TQTSC) because we are directly using the proposed controller in [32] on the same test case. In the second (B.) part of section V, we investigate and show the performance of the unmodified Deep and Shallow agents with the old reward function. The main difference between the Deep and Shallow agents in this experiment is the input to the controller. For the Shallow agent, we pre-process the information from the sensors to extract the queue lengths and then feed the information to the controller. However, for the Deep agent, we directly feed the position and speed of the existing vehicles to the controller and let the controller learn to extract useful features from this vast information using the CNN.

In order to evaluate their performance, we test each controller on 100 different simulation runs with 100 different randomly selected seeds. Each seed represents a different day in real life, to capture the stochasticity in traffic conditions from one day to another. We take the average of Measures-of-Effectiveness (MOEs) over the 100 simulation runs, ensuring the 100 seeds are the same for all the controllers to guarantee a fair comparison. The MOEs in this study are: (1) Average Network Travel Time: the total travel time of all the vehicles simulated in the traffic network (including all the neighboring intersections) divided by the number of vehicles simulated in each simulation run. (2) Average Intersection Travel Time: the total time that vehicles spend in the center intersection as soon as they enter the street leading to the intersection divided by the number of vehicles that entered and left the intersection. (3) Average In-Queue Time: This measurement gives the total time spent in the queue (in speeds below the queue threshold of

7 km/hr) for all the vehicles passing through the streets leading to the intersection, divided by the number of vehicles passing through the streets leading to the intersection. (4) Average Queue Length: This gives the average number of vehicles in the queues on the streets leading to the intersection over time, along with the standard deviation of all the measurements to evaluate the stability/reliability of the controllers. The average overall performance of controllers is summarized in Table III.

The results show that the modified Deep agent (DQTSC-M) has the capability to handle the raw high-dimensional sensory input properly and find the optimal policy. This traffic signal controller not only eliminates the need for queue detection and all its limitations, data pre-processing, and fine-tuning of the input, but also outperforms the modified Shallow agent (SQTSC-M) by 10.18% at the intersection level and 1.99% network-wide. The most significant improvements are achieved in variations of MOEs ranging from 18% to 39%, as the modified Deep agent takes into account all the vehicles, including approaching traffic that has not yet joined the queue, and thus can take actions more optimally. The Tabular agent (TQTSC) stores its state-action values in a tabular format and has no interpolation or extrapolation abilities.

The Q-table for this study has 16 million cells and requires 4 GB of memory to be stored. Furthermore, to converge to the optimal policy, the Tabular agent needs to be trained for 65,000 iterations, whereas the modified Deep and Shallow agents need only around 600 and 1000 iterations (Fig. 4 and Fig. 5), respectively, to converge, and their Q-Networks need less than 5 MB of memory for the Deep agent and 1 MB for the Shallow agent. Note that the replay memory takes up to 2 GB for the former and 50 MB for the latter but is only needed if the training has to continue after convergence or in the field.

In our experiment, the convergence time for both the modified Deep agent and unmodified Shallow agent is less than 24 hours (mostly limited by the micro-simulator), whereas TQTSC needs more than 30 days.

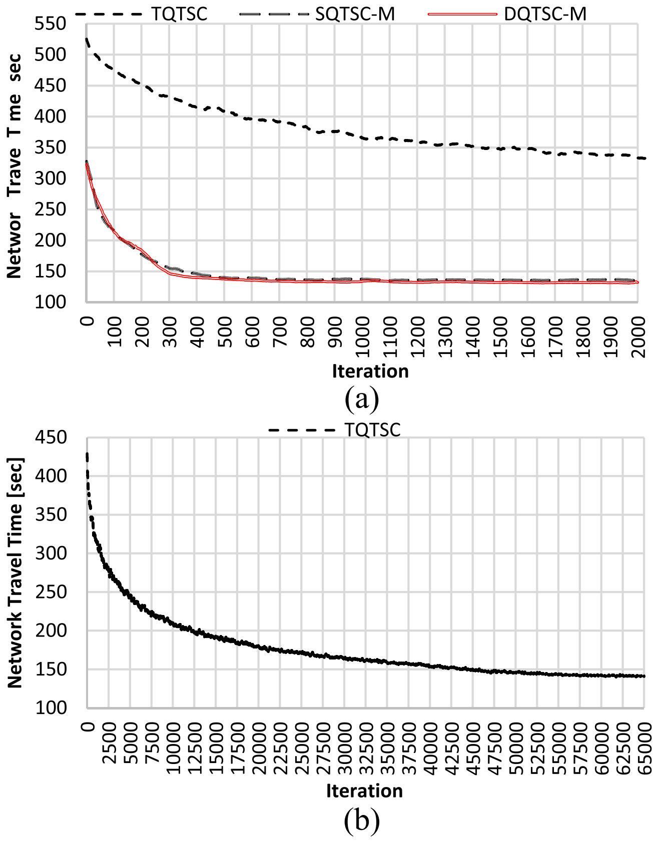


Fig. 4. Network travel times for DQTSC-M, SQTDC-M, and TQTSC (a) and (b).

B. Effect of Reward Function Definition

In this research, as one of the base cases, we use a Tabular agent (TQTSC) from [32]. In their controller, the reward function is defined as the average reduction (savings) in the cumulative delay of the intersection.

In this section, we show the limitations of the reward function from [27] and the improvements achieved from modifying this reward function, as shown in equations 4 and 5. The modified reward function leads not only to better performance of the Shallow agent (SQTSC-M vs SQTSC), but more importantly, more consistent learning behavior. We observe strange learning behavior of the Deep agent (DQTSC) with the unmodified reward function. Although we see that modified reward function leads to slightly better performance in the Shallow agent (SQTSC-M vs SQTSC), we observe a substantial difference between the two reward functions in the Deep agent (DQTSC-M vs DQTSC). To understand the reason, we take a closer look at the learning graphs of the Shallow and Deep agents (SQTSC and DQTSC). In Fig. 5, we observe the moving average of cumulative reward over the last 100 episodes.

The learning graph of the Shallow agent (SQTSC) (Fig. 5 (a)) has an increasing behavior at the beginning like any other RL agent, which is indicative of learning. However, after some point, its performance decreases, showing an issue with the learning process, while the network travel times for both reward functions remain in close range (Table IV). However, for the Deep agent (DQTSC), this is not the case; the cumulative reward graph follows the typical learning graph of an RL agent, increasing and gradually converging to its maximum value, although, the network travel time is almost 90% worse than the modified Deep agent (DRTSC-M).

According to the learning graphs below, and closely watching the controllers in action in the simulation, the Deep agent has apparently learned to *trick* the system to achieve higher cumulative rewards while performing worse in terms of traffic delays. This phenomenon can be explained by looking at the state definition and the reward function. After training the Deep agent, we observed that the controller learns to hold the traffic on the movements with higher flows while switching between other phases with minimum green time. Holding the traffic on one movement leads to higher increase in cumulative delay (more negative reward) that will be taken advantage of in later steps by switching back and forth between the other phases. Each time, the reward gets divided by the hold period (minimum green + yellow + all-red time). When the cumulative delay of

TABLE III
COMPARISON OF FIXED-TIME, ACTUATED, TQTSC, SQTSC-M, AND DQTSC-M TRAFFIC SIGNAL CONTROLLERS, AND PERCENTAGE IMPROVEMENTS OF DQTSC-M COMPARED TO THE OTHER CONTROLLERS

MOE	Fixed-time	Actuated	TQTSC	SQTSC-M	DQTSC-M
Avg. Int. TT. [sec]	87.54	78.97	67.1	57.19	51.37
% improvement of DQTSC	41.32%	34.95%	23.44%	10.18%	-
Avg. In-Q Time [sec]	56.77	48.68	39.07	28.17	23.72
% improvement of DQTSC	58.22%	51.27%	39.29%	15.80%	-
Avg. Q Length [veh]	9.8	8.73	7	5.2	4.48
% improvement of DQTSC	54.29%	48.68%	36.00%	13.85%	-
Avg. Net. TT. [sec]	158.8	151.79	143.9	136.49	133.77
% improvement of DQTSC	15.76%	11.87%	7.04%	1.99%	-
S.D. Int. TT. [sec]	7.19	6.51	10.15	5.43	3.42
% improvement	52.43%	47.47%	66.31%	37.02%	-
S.D. In-Q Time [sec]	6.92	6.17	10.05	5.17	3.16
% improvement	54.34%	48.78%	68.56%	38.88%	-
S.D. Q Length [veh]	15.54	12.91	10.42	7.33	6.01
% improvement of DQTSC	61.33%	53.45%	42.32%	18.01%	-
S.D. Net. TT. [sec]	12.98	5.29	8.53	3.94	2.83
% improvement	78.20%	46.50%	66.82%	28.17%	-

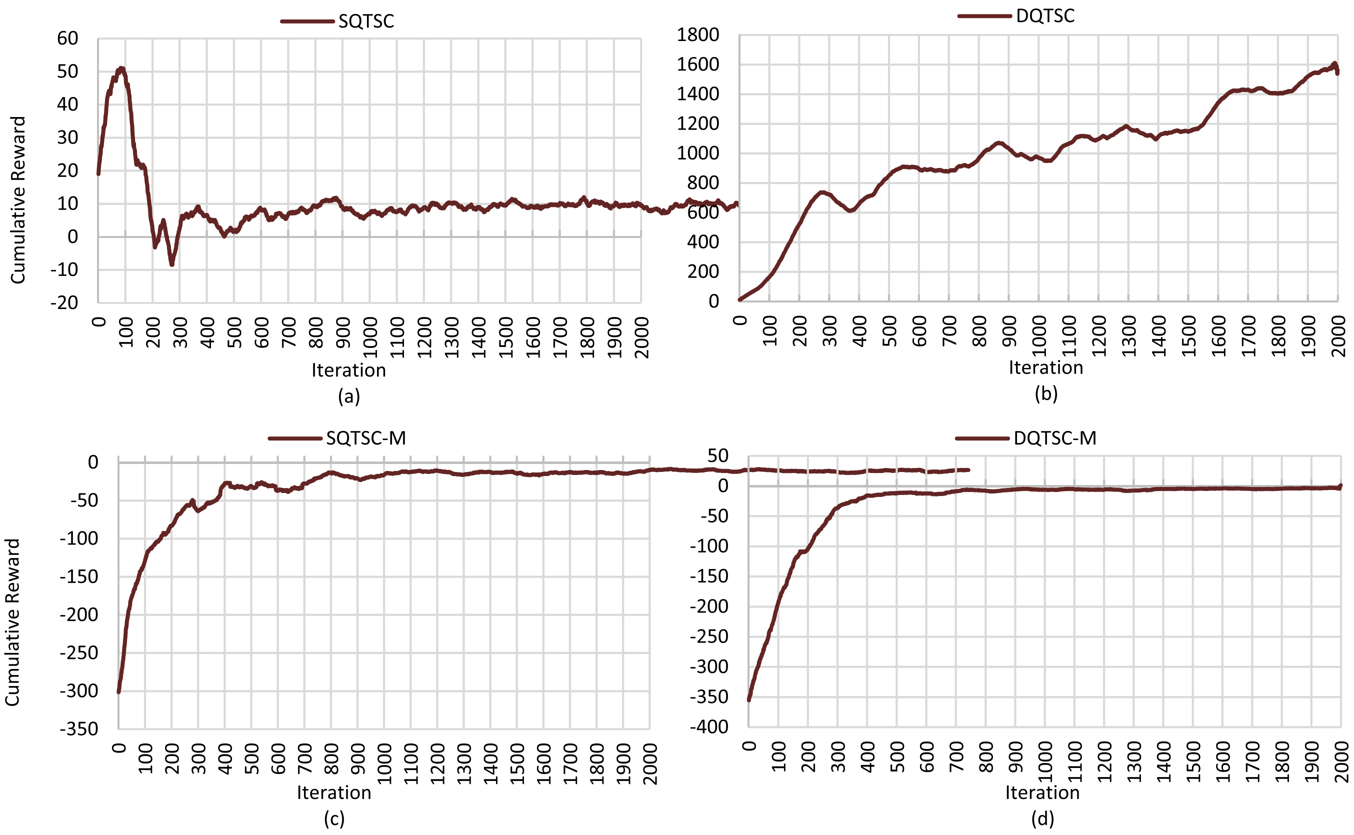


Fig. 5. Cumulative reward values in each episode, moving average over 100 episode for (a) SQTSC, (b) DQTSC, (c) SQTSC-M, and (d) DQTSC-M. (a) Strange learning behavior of SQTSC, (b) Typical learning graph for SQTSC-M. (c) DQTSC achieves much higher reward values by exploiting additional information provided directly from the sensors. (d) Modified reward function leads to expected learning behavior of an RL agent in DQTSC.

that movement is sufficiently significant, the controller turns the signal green and keeps extending it for the movement so that it will receive considerable positive rewards during extension times as a result of queue discharge. In other words, the controller stores traffic to gain from releasing it later, which is a direct result of the reward definition and averaging over the hold period, masking the negative impact of storing.

To explain why the Shallow agent does not face the same issue, we again emphasize that the Deep agent holds the traffic to achieve higher reward values by releasing the delayed vehicles in the extension period. The Shallow agent only receives the queue lengths as the state. Consider a movement with a number of cars waiting in the queue for the signal to turn green; a few seconds after the signal turns green, most cars in the queue start moving forward, although the

TABLE IV
COMPARISON OF DQTSC AND SQTSC WITH BOTH MODIFIED AND UNMODIFIED REWARD FUNCTIONS

MOEs	SQTSC	SQTSC-M	DQTSC	DQTSC-M
Avg. Int. TT.	59.95	57.19	113.41	51.37
Avg. In Q Time	30.82	28.17	83.99	23.72
Avg. Q Length	5.63	5.2	13.76	4.48
Avg. Net. TT.	138.19	136.49	202.84	133.77
S.D. Int. TT.	7.22	5.43	4.13	3.42
S.D. In Q Time	7.04	5.17	4.00	3.16
S.D. Q Length	8.61	7.33	26.29	6.01
S.D. Net. TT.	5.28	3.94	7.51	2.83

TABLE V
EFFECT OF DISCRETIZATION LENGTH ON THE PERFORMANCE OF DQTSC-AM

MOEs	*d = 5m*	*d = 10m*	*d = 20m*	*d = 50m*	*d = 100m*
Avg. Int. TT.	51.37	52.34	52.43	53.32	56.44
Avg. In-Q Time	23.72	25.09	24.66	25.51	28.60
Avg. Q Length	4.48	4.69	4.62	4.75	5.33
Avg. Net. TT.	133.77	134.33	134.36	134.88	140.35
S.D. Int. TT.	3.42	3.61	3.82	4.33	14.46
S.D. In-Q Time	3.16	3.33	3.55	4.06	14.53
S.D. Q Length	6.01	6.43	6.30	6.56	8.90
S.D. Net. TT.	2.83	2.81	2.98	3.22	31.87

majority of them are still behind the stop bar. Since the cars gained speed and are moving, the queue length by definition becomes zero and the controller becomes blind to the situation of the movement. For example, the Shallow agent cannot differentiate between 20 moving cars and an empty street. Depending on the situation, it receives different reward signals and fails to find the optimal policy that maximizes its cumulative reward. In the extension period, the delayed vehicles in the queue are moving but have not left the intersection, yet the Shallow agent does not sense the vehicles. In contrast, the Deep agent receives full unabstracted information about all the vehicles in the intersection, moving or stationary, and uses this information to learn the policy that maximizes the cumulative reward. It also exploits it when it can to gain reward, even at the expense of overall intersection delay.

These results show that how important and challenging designing a good reward function can be. With a change in the state space we see that the same reward function yields significantly different results (SQTSC vs DQTSC).

We conclude that the modified reward function not only leads to the better and more reliable performance of the Shallow agent (SQTSC), but also leads to meaningful learning behavior and hence is the viable reward function for the Deep agent (DQTSC).

C. Effect of Discretization Length

In this section, we investigate how changing the discretization precision (d) of the input may affect the modified Deep agent's performance (DQTSC-M). Initially, we set this value at 5 meters. The smaller the discretization length, the more detailed the input to the controller will be, which is analogous to the resolution of an image. However, if d is too small, the size of the input increases to the point that the learning process might become slow. On the other hand, when d is large, the size of the input decreases, along with the precision of the input data. If we choose a very large d, then each cell would include multiple vehicles and their information will be integrated when being presented to the controller. Choosing a discretization length close to the average length of vehicles is the sensible middle ground, since any value smaller than this for d would not provide more information and increasing the dimensionality of the input could potentially cause computational complexity.

We test the modified Deep agent (DQTSC-M) with different discretization lengths. The results presented in Table V show that the performance of the Deep agent is not significantly influenced by discretization lengths up to 50 meters. However, we see that higher values for d lead to more variations in the performance of the controller, while still in a good range and better than the modified Shallow agent (SQTSC-M). We can conclude that any value between 5 and 20 meters is a good discretization length and up to 50 meters is not detrimental.

From this point onward, and for comparison purposes in the rest of our experiments, we will continue with the 5-meter discretization length.

D. Effect of Approximation of Cumulative Delay

Our main goal is to design a practical controller that can be implemented on real intersections and given the limitations of detection technologies. The reward function that the modified Deep and Shallow agents (DQTSC-M and SQTSC-M) are using is dependent on knowing the delay of the individual vehicles. Such information, however, is impractical to hope for from current detection technologies. Therefore, we instead use the approximation method explained in section II to estimate the cumulative delay of the intersection from measurements. We test this approximation technique and investigate the performance of the controllers with approximate cumulative delay and actual cumulative delay (obtainable for simulation) of the intersection. For this experiment, we have two sets of two identical agents: one with the modified reward function (M) calculated based on the delay of each vehicle, and the other with the approximate modified reward signal (AM) calculated based on the approximation algorithm. The results for the two controllers are presented in Table VI.

The results show that using the approximation algorithm results in a minor performance deterioration of the controllers. The approximation method affects the controllers' reliability more than their average performance. The average performance loss is not considerable, while the invaluable gain is a controller that can be implemented in practice.

In the experiments in the rest of the paper, we will use the AM agents, due to their practicality.

E. Effect of Penetration Rate of Connected Vehicles (CVs)

One of the main reasons why we developed the Deep agent (DQTSC/-M/-AM) is to harness the power of new and pervasive sensory technologies and to be able to process the vastness of the resulting information. In the near future, we expect to receive traffic data through connected vehicle technology. As connected vehicles are introduced to the market, we will be

TABLE VI
EFFECT OF CUMULATIVE DELAY APPROXIMATION ON THE PERFORMANCE OF SQTSC-M AND DQTSC-M

MOEs	DQTSC -M	DQTSC -AM	Performance Diff. (%)	STQTSC -M	SQTSC -AM	Performance Diff. (%)
Avg. Int. TT.	51.37	51.63	-0.50%	57.19	58.97	-3.1%
Avg. In-Q Time	23.72	24.02	-1.25%	28.17	29.96	-6.4%
Avg. Q Length	4.48	4.52	-0.96%	5.2	5.51	-6.0%
Avg. Net. TT.	133.77	133.94	-0.13%	136.49	137.68	-0.9%
S.D. Int. TT.	3.42	4.02	-17.54%	5.43	6.32	-16.39%
S.D. In-Q Time	3.16	3.25	-2.85%	5.17	6.07	-17.41%
S.D. Q Length	6.01	6.02	-0.17%	7.33	7.64	-4.06%
S.D. Net. TT.	2.83	3.73	-31.80%	3.94	4.84	-22.84%

able to receive numerous types of information from individual vehicles. However, the transition from regular vehicles to connected vehicles is a long process that might actually never unfold to its full extent anytime soon.

To consider this factor, we designed an experiment in which we test our controller under different penetration rates (PRs) of connected vehicles, from 100% down to 0%. For example, if the PR is 40%, this indicates that 4 out of 10 vehicles in the system are connected vehicles, and that only the information of these connected vehicles is visible to the controller. In this experiment, if a vehicle is tagged as a non-connected vehicle (regular vehicle), the controller receives no information about this vehicle, neither in the input nor in the reward signal. The reward signal, which is the approximation of the intersection cumulative delay, is also adjusted to consider only the vehicles that are tagged as connected vehicles. The results are presented in Table VII. As can be seen in the table, the lower the penetration rates of connected vehicles in the system, the lower the performance of the Deep agent (DQTSC-AM), which is expected. However, even a PR as low as 40% can outperform the Shallow agent (SQTSC-AM). The decrease in performance accelerates as the penetration rate drops closer to 0%. At 0%, since the controller is technically blind to traffic conditions and has no means of learning, it converges to a random policy. The randomness of the policy is more obvious in the large standard deviation of the MOEs. The average performance of the controller is not highly affected by the changes in the PR of CVs when the PR is close to 100%. However, changes in the CVs PR results in more variations in the performance of the controller.

We further compare our base-case traffic signal controllers, fixed-time, actuated, Tabular agent (TQTSC) and Deep agent (SQTSC-AM) with the Deep agent (DQTSC-AM) under the conditions of 40% and 20% PRs of CVs in Table VIII. We see that for a PR of 20%, the Deep agent (DQTSC-AM) outperforms the fixed-time and actuated controllers and performs slightly better than the Tabular agent (TQTSC) at the intersection level. The improvement for a 40% PR reaches up to 34%, 27%, and 14% compared to fixed-time, actuated, and Tabular agent (TQTSC), respectively, for intersection travel time, while for a PR of 40%, the Deep agent (DQTSC-AM) performs slightly better than the Shallow agent (SQTSC).

F. Effect of Fusing Multiple Sensor Types

As explained in the previous section, achieving a very high PR of connectivity in the traffic network, if even possible, is not expected to happen any time soon. In the near future, we might have a single-digit PR of connected vehicles in the traffic network and would likely take several years to achieve the 40% PR that leads to the better performance of the Deep agent (DQTSC-AM) compared to the Shallow agent (SQTSC-AM). On the other hand, cameras are being relatively successfully used to extract traffic information, such as queue length at intersections. In addition, many municipalities are implementing cameras at intersections for signal control or safety purposes.

As previously presented, we tested both the Deep and the Shallow agents (DQTSC-AM and SQTSC-AM). The former works with raw sensory information received from CVs with detailed data about the position and speed of each vehicle, while the latter works with queue lengths of each movement, as a pre-processed traffic data input. Although the Deep agent (DQTSC-AM) clearly outperforms the Shallow agent (SQTSC-AM), the performance suffers when the penetration rate of CVs is low. Hence, in the interim until CVs are prevalent, we propose combining both systems to overcome the limitations of each and harness the advantages of fusing inputs from multiple sources. In this setup, we input the position and speed information to the CNN and concatenate its output with the queues of each movement (8 movements), current green phase and elapsed time, as the input to a 2-layer FNN. The reward function is also calculated based on the queue information rather than connected vehicle data. The results from this experiment are presented in Table IX.

There are two important points that we can infer from the results of this section. First, adding the queue lengths to the inputs of the Deep agent (DQTSC-AM) limits the worst performance of the controller to the Shallow agent (SQTSC-AM) agent. It also serves as a reasonable safety measure that can prevent our controller from acting completely randomly in instances of very low penetration rates or communication disruptions. Furthermore, as the PR drops, the performance of the Deep agent (DQTSC-AM) decreases as well, though not as drastically as in the previous experiment in low ranges of penetration rate. Also, we see that the Deep agent that receives the queue lengths in addition to the position and speeds of the vehicles slightly outperforms the Deep agent that does not receive the queue lengths.

G. Effect of Demand Increase Over Time

In our final experiment, we want to evaluate the adaptability of the Deep and Shallow agents (DQTSC-AM and

TABLE VII
EFFECT OF CV PENETRATION RATE IN THE TRAFFIC NETWORK ON DQTSC-AM

	Penetration Rate					
MOEs	**100%**	**80%**	**60%**	**40%**	**20%**	**0%**
Avg. Int. TT.	51.63	52.4	54.79	57.24	65.84	363.54
%Diff vs 100% PR	-	-1.5%	-6.1%	-10.9%	-27.5%	-604.1%
Avg. In-Q Time	24.02	24.74	27.12	29.45	38.06	342.88
Avg. Q Length	4.52	4.65	5.06	5.46	6.99	28.8
Avg. Net. TT.	133.94	134.41	135.97	137.51	145.23	656.91
S.D. Int. TT.	4.03	5.1	5.55	6.64	8.56	225.25
%Diff vs 100% PR	-	-26.6%	-37.7%	-64.8%	-112.4%	-5489.3%
S.D. In-Q Time	3.74	4.77	5.25	6.32	8.45	228.82
S.D. Q Length	6.02	6.25	6.78	7.45	9.95	36.92
S.D. Net. TT.	3.26	3.96	4.15	5.12	9.72	318.35

TABLE VIII
PERFORMANCE OF DQTSC-AM UNDER 40% AND 20% PR OF CVS COMPARED TO BASE-CASE CONTROLLERS

	Base-case Controllers				DQTSC-AM	
MOEs	**Fixed-time**	**Actuated**	**TQTSC**	**SQTSC-AM**	**40% PR**	**20% PR**
Avg. Int. TT.	87.54	78.97	67.1	58.97	57.24	65.84
%Diff vs 40% PR	34.6%	27.5%	14.7%	2.9%	-	-
%Diff vs 20% PR	24.8%	16.6%	1.9%	-11.6%	-	-
Avg. In-Q Time	56.77	48.68	39.07	29.96	29.45	38.06
Avg. Q Length	9.8	8.73	7	5.51	5.46	6.99
Avg. Net. TT.	158.8	151.79	143.9	137.68	137.51	145.23

TABLE IX
EFFECT OF CV PENETRATION RATE ON DQTSC-AM USING CV AND CAMERA INFORMATION TOGETHER

	Penetration Rate					
MOEs	**100%**	**80%**	**60%**	**40%**	**20%**	**0%**
Avg. Int. TT.	50.96	51.88	52.62	53.36	55.1	58.82
%Diff vs 100% PR	-	-1.8%	-3.3%	-4.7%	-8.1%	-15.4%
Avg. In-Q Time	23.44	24.26	24.99	25.69	27.39	29.91
Avg. Q Length	4.42	4.57	4.69	4.82	5.14	5.43
Avg. Net. TT.	133.53	134.09	134.55	134.98	136.16	137.51
S.D. Int. TT.	3.44	3.96	4.8	4.98	5.16	5.68
%Diff vs 100% PR	-	-15.1%	-39.5%	-44.8%	-50.0%	-65.1%
S.D. In-Q Time	3.15	3.67	4.49	4.63	4.79	5.27
S.D. Q Length	5.86	6.04	6.23	6.38	6.76	7.66
S.D. Net. TT.	2.69	2.95	3.64	3.71	3.78	4.11

SQTSC-AM) under conditions characterized by traffic demand changes. Contrary to tabular RL agents, RL agents with function approximation have better generalization ability and are able to react better to situations for which they have not been trained. Also, due to their faster training process, they can adapt to the changes quickly when implemented in the field.

There are many possible scenarios that traffic demand can change. The most common form of change in traffic demand is the increase in demand over time, which causes conventional traffic signal control systems and their timing plan to "age" and become outdated. In our experiments, we assume a linear growth of 10% over 10 days (a 1% increase per day) in traffic demand uniformly across the network. After the 10th day, we keep the traffic demand fixed to see how fast the traffic signal controllers with uninterrupted learning can find the new optimal policy, as shown in Fig. 6. We test whether the controller's previously-learned policy becomes outdated and how well the Deep and the Shallow agents (DQTSC-AM and SQTSC-AM) will, firstly, be able to react to any change in demand due to their generalization ability, and secondly, be able to learn the new optimal policy corresponding to the new traffic demand.

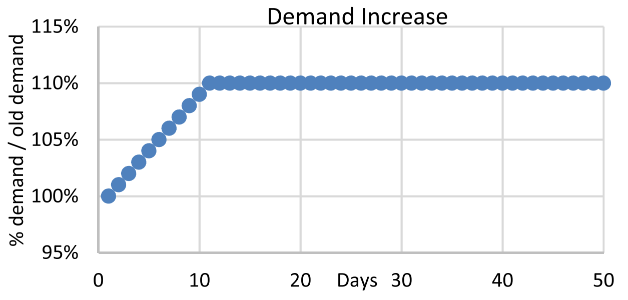


Fig. 6. Trend of increasing demand over 10 days and holding at 110% for 40 days.

First, we train a Shallow agent and a Deep agent on the original traffic demand and expose the trained agents to new demands while their learning rates are set to zero (SQTSC-AM-F and DQTSC-AM-F, with -F representing the fixedness of the agents' parameters). In the second experiment, we train a Shallow and a Deep agent on the original traffic demand, and then expose the controllers to the new demands. All the while, the learning process continues (SQRSC-AM-C and DQTSC-AM-C, with -C representing the continuous updates

TABLE X

COMPARISON OF FIXED-TIME, ACTUATED, SQTSC-AM-F, SQTSC-AM-C, DQTSC-AM-F, AND DQTSC-AM-C DURING DEMAND INCREASE SCENARIOS: -F REPRESENTS THE FIXEDNESS OF THE CONTROLLER'S PARAMETERS, I.E., NO LEARNING, WHILE-C REPRESENTS CONTINUOUS LEARNING IN THE CONTROLLER

		Fixed-time	Actuated	SQTSC-AM		DQTSC-AM	
	MOEs			-F	-C	-F	-C
First 10 days	**Avg. Int. TT.**	104.24	93.34	71.44	70.05	62.02	63.1
	Avg. In-Q Time	72.67	62.29	41.84	40.49	33.45	34.54
	Avg. Q Length	12.31	10.97	7.7	7.47	6.34	6.54
Days 11 to 40	**Avg. Int. TT.**	115.32	109.39	87.62	83.39	80.07	77.95
	Avg. In-Q Time	83.26	77.68	57.78	53.33	50.5	48.4
	Avg. Q Length	13.92	13.5	10.63	9.85	9.45	9.13
Last 10 days	**Avg. Int. TT.**	115.32	109.39	87.62	82.28	80.07	76.81
	Avg. In-Q Time	83.26	77.68	57.78	52.23	50.5	47.28
	Avg. Q Length	13.92	13.5	10.63	9.63	9.45	8.94

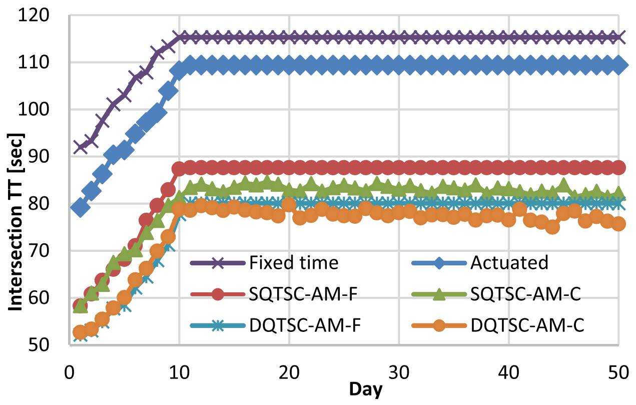


Fig. 7. Intersection network travel times for the six traffic signal controllers during the demand increase scenario.

of the agents' parameters), so the controllers learn to find the new optimal policy. The difference between the two experiments is that, in the latter case, the agent occasionally takes random actions to explore new policies. Despite the fact that occasional random actions slightly reduce the performance of the controller when the traffic demand is fixed, they are necessary for continued learning and adaptability.

SQTSC-AM-F and DQTSC-AM-F have zero-exploration policies and zero learning rates. Rather, they are trained on the original traffic demand and are being applied to the increased demand. While SQTSC-AM-C and DQTSC-AM-C never stop their learning process and are constantly updating their knowledge, z is set to 0.001 and α is set to 0.00001. We reset the replay memory in day 1 and pause the learning until day 2, so that the replay memory will be filled with at least one day of interactions.

The results from these experiments are summarized in Table X and Fig. 7. As shown in the table, we divide the 50 days into three categories: 1) the first 10 days, which is the period when the demand is changing; 2) the middle 30 days, which is the adjustment period for the controller; and the last 10 days, which is the period when the controller is expected to have converged. In the table, we observe that the Deep agents (DQTSC-AM-F/-C) outperform all the other traffic signal controllers, including both of the Shallow agents (SQTSC-AM-F and SQTSC-AM-C) by a margin of at least 6.6% at the intersection level. The improvements rise up to 33.4% for the DQTSC-AM-C traffic signal controller compared to the fixed-time controller when the DQTSC-AM-C converges to its new optimal point. This value for the DQTSC-AM-F is 30.6%. We also see the slightly worse performance of DQTSC-AM-C compared to DQTSC-AM-F at the beginning of the experiments due to the non-zero exploration rate, in addition to the change in DQTSC-AM-C knowledge to adjust itself with the new information.

In Fig. 7, we see the performance of the Deep agents (DQTSC-AM-F/-C) when they are exposed to the gradual increase in demand. The performance of DQTSC-AM-C slightly fluctuates, which is due to its uninterrupted learning ability forcing the controller to try new strategies in order to cope with new situations. In addition, we observe the smaller gap between the two Deep agents (DQRSC-AM-F and DQTSC-AM-C) compared to the gap between the two Shallow agents (SQTSC-AM-F and SQTSC-AM-C). This smaller gap arises from the better generalization abilities of the Deep agent (DQTSC-AM-F) compared to the Shallow agent (SQTSC-AM-F)

VI. CONCLUSION

In this paper, we proposed an adaptive traffic signal controller that has the ability to process and use high-dimensional sensory inputs and learn the optimal policy by directly interacting with the environment. We based our controller on a well-known deep learning method called a Deep Q-Network, but modified our controller to specifically suit our traffic signal control problem. We used a technique to present the traffic sensory information in an image-like form, which is a suitable input format for deep learning methods.

Furthermore, we employed a new state space definition that does not require pre-processing or expert knowledge to extract features. We trained our controller in a micro-simulation of a real-world intersection under different challenging scenarios and compared the results against the state-of-practice and state-of-the-art traffic signal controllers. We showed the advantage of directly feeding high-dimensional sensory inputs to the traffic signal controller and how it affects the controller's strategy. As well, we proposed a modified reward function to correct a learning challenge for the controller. The results show the superiority of the deep controller with the new reward function over the benchmark controllers.

Tabular RL algorithms rely on many parameters that can significantly influence the performance of the controller, including state discretization intervals, action selection probability function and learning rate. Conversely, RL algorithms with function approximation are more robust to changes in these parameters, and are more convenient to work with. We conducted several comprehensive sensitivity analyses and demonstrated the impacts of different practical limitations. We analyzed the performance of the Deep agent under various pixilation ranges dictated by the precision of the traffic sensors or the computational power of the controller. We also investigated scenarios for different penetration rates of the connected vehicles as the futuristic choice of traffic sensors. The results showed the weakness of the Deep agent in very low penetration rates (below 20%). Consequently, we proposed a solution by fusing the inputs from multiple traffic sensors, each addressing the shortcoming of the other. The results indicated the improvements in the performance of the Deep agent for all penetration rates, in addition to its robustness to the variations in penetration rate of connected vehicles. Finally, we tested both the Shallow and the Deep agents' behaviors in case of changes in traffic demand. We evaluated their generalization capabilities and reaction time to find the new optimal policy, finding that the Deep agent outperformed the Shallow agent in both.

In our experiments, we tested the generalization and robustness of our controller to some extent, demonstrating the practicality of the controller. We hope that our controller, when trained in a simulation environment, would be able to be applied in the field and work close to the optimal point, and that after some time it would adapt itself to the changes between the simulator and the real world and refine the optimal strategy. With that said, there are many possible sources of uncertainties, some of which we have examined, such as the impact of discretization length, demand changes, and market penetration or the presence of connected vehicles. Further case-specific sources of uncertainty, such as the atypical geometry of an intersection, should be addressed in the training stage of the agent.

Future work of this research extends to including other modes of transportation, like transit vehicles and pedestrians. In city cores, transit systems play a significant role in transporting people; hence, they need to be treated with priority over regular traffic, as they serve more users in the system. Also, pedestrians are widely affected by traffic signals, especially in downtown areas and business districts, but they are rarely included in optimizations. Since loop detectors are still the most common form of traffic sensors used in the field, we will also focus on developing an adaptive intelligent traffic signal controller that optimizes the traffic signal based on the information provided only from loop detectors. Another important thread to continue is investigating the generality and transferability of our controller. Training a specific controller for each intersection is computationally and manually expensive and sample inefficient. Finding a way to transfer knowledge between controllers from one intersection to another would be a significant advantage. Finally, while it has been established that coordinating traffic signals enables smoother traffic operations along corridors and in networks, creating a coordinated network of self-learning traffic signals is a challenging task due the creep in dimensionality of state action space. These are subjects of our continued and future research.

References

[1] F. V. Webster, "Traffic signal settings," Road Res. Lab., Crowthorne, U.K., Tech. Paper 39, 1958.

[2] L. John, M. D. Kelson, and N. H. Gartner, "A versatile program for setting signals on arteries and triangular networks," *Transp. Res. Rec. J. Transp. Res. Board*, vol. 795, pp. 40–46, Jan. 1981.

[3] N. H. Gartner, S. F. Assmann, F. Lasaga, and D. L. Hous, "Multiband—A variable-bandwidth arterial progression scheme," *Transp. Res. Rec.*, no. 1287, pp. 212–222, 1990.

[4] D. I. Robertson, "TRANSYT: A traffic network study tool," RRL, Road Res. Lab. Crowthorne, Crowthorne, U.K., Tech. Rep. LR 253, 1969.

[5] Trafficware. (2015). *Synchro*. [Online]. Available: http://synchrogreen.com

[6] R. P. Roess, E. S. Prassas, and W. R. McShane, *Traffic Engineering*. Upper Saddle River, NJ, USA: Prentice-Hall, 2004.

[7] P. B. Hunt, D. I. Robertson, R. D. Bretherton, and R. I. Winton, "SCOOT—A traffic responsive method of coordinating signals," Transp. Road Res. Lab., Wokingham, U.K., Tech. Rep. TRRL-LR-1014, 1981.

[8] A. G. Sims and K. W. Dobinson, "Scat-the Sydney co-ordinated adaptive traffic system philosophy and benefits," in *Proc. Int. Symp. Traffic Control Syst.*, vol. 2B, 1979, pp. 19–42.

[9] J.-J. Henry, J.-L. Farges, and J. Tuffal, "The prodyn real time traffic algorithm," in *Proc. 4th IFAC/IFIP/IFORS Conf., Baden-Baden, Federal Republic Germany*, 1984, pp. 305–310.

[10] N. H. Gartner, "OPAC: A demand-responsive strategy for traffic signal control," Transp. Res. Rec. J. Transp. Res. Board, Washington, DC, USA, Tech. Rep. 906, 1983.

[11] V. Mauro and C. Di Taranto, "UTOPIA," in *Proc. IFAC/IFIP/IFORS Symp. Control, Comput., Commun. Transp.*, 1989, pp. 245–252.

[12] K. L. Head *et al.*, "Hierarchical framework for real-time traffic control," Transp. Res. Rec. J. Transp. Res. Board, Washington, DC, USA, Tech. Rep. 1360, 1992.

[13] B. Abdulhai and L. Kattan, "Reinforcement learning: Introduction to theory and potential for transport applications," *Can. J. Civil Eng.*, vol. 30, no. 6, pp. 981–991, Dec. 2003.

[14] B. Abdulhai and G. J. Karakoulas, "Reinforcement learning for true adaptive traffic signal control," *J. Transp. Eng.*, vol. 129, no. 3, pp. 278–285, 2003.

[15] A. L. C. Bazzan, "Opportunities for multiagent systems and multiagent reinforcement learning in traffic control," *Auton. Agents Multi-Agent Syst.*, vol. 18, no. 3, pp. 342–375, Jun. 2008.

[16] B. Chen and H. H. Cheng, "A review of the applications of agent technology in traffic and transportation systems," *IEEE Trans. Intell. Transp. Syst.*, vol. 11, no. 2, pp. 485–497, Jun. 2010.

[17] S. El-Tantawy, B. Abdulhai, and H. Abdelgawad, "Multiagent reinforcement learning for integrated network of adaptive traffic signal controllers (MARLIN-ATSC): Methodology and large-scale application on downtown Toronto," *IEEE Trans. Intell. Transp. Syst.*, vol. 14, no. 3, pp. 1140–1150, Sep. 2013.

[18] W. Genders and S. Razavi, "Using a deep reinforcement learning agent for traffic signal control," 2016, *arXiv:1611.01142*.

[19] J. Gao, Y. Shen, J. Liu, M. Ito, and N. Shiratori, "Adaptive traffic signal control: Deep reinforcement learning algorithm with experience replay and target network," 2017, *arXiv:1705.02755*.

[20] Y. Gong, M. Abdel-Aty, Q. Cai, and M. S. Rahman, "Decentralized network level adaptive signal control by multi-agent deep reinforcement learning," *Transp. Res. Interdiscipl. Perspect.*, vol. 1, Jun. 2019, Art. no. 100020.

[21] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[22] X. Xu, L. Zuo, and Z. Huang, "Reinforcement learning algorithms with function approximation: Recent advances and applications," *Inf. Sci.*, vol. 261, pp. 1–31, Mar. 2014.

[23] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," 2013, *arXiv:1312.5602*.

[24] T. Chu, J. Wang, L. Codecà, and Z. Li, "Multi-agent deep reinforcement learning for large-scale traffic signal control," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 3, pp. 1086–1095, Mar. 2019.

[25] T. Tan, F. Bao, Y. Deng, A. Jin, Q. Dai, and J. Wang, "Cooperative deep reinforcement learning for large-scale traffic grid signal control," *IEEE Trans. Cybern.*, vol. 50, no. 6, pp. 2687–2700, Jun. 2019.

[26] H. Wei *et al.*, "Presslight: Learning max pressure control to coordinate traffic signals in arterial network," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 1290–1298.

[27] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[28] *Paramics Microscopic Traffic Simulation Software*. Edinburgh, U.K.: Quadstone Paramics.

[29] *Smartmicro: Intersection Management Radar*. Accessed: Mar. 2018. [Online]. Available: http://www.smartmicro.de/traffic-radar/intersection-management/

[30] C. Hill, B. A. Hamilton, and G. Krueger, "Module 13: Connected vehicles," ITS ePrimer-U.S. DOT, Tech. Rep., 2013, pp. 1–50.

[31] S. M. A. Shabestary and B. Abdulhai, "Deep learning vs. discrete reinforcement learning for adaptive traffic signal control," in *Proc. 21st Int. Conf. Intell. Transp. Syst. (ITSC)*, Nov. 2018, pp. 286–293.

[32] H. Abdelgawad, B. Abdulhai, S. El-Tantawy, A. Hadayeghi, and B. Zvaniga, "Assessment of self-learning adaptive traffic signal control on congested urban areas: Independent versus coordinated perspectives," *Can. J. Civil Eng.*, vol. 42, pp. 353–366, 2015.

[33] S. El-Tantawy and B. Abdulhai, "Multi-agent reinforcement learning for integrated network of adaptive traffic signal controllers (MARLIN-ATSC)," Tech. Rep., 2012.

Soheil Mohamad Alizadeh Shabestary received the B.Sc. degree in electrical engineering and the M.Sc. degree in electrical engineering control and systems from the University of Tehran, Iran, in 2010 and 2013, respectively, and the Ph.D. degree in intelligent transportation systems from the University of Toronto, Toronto, ON, Canada, in 2018. He is a Transportation Engineer with Huawei Technologies Canada.

Baher Abdulhai (Member, IEEE) was born in Cairo, Egypt, in 1966. He received the Ph.D. degree in engineering from the University of California Irvine, Irvine, California, USA, in 1996. He has 31 years of experience in transportation systems engineering and intelligent transportation systems (ITS). He has been a Professor with the University of Toronto, since 1998. He is the Director of the Toronto ITS Centre and the new i-City Centre for Automated and Transformative Transportation Systems (iCity-CATTS). In 2015, he was inducted as a Fellow of the Engineering Institute of Canada (EIC). He has received several awards, including the IEEE Outstanding Service Award, the Teaching Excellence Award, and the Research Awards from the Canada Foundation for Innovation, Ontario Research Fund, and Ontario Innovation Trust. The ITS Centre won the Ontario Showcase Merit Award of Excellence and the National GTEC Bronze Medal Award in 2005. His research teams have won international awards, including the International Transportation Forum Innovation Award in 2010 (Hossam Abdelgawad), IEEE ITS 2013 (Samah El-Tantawy), and INFORMS 2013 (Samah El-Tantawy). In 2014, he won the University of Toronto Inventor of the Year Award. In 2018, he won the prestigious CSCE Sandford Fleming (Career Achievement) Award for his contribution to transportation in Canada. He was at the Board of Directors of the Government of Ontario (GO) Transit Authority from 2004 to 2006. He served as a Canada Research Chair (CRC) in ITS from 2005 to 2010.