Deep Reinforcement Learning for Traffic Light Optimization

Conference Paper · November 2018

DOI: 10.1109/ICDMW.2018.00088

3 authors:

Mustafa Coskun
Ankara University
34 PUBLICATIONS **354** CITATIONS
SEE PROFILE

Abdelkader Baggag
Université Laval
36 PUBLICATIONS **511** CITATIONS
SEE PROFILE

Sanjay Chawla
Qatar Computing Research Institute
216 PUBLICATIONS **7,814** CITATIONS
SEE PROFILE

2018 IEEE International Conference on Data Mining Workshops (ICDMW)

Deep Reinforcement Learning for Traffic Light Optimization

Mustafa Coşkun
Qatar Computing Research Institute
HBKU, Doha, Qatar
mcoskun@hbku.edu.qa

Abdelkader Baggag
Qatar Computing Research Institute
HBKU, Doha, Qatar
abaggag@hbku.edu.qa

Sanjay Chawla
Qatar Computing Research Institute
HBKU, Doha, Qatar
schawla@hbku.edu.qa

***Abstract*—Deep Reinforcement Learning has the potential of practically addressing one of the most pressing problems in road traffic management, namely that of traffic light optimization (TLO). The objective of the TLO problem is to set the timings (phase and duration) of traffic lights in order to minimize the overall travel time of the vehicles that traverse the road network.**

In this paper, we introduce a new reward function that is able to decrease travel time in a micro-simulator environment. More specifically, our reward function simultaneously takes the traffic flow and traffic delay into account in order to provide a solution to the TLO problem. We use both Deep Q-Learning and Policy Gradient approaches to solve the resulting reinforcement learning problem.

***Index Terms*—Deep Q-Learning, Deep Policy Gradient, Traffic Light Optimization**

I. INTRODUCTION

With the fast-growing population around the world and large scale migration to cities, road traffic congestion has become one of the major problems around the world. In the context of urban infrastructure management, there is an urgent need to come up with approaches to alleviate traffic congestion. One of the most promising approaches is to design and develop smart and adaptive traffic management systems which leverage availability of continuous traffic data from sensors in conjunction with modern machine learning and artificial intelligence methods to manage traffic. The upside of efficient traffic management systems is huge in terms of both economic productivity as well as decreasing carbon emission. For example, traffic congestion costs across Europe are estimated at 1% of EU's GDP [6]. Considering EU's current GDP is about twenty trillion USD, easing traffic congestion by even 1% can save two billion USD!

Traditionally traffic congestion has been studied using mathematical modeling of traffic flows, e.g. see [5]. For example, the traffic flow on a road network is modeled as an optimal control problem where the state is governed by a coupled system of partial differential equations. In contrast, more recent Reinforcement Learning (RL) techniques use a data-driven approach where the state can be more abstract, e.g., video frames at an intersection, and are governed by a Markov Decision Process (MDP), see [2]–[4], [20].

Initial results of using Reinforcement Learning techniques for traffic light optimization problem have led to some promising results [11], [19], [21], [25]. An RL technique does not require perfect knowledge of the environment. Rather, it interacts with the environment in a sampling fashion to gain knowledge about the environment. An RL method can be thought of as supervised learning with changing datasets [21]. The learning process is carried out by an RL agent based on a trial and error fashion. The agent receives a scalar-reward upon taking an action in the environment. This scalar reward plays the role of the importance of action taken and it can be thought of as an evaluation of the action. In the end, the goal of an RL agent is to maximize the reward collected from the environment over the time.

Typical reinforcement learning techniques, for instance, Q-Learning [21], requires tabular representation of a state and action space. However, in the traffic light management problem, the complexity of an RL technique grows exponentially with the size of the state and action space. Recently, the complexity problem of traditional RL techniques has been overcome by translating the problem from tabular version (local approach) to a functional version (global approximation approach). This change led to some successful development of deep reinforcement learning applications including Atari 2600 games [17] and the Computer Go program [24]. From a practical perspective, the most important part of an RL system is to design an appropriate reward function for the application domain under study and this is the focus of this paper.

In this paper, we propose a new reward function that considers traffic flow and delays simultaneously. This reward function is used in the adaptive traffic control problem to be maximized by a deep learning approach. In our experiment we use Deep Q-Learning (DQN) and Deep Policy Gradient (DPG) as benchmark deep reinforcement learning approaches in conjunction with SUMO(Simulation of Urban MObility), a well known traffic microsimulator [13].

The rest of the paper is organized as follows: in the next section, we provide an overview of the literature on traffic light control problem. In Section III, we describe the terminology of reinforcement learning as well as deep reinforcement learning. In Section IV we establish the background on traffic light control problem based on deep reinforcement learning and describe our approach. In Section V, we provide a detailed experimental evaluation of our method. We draw conclusions and summarize directions for future research in Section VI.

II. Related Work

Adaptive traffic light control systems have received significant attention over the past years. In particular, by using reinforcement learning, significant research has been done in the traffic light control [2], [4], [20], [27]. All of them formulated TLO problem as reinforcement learning and Ritcher *et al.* took a credit assignment approach to TLO, see [20]. Despite their own successes, these methods suffer from the applicability of real-world scenarios since they use reinforcement learning approaches instead of deep reinforcement leaning methods [19].

More recently, with the advancement of reinforcement learning, the traffic light control problem has been reformulated as deep reinforcement learning problems [14], [19], [25], [26]. Among them, Pol *et al.* [25] provides the most extensive results by leveraging different parameters for a reward function. In particular, their main contribution was defining a reward function that considers five different parameter coefficients [25]. Specifically, they consider *number of teleports, number of action switches, number of emergency stops, a sum of delay, and a sum of wait time* as parameter coefficients in their reward function [25]. Later, Jaimy Van Dijk in [10] shows that the first three coefficients do not drastically affect the performance of Deep Learning methods. Also, the delay has been considered as only parameter coefficient in [14], [19]. Most recently, Wei *et al.* gives experimentally very intense approach to TLO [26]. However, any of the above-proposed approaches have not considered the effect of traffic flow in the traffic light control problem.

In this paper, we mainly focus on the deep traffic control problem in the context of defining a new reward function. Specifically, we define a reward function that considers the traffic flow, in addition, to delay as a parameter. This way, we aim at maximizing the reward function while decreasing the travel time. Also, our reward function can reduce the waiting vehicles queue in an intersection comparing the reward function that does not take the traffic flow into account in the same deep reinforcement algorithms.

III. Background

In this section, we briefly describe terminology for RL and Deep Reinforcement Learning. Then, we give an overview about why we need to use Deep Reinforcement Learning over traditional RL technique for traffic light control problem.

A. Reinforcement Learning

In traditional RL setting [7], an agent interacts with an environment. For a finite state space, the interaction lasts till the agent reaches a terminal state. Here, the problems we intend to solve are treated as Markov Decision Processes. A MDP consists of five-tuples $\langle \boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{A}}, \mathbf{P}, R, \gamma \rangle$ where $\boldsymbol{\mathcal{S}}$ is a set of states in the state space, $\boldsymbol{\mathcal{A}}$ is a set of actions, $\mathbf{P}$ is a transition probability matrix which defines the probability of moving for an agent from one state to another state, R is a reward function, and $\gamma \in (0,1)$ is a discount factor which is used to model or balance the importance of an immediate

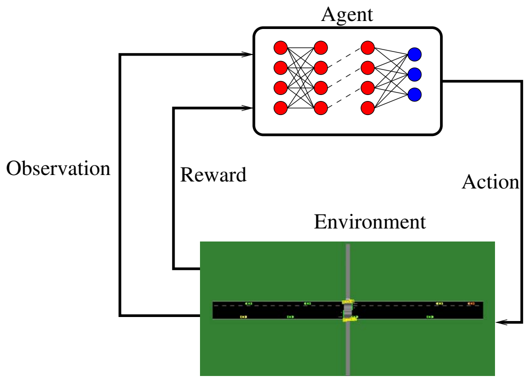


Fig. 1: **Deep RL Agent.** The agent interacts with the environment and gain a reward. Based on the reward it makes an observation and takes the action.

reward versus a future reward. In the context of RL, an agent in a state $s_t \in \boldsymbol{\mathcal{S}}$ at time step t interacts with its environment and takes an action $a_t \in \boldsymbol{\mathcal{A}}$ by following a transition probability given in the matrix $\mathbf{P}$. This process takes the agent from s_t to s_{t+1} with a reward r_t.

The goal of the agent is to learn an optimal policy π that defines the probability of selecting an action a_t in the state s_t so that the agent can follow this policy to maximize the discounted cumulative reward given as follows:

$$R_t = \mathbb{E}\left[\sum_{k=1}^{\infty} \gamma^k r_{t+k}\right]. \tag{1}$$

1) Value Function Based RL: The expected value of cumulative discounted reward under a policy π is given by a *value function*, $V^\pi : \boldsymbol{\mathcal{S}} \to \mathbb{R}$, which is a mapping from a state to a real number starting from a state and following the policy π thereafter [21]. Formally, by following Bellman Equation [21], *the value function* is given as:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi\left[R_t \mid s_t = s\right] \\ &= \sum_{a_t \in \boldsymbol{\mathcal{A}}} \pi(s_t, a_t)[R_{s'} + \gamma \sum_{s' \in \boldsymbol{\mathcal{S}}} \mathbf{P}_{\mathbf{s}'} V^\pi(s')]. \end{aligned} \tag{2}$$

where $\sum_{a_t \in \boldsymbol{\mathcal{A}}} \pi(s_t, a_t)$ is an indication of a stochastic policy, $\mathbf{P}_{\mathbf{s}'}$ is the transition matrix specifying the probability of taking action a_t in s_t and ending up in s' and $R_{s'}$ is a reward function.

In Equation (2), a *value function* based RL approach requires full knowledge of the environment, i.e., $\mathbf{P}_{\mathbf{s}'}$. However, an *action value function* based approach does not require knowledge of the environment since at state s_t an agent knows which action, a_t, it will take. This *action value function* based approach led to the well known *Q-learning* algorithm [21].

2) Tabular Q-learning based RL: *Q-learning* is a model-free RL method which does not requires full knowledge of the environment. Rather, an agent in Q-learning directly estimates the value of taking a specific action, a_t in s_t with the tabular lookup fashion [21]. Formally, in *Q-learning* algorithm, the agent updates *Q-table* of (s_t, a_t) pairs as follows:

$$\begin{aligned} Q^{\pi}_{t+1}(s_t, a_t) &= Q^{\pi}_t(s_t, a_t) \\ &+ \alpha \left[r_t + \gamma [\max_{a'} Q^{\pi}_t(s_t, a')] - Q^{\pi}_t(s_t, a_t) \right] \end{aligned} \tag{3}$$

where α is a learning parameter.

In most of the real-world applications, we have many states and actions or even continuous state space which prevents us to apply classical RL techniques. Therefore, in practice, we use a function approximation technique to deal with the large state and action spaces problem [18]. In this case, the *Q-value* is no longer an entry of $|\boldsymbol{\mathcal{S}}| \times |\boldsymbol{\mathcal{A}}|$ table ,rather, it is a function parametrized by learned parameters θ. These parameters can be updated by using gradient decent methods to minimize the mean squared error in between current estimate $Q^{\pi}_t(s_t, a_t)$ and $r_t + \gamma \max_{a'} Q^{\pi}_t(s_t, a')$.

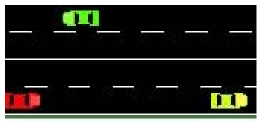
B. Deep Q-learning

Recent successes of deep neural networks have brought the concept of *deep learning*. Deep learning is one of the field of machine learning that consists of a neural network with many layers and focuses on how these layers should be trained efficiently. A neural network is a model that is parameterized by some parameters [8]. These parameters play the role of mapping N-dimensional input vectors to K-dimensional output vector via series of hidden layers ℓ and activation functions [8]. Concretely, interconnected layers of a neural network compute a linear mapping between input vector and its weights, adding bias term and then mapping the result via a non-linear activation function. That is, for a given input $\mathbf{x}$ vector and hidden layers with weights $\mathbf{W}_\ell$ and bias terms, $\mathbf{b}_\ell$, and a non-linearity h_ℓ mapping , we form an output vector $\mathbf{x}'$ as follows:

$$\mathbf{x}' = h_\ell(\mathbf{W}_\ell \mathbf{x} + \mathbf{b}_\ell).$$

The output vector, $\mathbf{x}'$, becomes an input vector to the next layer and this process continues. To approximate more complex function, we need to grow the neural network deeper with the trade-off of training cost.

In the context of RL, *deep reinforcement learning* indicates the use of deep neural networks as a function approximation. The deep reinforcement learning enables to learn large number of functions' approximation for large state and action spaces and result in transforming tabular Q-Learning concept to global function approximations. One of the well-known deep reinforcement learning is Deep Q-learning (DQN) that is a variant of Q-Learning with function approximation [17]. One of the issues with transforming the RL method from tabular to function approximation approach is that of the convergence fluctuation problem. This problem rooted in assumption of i.i.d of neural networks whereas the data generated by an agent is dependent. To alleviate this problem, DQN samples experience from an experience replay database known as $\boldsymbol{\mathcal{D}}$, see [17]. The resulting algorithm's pseudocode is given by Algorithm 1 .



$$\begin{array}{|ccccc|} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{array}$$

Fig. 2: **State Representation of** S<span>UMO</span>**.** The state is discretized by the detectors in SUMO . 1 represents the appearance of a vehicle in the state and 0 indicates the absence of a vehicle in the discretized cell.

In any reinforcement learning approach, one of the problems is to take exploratory or exploitative actions. This issue is rooted in the formulation of any reinforcement learning because of sampling approaches. That is, if the dynamic of the system is not available, we need to sample the trajectories. In this case, greedy action selection could result in missing other high rewards. Thus, to remedy this problem, ε-greedy approach, which selects a random action (explore) with a probability of ε and highest value (exploit) action with probability $1 - \varepsilon$, is taken while choosing an action [21].

C. Policy Gradient Approach

A Deep Policy Gradient (DPG) method tries to optimize a policy function iteratively by estimating the gradient of the policy's performance via learned parameters θ [21]. That is, DPG method directly searches a policy from the policy space instead of estimating the state and action value functions. In RL, Neural Networks become policy networks (policy space) and flattened weights and bias terms are used as θ [21].

In DPG, an agent first samples a trajectory consisting of a sequence of actions and states. Then, the agent's aim is to maximize the total reward in the trajectory [22]. One of the remarkable aspects of DPG is that the system dynamic, known as $\mathbf{P}$, does not depend on the parameters θ. Hence, in the gradient of policy, we do not require to know $\mathbf{P}$ because of derivative [22] .

One drawback of DPG methods is that they typically result in high variance in their gradient estimates [28]. This problem occurs because of the random sampling of the trajectories in the gradient estimate. That is, the derivative of logarithm of a policy and reward difference can be very high. To eliminate this high variance problem, a baseline function is subtracted from the reward. This baseline function can be arbitrary [22].

IV. METHOD

In this section, we first formulate the traffic light optimization problem as a reinforcement learning task by giving details about the state, action and reward. Finally, we present Deep Q-learning and Policy Gradient methods to train the problem.

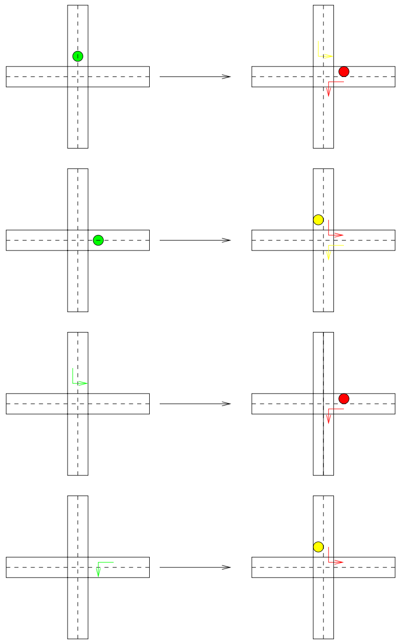


Fig. 3: **Action transition.** In the figures, dots represent the colors of lights and arrows depict the left or right turn signals. The left side of the figure shows the current action configuration. The right side illustrates the all possible intermediate actions can be considered from current action before selecting an action from the action set. This guarantees safety control of the agent.

A. *State Representation*

In this paper, we present the state as a vector which has value one for the presence of vehicles and zero elsewhere in the vector as seen in the Figure 2. This type of vector representation of a state has been vastly used in traffic light optimization literature [11], [12], [25]. There are some approaches that represent state as images [19], however, this type of approaches requires sophisticated sensors such as cameras or radar. Hence, our aim in this study is to use exiting intersection traffic controls such as loop detectors instead of using additional sophisticated sensors.

B. *Action Set*

To control the signal in the traffic light optimization problem at an intersection, we define the possible actions as North-to-South Green (NSG), East-to-West Green (EWG), North-to-South Priority Greed (NWPG), East-to-West Priority Green (EWPG). Concretely, we can define a set of all possible actions as $\boldsymbol{\mathcal{A}} = \{NSG, EWG, NWPG, EWPG\}$.

Here, when the controller at an intersection chooses its action, it also has to consider some safety constraints. To ensure the safe transition from one action configuration to another, the agent needs to consider intermediate traffic signals. That is, an agent has to take current traffic signal and selected action into account while transitioning the traffic signal so that it guarantees the safety. To give a visual illustration, in Figure 3, we show the transition from all possible current signal to the chosen action. The left side of the figure shows all possible current signal while the right side depicts the possible chosen action to ensure safe control of the intersection.

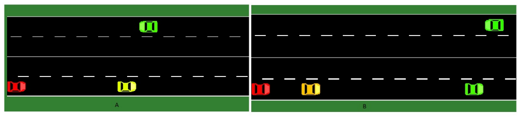


Fig. 4: **Reward Example.** The left figure represents simulation at time 0 and the right figure shows the simulation at time 1.

C. *Reward Function*

An immediate reward $r_t \in \mathbb{R}$ is a scalar value which is gained by the agent after taking an action at each time step t. In this paper, following similar approach to [19], we first set the reward as the total cumulative delay of two consecutive actions defined as:

$$r_t^{(1)} = D_{t-1} - D_t$$

where D_{t-1} and D_t are total cumulative delays in between current and previous time step. That is, the delay of each vehicle is defined as subtracting the vehicle's current speed from maximum allowed speed in a lane and dividing the maximum speed allowed in that lane. The total cumulative delay at time t is the sum of cumulative delays from $t = 0$ to current time for all the vehicles which appeared in the simulation. Hence, a positive reward value shows that the taken actions up to time step t decreases the delay. In the above reward function, which we call *Cumulative Delay Reward* , we want to maximize $r_t^{(1)}$ to decrease the delay.

We can also consider the traffic flow in a lane as positive reward by looking at occupancy which is defined as a given lane fullness in terms of percentage. Moreover, we can treat the number of halting vehicles in a lane as a discouraging reward. Considering the above parameters together we can positively reward the traffic flow as follows:

$$r_t^{(2)} = D_{t-1} - D_t + \frac{\text{Occupancy}}{\text{NumberOfHaltingVehicles} + c}$$

where $c \in (0, 1)$ is constant used to prevent zero division. Intuitively, we are giving a positive reward to a lane fullness if the vehicles on it are not halted to encourage traffic flow.

Take the Figure 4 as an example. Here, we explain how the reward functions are formed during the training phase to give some intuition. Assume we have a highway that allows 90 miles per hour. Further, assume that all the vehicles' speeds are 80 miles per hour in the left figure. Then, the delay for each vehicle is $\frac{90-80}{90} = \frac{1}{9}$. Since we have 3 vehicles in the left figure, the delay is $\frac{1}{3}$ at $t = 0$ time. For the right figure, assume another vehicle entered the highway at $t = 1$ with 80 miles per hour speed. Then, the delay at time $t = 1$ is $\frac{4}{9}$. In this scenario, $r_1^{(1)} = \frac{1}{3} - \frac{4}{9} = -\frac{1}{9}$. For the second reward function, assume each lane can hold 10 vehicles and $c = 1$. The occupancy at time $t = 0$ is $\frac{1}{10} + \frac{2}{10} = \frac{3}{10}$, whereas the occupancy at $t = 1$ is $\frac{4}{10}$ since there is no halted vehicles. From the example, it should be clear that $r_1^{(2)}$ encourages flowing traffic by adding positive terms to the total reward since $\frac{4}{10} > \frac{3}{10}$ whereas $r_1^{(1)}$ discourages the traffic flow with minus reward as $-\frac{1}{9}$.

D. Objective Function and Training

The objective of an agent in Reinforcement Learning is to maximize the cumulative discounted reward. That is, we aim at maximizing the reward under a policy $\pi(a_t|s_t;\theta)$. Under traditional RL, this can be achieved by using tabular Q-learning, however, for large state and action space, tabular Q-learning cannot be employed anymore. Instead, via learned parameters θ, function approximation is used to maximize the cumulative reward, i.e.,

$$J(\theta) = \frac{1}{2}[Q^{\pi}(s,a;\theta^*) - Q_t^{\pi}(s,a;\theta^t)]^2. \quad (4)$$

where $Q^{\pi}(s,a;\theta^*) = r_{t+1} + \gamma[\max_{a'} Q_t^{\pi}(s_{t+1},a';\theta^t)]$ and $Q_t^{\pi}(s,a;\theta^t) = Q_t^{\pi}(s_t,a_t)$. The gradient decent update can be derived by taking the derivative of equation (4). Now, the aim is to compute

$$\frac{\partial J(\theta)}{\partial \theta} = [Q^{\pi}(s,a;\theta^*) - Q_t^{\pi}(s,a;\theta^t)]\frac{\partial Q_t^{\pi}(s,a;\theta^t)}{\partial \theta^t}. \quad (5)$$

By following the above derivation, DQN updates the parameters θ as follows:

$$\theta^{t+1} = \theta^t + \alpha[Q^{\pi}(s,a;\theta^*) - Q_t^{\pi}(s,a;\theta^t)]\frac{\partial Q_t^{\pi}(s,a;\theta^t)}{\partial \theta^t}. \quad (6)$$

Another approach known as Deep Policy Gradient falls into manifold of RL algorithms that work iteratively by estimating the gradient of the policy's performance via learned parameters θ [21]. The basic approach to derive DPG is to make use of a general method of estimating the gradient of expectations [22]. That is, the gradient of the objective function (4) is derived by using expectation's integral formulate as

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \nabla_{\theta} \log(a_t \mid s_t;\theta) R_t\right], \quad (7)$$

where T represents the end of an episode and R_t is the total reward in that episode. Intuitively, Equation (7) tells us that when the total reward in an episode, R_t, is high, then, we must move in the direction in the parameter space that increases $\log(a_t|s_t;\theta)$. The solution to the equation (7) was first introduced in REINFORCE algorithm [28].

With classical REINFORCE [28] approach, the DPG is known to exhibit high variance of the gradient estimate [21]. To alleviate the high variance to some extend, there are some common techniques such as subtracting a baseline function, $b_t(s_t)$ from the total reward [22]. A nearly optimal choice of the baseline function, $b_t(s_t)$, is the state-value function, $V^{\pi_\theta}(s_t)$ in equation (2). Now, we can write adjusted gradient as $\nabla_\theta \log(a_t|s_t;\theta)(R_t - b_t s_t)$. Here, $R_t - b_t(s_t)$ is called the *advantage function*.

Algorithm 1: The Deep Q-learning Algorithm

```
1  Initialize $\theta^T$ and $\theta$ with random weights
2  Initialize replay database $D = empty$
3  Initialize number of simulation as $J$
4  Initialize number of step in a simulation $K$
5  Choose an action $a$ with $\varepsilon$-greedy policy
6  $j = 0, k = 0$
7  while $J > j$ do
8      Initialize $s$ with current intersection matrix
        representation
9      while $K > k$ do
10         Take action $a$ with $\varepsilon$-greedy, get a reward $r$, land
            in $s'$
11         Store $\langle s, a, r, s' \rangle$ in $D$
12         Randomly sample a mini-batch b from $D$
13         for each $\langle s_t, a_t, r_t, s_t' \rangle \in b$ do
14             if $s_t' \sim= terminal$ then
15                 $Q^{\pi}(s,a;\theta^*) = r_t + \gamma[\max_{a'} Q_t^{\pi}(s_t', a'; \theta^T)]$
16             end
17             else
18                 $Q^{\pi}(s,a;\theta^*) = r_t$
19             end
20             Update parameters $\theta$ by (6)
21         end
22         Every $N$ step:
23         $\theta^T = \theta$
24         increment $k$
25     end
26     increment $j$
27 end
```

The combination of the policy (the actor), $\pi(a_t, s_t; \theta)$ and value function (critic), $V^{\pi_{\theta^T}}(s_t)$, results in well-known A2C algorithm [16]. Moreover, similar to buffering approach in DQN, A2C algorithm performs a single update by selecting actions up to D step or terminal. That is, the agent collects D rewards from the environment and updates its policy parameters in a minibatch fashion. Then, the parameters θ are updated via DGD as follows:

$$\theta = \theta + \alpha\left[\sum_{t} \nabla_\theta \log(a_t|s_t;\theta) A(s_t, a_t; \theta, \theta^T)\right], \quad (8)$$

where $A(s_t, a_t; \theta, \theta^T)$ is the current estimate of the *advantage function* corresponding to

$$\sum_{i=0}^{n-1} \gamma^i r_{t+1} + \gamma^n V(s_{t+n};\theta) - V(s_t;\theta^T)$$

where $n \leq D$. The A2C algorithm is defined as Algorithm 2.

V. Experimental Results

In this section, we systematically evaluate our proposed reward function approach that is used DQN and DPG algorithm. We start our discussion by describing the dataset and

Algorithm 2: The Deep Policy Gradient Algorithm for Traffic Light Optimization

```
1  Initialize $\theta^T$ and $\theta$ with random weights
2  Initialize replay database $D = empty$
3  Initialize number of simulation as $J$
4  Initialize number of step in a simulation $K$
5  $j = 0, k = 0$
6  while $J > j$ do
7      Initialize $s$ with current intersection matrix
         representation
8      while $K > k$ do
9          Take action $a$ under policy $\pi(a|s;\theta)$, get a reward
             $r$, land in $s'$
10         Store $\langle s, a, r, s' \rangle$ in $D$
11         if ${s_t}' \sim= terminal$ then
12             $R = V(s;\theta^T)$
13         end
14         else
15             $R = 0$
16         end
17         Take $N$ sized mini-batch b from $D$
18         for each $\langle s_t, a_t, r_t, {s_t}' \rangle \in b$ do
19             $R = r_i + \gamma R$
20             Updata $\theta$ with (8)
21             $\theta^T = \theta^T + \nabla_\theta^T (R - V(s_i;\theta^T))$
22         end
23     end
24 end
```

the experimental setup. We then assess the performance of DQN and DPG for the traffic light optimization problem. Subsequently, we compare the performance of two reward function in terms of average travel time and the length of the queue formed in the intersections. Finally, we broadly give our observations about TLO.

A. Experimental Setup

In our experiments, we use SUMO(Simulation of Urban MObility) [13] to simulate the traffic. SUMO is a well known open source traffic simulator which provides Application Programming Interface (API), specifically, Traffic Control Interface (TRaCI) for programming languages including Matlab and Python. More concretely, we use TRaCI written in Python along with SUMO v.0.32.

Figure 5 depicts the intersections of traffic simulation for the city of Bangalore. In our experiment, we use this simulation dataset for 1000 simulation step. To generate the traffic demands for a single simulation, we set $N = 1000$ and $p = 0.1$ where N is the time duration and p is an uniform probability distribution in the SUMO . With this setting, in a four ways single junction, the expected number of vehicles is $1000 \times 4 \times 0.1 = 400$.

In all experiments, we use the same neural network architecture in [18] with rectifier nonlinearity and update weights of

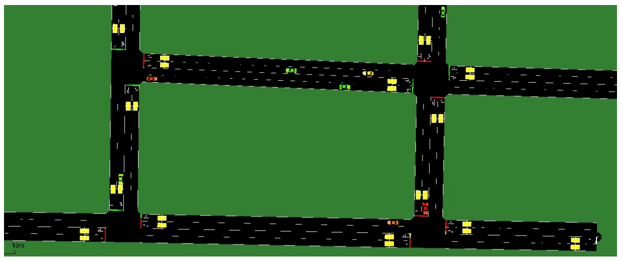


Fig. 5: **Visualization of a Bangalore street.** SUMO provides real-world street simulations.

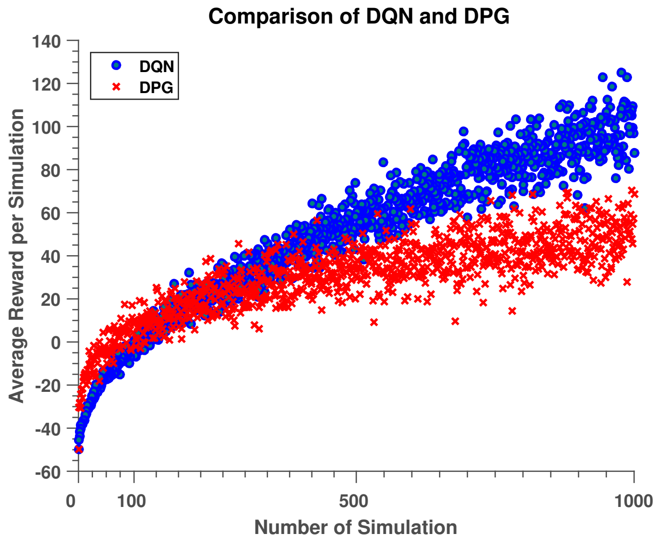


Fig. 6: **Comparison of the performance of DQN and DPG.** The figure shows performance of DQN and DPG in terms of average reward during the simulation.

the network with Adam optimizer [9]. We set discount factor $\gamma = 0.995$ and learning rate $\alpha = 0.001$. We set memory size to $D = 2000$ and mini-batch size to $b = 200$. The network is trained for 1000 episodes, 1 million time steps. That is, each episode corresponds to 1000 SUMO simulation. The learned policies were evaluated every two episodes by running one SUMOGUI.

B. Results

We then assess the performance of DQN and DPG methods in learning traffic light optimization problem with the reward functions we defined in the former section. The purpose of this analysis is to quantify the improvement using Deep Learning approaches and to demonstrate their applicability to the problem at hand. We also conduct the experiments for the baseline method, however, we omit the baseline results since it gives almost constant results which is consistent with former studies [19], [25].

Figure 6 shows the average reward per simulation for DQN and DPG. We limit this analysis to second reward function since the DQN and DPG provide similar results for first reward

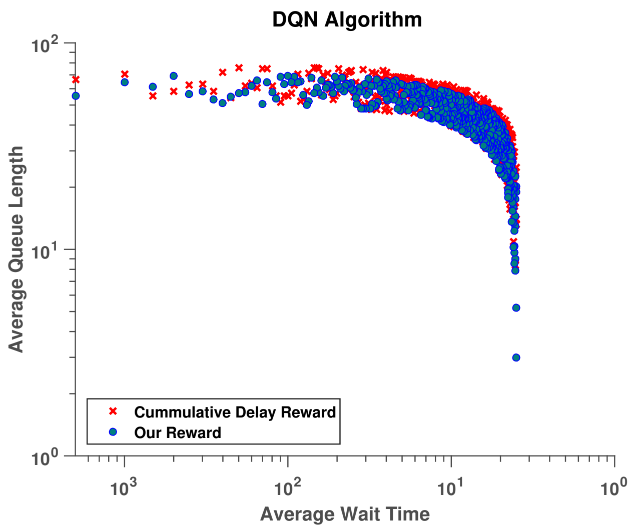


Fig. 7: **Average Queue Length and Wait Time.** The figure shows the average queue length formed at the intersection during the evaluation phase and Average Wait Time. In this figure, we demonstrate that our proposed reward function in DQN outperforms the Cumulative Delay based reward function.

function in terms of their comparison. As it can be seen in the figure, both methods oscillate due to their instability issues [25]. This problem is also pronounced in [1], [3], [19], [25] and could be rooted in *catastrophic forgetting* [15], a phenomenon in neural network that the network could forget formerly learned tasks while trying to learn more recent tasks. Apart from the oscillation problem, DQN outperforms DPG for traffic light optimization problem which is consistent with the findings with the former study in [19].

C. Evaluation

To evaluate the proposed reward function, we compare both reward function inside DQN by running 1000 SUMO simulations in the testing phase. Here, we omit the evaluation of the reward functions in DPG because DQN outperforms DPG for TLO. As for performance metrics we use both average travel time and queue length formed at the intersections. As seen in the Figure 7, our proposed reward function is able to considerably decrease the average travel and queue length comparing to the Cumulative Reward function used in [19].

D. Discussion and Observations

Applying Deep Q-learning or Deep Policy Gradient to traffic light optimization problem might seem appealing because of their recent successes in [16], [18], [24]. However, as expected, the TLO problem is non-trivial and based on our experience we make the following observations.

First, in TLO, if an agent takes suboptimal actions, these often result in increase in traffic congestion. At this point, recovering from the traffic jam is almost impossible. On the other hand, in the earlier works, e.g. see [16], [18], [24], recovering from suboptimal actions for an agent is relatively easy since Atari games allow the agent to make a fresh start. Thus, the explore and exploit strategy for TLO has to be much more delicate.

Second, a reward function can drastically affect the performance of the underlying DQN and DPG algorithm. For instance, most of the earlier works on TLO [12], [19], [25] and this work consider the delay of vehicles as a major part of the reward function. However, we observe that both DQN and DPG tend to converge one traffic light configuration, i.e, letting the vehicles on a major road to pass through while halting the minor road forever. To alleviate this problem, more recent constrained optimization approaches such as TRPO method [23] might be a useful option instead of DQN or DPG.

Third, delay based reward function causes DQN and DPG learn to open a road and closing the other road completely as an optimum solution [25]. This means that an RL algorithm converges to either a fixed or flickering light configuration. In the SUMO environment [13], this fixed or flickering light configuration is handled by introducing emergency stops (since SUMO [13] is collision free). However, these emergency stops are the cause of unpleasant driving. Hence, a reward function should also consider emergency stops in the SUMO environment by considering the action flickering rather than yellow light caused emergency stops as in [25].

Finally, action space in TLO grows exponentially with the state space. Though, in this paper, we consider only four intersections, in the real-world application, action space is one of the major concerns. In the real-world application, one might tackle this problem by clustering states to reduce the action space.

VI. Conclusion

In this paper, we propose an alternate reward function for TLO problem. The proposed approach is based on the idea of considering traffic flow as well as delay information simultaneously. We show that our approach considerably decreases travel times and queue length in practice on real-world simulation environment. Using a four intersection part of the real-world road network of Bangalore, we show that our approach outperforms existing approaches in Deep Q-learning and Deep Policy Gradient.

Future efforts in this direction would include incorporation of other parameters in the reward function into our framework, extensions to other constrained optimization approaches to avoid Deep RL method learn blocking minor roads, and multi-agent scenarios.

References

[1] M. Abdoos, N. Mozayani, and A. L. Bazzan. Holonic multi-agent system for traffic signals control. Engineering Applications of Artificial Intelligence, 26(5):15751587, 2013.
[2] B. Abdulhai, R. Pringle, and G. J. Karakoulas. Reinforcement learning for true adaptive traffic signal control. Journal of Transportation Engineering, 129(3):278285, 2003.

[3] P. Balaji, X. German, and D. Srinivasan. Urban traffic signal control using reinforcement learning agents. IET Intelligent Transport Systems, 4(3):177188, 2010

[4] E. Brockfeld, R. Barlovic, A. Schadschneider, and M. Schreckenberg. Optimizing traffic lights in a cellular automaton model for city traffic. Physical Review E, 64(5):056132, 2001.

[5] Childress, S. (2005). Notes on Traffic Flow.

[6] Commission of the European communities. White paper-European transport policy for 2010: time to decide. Office for Official Publications of the European Communities, 2001.

[7] Christopher JCH Watkins and Peter Dayan. Q-learning. Machine learning, 8(3-4):279292, 1992.

[8] Christopher M. Bishop. Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[9] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[10] J. V. Dijk . Recurrent Neural Networks for Reinforcement Learning: an Investigation of Relevant Design Choices https://esc.fnwi.uva.nl/thesis/centraal/files/f499544468.pdf

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

[12] Genders, W., and S. N. Razavi. Impact of connected vehicle on work zone network safety through dynamic route guidance. Journal of Computing in Civil Engineering, Vol. 30, No. 2, 2015, p. 04015020. [18] Du, L., and H. Dao. Information Dissemination Delay in Vehicle-to-Vehicle Communication Networks in a Traffic Stream. Intelligent Transportation Systems, IEEE Transactions on, Vol. 16, No. 1, 2015, pp. 66-80

[13] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker. Recent development and applications of sumo-simulation of urban mobility. International Journal On Advances in Systems and Measurements, 5(3&4), 2012.

[14] X. Liang X. Du G. Wang Z. Han ”Deep reinforcement learning for traffic light control in vehicular networks” IEEE Trans. Veh. Technol. [online] Available: https://arxiv.org/abs/1803.11115.

[15] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. Psychology of learning and motivation, 24:109165, 1989.

[16] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In International Conference on Machine Learning, 2016.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu,J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. Nature, 518(7540):529533, 2015.

[18] V Mnih, K Kavukcuoglu, D Silver, A A Rusu,J Veness, M G Bellemare, A Graves, M Riedmiller, A K Fidjeland, G Ostrovski, et al Human-level control through deep reinforcement learning. Nature, 518(7540):529533, 2015.

[19] S. S. Mousavi, M. Schukat, P. Corcoran, and E. Howley, Traffic light control using deep policy-gradient and value-function based reinforcement learning, arXiv preprint arXiv:1704.08883, A

[20] S. Ritcher. Traffic light scheduling using policy-gradient reinforcement learning. The International Conference on Automated Planning and Scheduling., ICAPS, 2007.

[21] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. MIT press, 1998

[22] Schulman, John, Heess, Nicolas, Weber, Theophane, and Abbeel, Pieter. Gradient estimation using stochastic computation graphs. In Advances in Neural Information Processing Systems, pp. 3510 3522, 2015a

[23] Schulman, John, Levine, Sergey, Moritz, Philipp, Jordan,Michael I, and Abbeel, Pieter. Trust region policy optimization. In International Conference on Machine Learning (ICML), 2015a

[24] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. Nature, 529(7587):484489, 2016.

[25] E. Van der Pol and F. A. Oliehoek. Coordinated deepreinforcement learners for trac light control. In NIPS'16 Workshop on Learning, Inference and Control of Multi-Agent Systems, 2016.

[26] Hua Wei, Guanjie Zheng, Huaxiu Yao, and Zhenhui Li. 2018. IntelliLight: A Reinforcement Learning Approach for Intelligent Traffic Light Control. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining (KDD '18). ACM, New York, NY, USA, 2496-2505. DOI: https://doi.org/10.1145/3219819.3220096

[27] M. Wiering et al. Multi-agent reinforcement learning for traffic light control. In ICML, pages 11511158, 2000.

[28] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning, 8(3-4):229256, 1992.