Intelligent Traffic Control System using Deep Reinforcement Learning

Anirudh R†
Packet Hardware Group
Centre for Development of Telematics
Bangalore, India
anirudhr@cdot.in

Mithun Krishnan†
4G-RAN Group
Centre for Development of Telematics
Bangalore, India
mithunk@cdot.in

Akshay Kekuda
Department of CSE
Ohio State University
Columbus, Ohio
kekuda.1@osu.edu

2022 International Conference on Innovative Trends in Information Technology (ICITIIT) | 978-1-6654-0670-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICITIIT54346.2022.9744226

***Abstract*—In this paper, we propose a deep reinforcement learning based traffic signal controller. We use the recently developed Distributional Reinforcement Learning with Quantile Regression (QR-DQN) algorithm to design a risk-sensitive approach to traffic signal control. A neural network is used to estimate the value distribution of state-action pairs. A novel control policy that gives variable weightage to the risk of an action depending on the congestion state of the system, effectively minimizes congestion in the network. Our results show that our algorithm outperforms conventional approaches and also classic RL based ones.**

***Index Terms*—Distributional Reinforcement Learning, Risk, Simulation of Urban MObility, SUMO, QR-DQN.**

I. Introduction

The traffic control problem has become a hot topic of research today. The high population density caused by rapid urbanization leads to a situation where the increased traffic demand has to be accommodated within the existing road infrastructure. This has forced researchers and engineers to explore ways of increasing the efficiency of the traffic flow. We present in this paper an Artificial Intelligence-based traffic controller. Fortunately, recent advancements in AI techniques and edge computing have made these implementations practically feasible.

Of the three major branches of machine learning, Reinforcement Learning (RL) is the one most suited for experience-driven autonomous learning [1]. Once the states, actions, and reward structure are defined and set up, the agent trains itself in a self-explorative manner to find the optimal actions at every step. This type of approach lends itself very well to solving complex control problems. Reinforcement learning provides us with an iterative and incremental technique for learning the best course of action in a possibly varying environment setting.

The objective of this paper is to present an RL based traffic signal controller that aims to maximize the efficiency of the traffic flow in the network, while minimizing the risk of traffic jams. We propose, for the first time, a distributional RL perspective to traffic control. In our approach, we quantify the risk associated with an action to consciously decide the signal to actuate.

We simulate our algorithm using SUMO (Simulation of Urban MObility) on a real-world road network and compare it against the traditional Static Signalling (SS), Longest Queue First (LQF), and another RL-based approach of predictive n-step SARSA [1]. We evaluate these algorithms based on traffic parameters such as waiting time, queue length, and traffic dispersion time.

II. Related Works

Reinforcement learning is well suited for learning optimal solutions for high-dimensional control problems. The traffic control problem, being one, has seen several successful attempts in applying RL techniques to it. We will present here a few notable works and their outcomes.

The concept of risk in reinforcement learning has been well explored. The motivation behind using risk is to give an edge to the RL agent in dire situations. The term risk in RL can be formulated in several ways according to the problem at hand; there is no one universally accepted definition. One of the earliest attempts to incorporate risk measure into reinforcement learning has been discussed in [2]. Here, the vanilla Q-learning algorithm is modified to utilize a parameter $\kappa \in [-1, 1]$, which controls how risk-seeking or risk-avoiding the agent is. This method enables seamless transition of Q-learning based algorithms to a risk-sensitive one.

In [3], the authors employ the commonly used reward variance as the risk measure, and attempt to devise an actor-critic algorithm to solve the constrained optimization problem of maximizing the reward while keeping the variance below a threshold. They prove that when the value and policy functions are represented by a multi-layered overparameterized neural network, we can converge to a globally optimum policy at a sublinear rate. Although, in general, finding optimal policies for an MDP which involves both mean and variance of the reward are considered NP-hard [4], research works such as [3] paved way for subsequent advancements in this domain.

While the previous works focussed on using reward variance as the risk measure, works like [5] propose an alternative way to construe the notion of risk. They detail a heuristic, model free reinforcement learning algorithm using the probability of

† R. Anirudh and Mithun Krishnan are with the Centre for Development of Telematics, Government of India, and have contributed equally to the work.

978-1-6654-0670-3/22/$31.00 ©2022 IEEE

entering fatal states as the risk measure. An infinite horizon criterion using a discount factor for the value of state and an undiscounted one for the risk is arrived at. Unlike our paper which uses an online learning approach, [6] utilizes an offline method called O-RAAC for optimizing a risk-averse criterion. They have shown that this fares better than risk-neutral approaches in robotic control tasks. Application of utility functions as a risk measure has been discussed in [7]. A risk-sensitive Q-learning algorithm is conceived that converges to the optimal policy.

Almost all previous work on RL based traffic light control is based on Deep Q-Networks (DQN). Since the state spaces in these traffic problems are huge, the traditional table-based methods of Q-learning are rendered impractical. DQN instead uses Neural Networks (NNs) to directly estimate the Q-values from the feature vectors. The native DQN is prone to overestimation, hence, a Double DQN [8], consisting of two periodically synchronized identical NNs, is used to counteract it. We use a slight variant of it in this paper. To improve the stability and reliability of the traffic control, a dual-agent Double DQN is proposed in [9] for actuating 4-phase intersections. Oftentimes, we may encounter states in the MDP with very closely valued actions. It becomes difficult for the agent to make an informed choice in these situations. In [10], the authors define a quantity called the Advantage function, $A_\pi(s, a)$, as the difference between the action-value function and the state value function, that shows how advantageous selecting an action is relative to the other actions in a given state. The authors also propose a Dueling DQN scheme wherein, two NNs are used: one for the state value function and one for the state-dependent action advantage function. The authors demonstrate better policy evaluation performance with this technique. An application of this technique to traffic control in [11] shows promising results.

What sets our work apart from the above-mentioned ones is our use of a risk-sensitive approach based on distributional RL to combat congestion in traffic networks. In distributional RL, we try to model the entire distribution of the action-value function rather than only its mean value. Modelling the entire distribution gives a complete picture of the range of effects an action can possess. This gives the agent the ability to avoid risky situations such as traffic jams. Our results show that this foresightedness of our agent helps it perform better than the other proven TD predictive techniques such as n-step SARSA.

III. Experimental Setting

Our experimental setup is the same as for our work in [12]. In [12], we develop an n-step SARSA based traffic controller. In this paper, we develop a QR-DQN based traffic controller and compare against the results in [12].

We have taken a real-world 4-junction road network from Texas, USA using OpenStreetMap for our experiment. We use Simulation of Urban Mobility (SUMO) software to simulate traffic scenarios. Using SUMO, traffic patterns are generated and are used to test the Deep Reinforcement Learning algorithm.

Fig. 1, shows the complete network. The 4 intersections of interest are marked in blue. Fig. 2, shows the traffic signals that are operating in our setting. There are a total of 15 signals across the 4 intersections. These signals are numbered $N1$ to $N15$. In our work, intersections are operated cyclically (INT1 $\rightarrow$ INT2 $\rightarrow$ INT3 $\rightarrow$ INT4 $\rightarrow$ INT1) at a gap of 16s. At each intersection, the decision on which signal to operate is made by the RL agent. The goal of the RL agent is to minimize Queue Lengths, Waiting Time, Time Loss, and to disperse traffic through the network in the least time.

To detect the amount of traffic waiting at the signals, we use Lane area detectors (E2 detectors) covering up to 70m from the intersections. These detectors output the percentage of its area on which vehicles are standing, termed occupancy.

The network consists of roads of one to three lanes. All the individual lanes are 10ft wide. The main arterial roads are 1.5km to 2km in length. The traffic consists of 4 types of vehicles. The traffic distribution is modelled to represent urban traffic - Bikes (41%), Cars (37.5%), Trucks (12.5%), Buses (8.3%). These vehicles have different speeds, acceleration, and braking time parameters.

The number of vehicles entering the network follows a Binomial distribution with $n = 5$ & $p = 0.15$, which closely resembles a real-world scenario. The vehicles are also distributed according to the number of lanes in a road, with higher traffic in multi-lane roads.

As mentioned earlier, the intersections are operated cyclically. This means that the intersections are not operated simultaneously. The reason for choosing such a scheme is to reduce the action space. When they are operated at the same time, the number of possible actions at any time is 4*4*4*3 = 192. However, with a cyclic scheme, the maximum number of possible actions reduces to just 4 (3 in the case of INT4). This greatly decreases the size of the neural network, as the size of the last hidden layer is proportional to the number of

Fig. 1. The complete network used for our experiment.

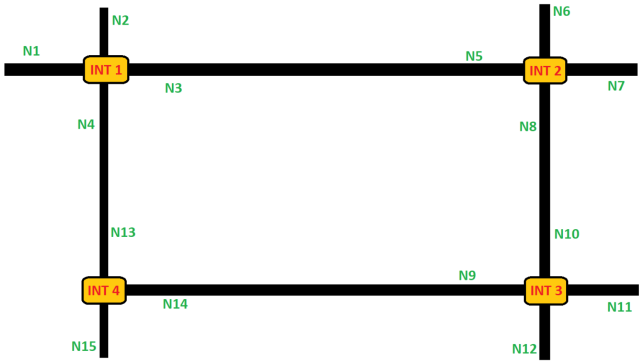

Fig. 2. Traffic signals in our network.

actions. Our observations show that our algorithm can achieve excellent results with just 15 runs of training.

Secondly, this scheme makes the algorithm highly scalable. For the traditional approach, with n intersections, the number of actions increases exponentially as 4^n. But for cyclic operation, the dependency is a linear $4n$ This avoids the various issues prevalent when training a large neural network of a city-level traffic network.

The reward structure is designed to encourage quick dispersal of the traffic waiting at the intersections and to avoid the build-up of traffic jams. The rewards are decided based on the occupancy percentage provided by the lane area detectors present on each side of the intersection. The traffic at each signal is graded as low, medium, or high based on two thresholds of 30% and 70% occupancy. This traffic level is compared with its previous value at the signal to determine the effectiveness of the current action. The reward value ranges from -40 to +40. The simplified logic of reward calculation is given in Table I.

It is worth noting what previous and current traffic levels mean. Assuming the action taken at time t is opening signal X, then, the previous traffic level is the measured traffic before the action was taken. The signal is left open for 16s and then the traffic level is again measured as the current traffic level. The basic idea is that signals which deal with low traffic flow (prev = Low & curr = Low) need not be opened too frequently, hence a negative reward is given for such actions. Whereas, signals which record high traffic levels even after keeping the signal open for 16s need to be opened more frequently. A high positive reward is given for opening these high traffic flow signals. For the intermediate traffic flow signals, the reward is based on the relative occupancies of the signals at an intersection, i.e., the intersection side with a greater number of waiting vehicles is given a higher priority.

TABLE I
REWARD STRUCTURE

Current traffic level at the signal	Previous traffic level at the signal	Reward
Low	Low	-40
Medium/High	Low	-20
Low	Medium	5 - 10
Medium/High	Medium	15 - 20
Low	High	25 - 30
Medium/High	High	40

Next, we formally present the traffic control problem as a reinforcement learning Markov Decision Process (MDP). The state s of the environment is denoted by the array containing the occupancy levels of signals $N1$ to $N15$. The occupancy levels can take any value from 0% to 100%. Thus, there are infinite states in our system.

$$s = \{n1, n2, n3, \cdots, n14, n15\} \tag{1}$$

where, ni is the occupancy level of signal i, and $0\% \leq ni \leq 100\%$.

The action set A consists of 15 actions, each denoting the opening of signals $N1$ to $N15$ respectively. In our approach, we use a 4-phase signalling system wherein, during each phase, inbound traffic from one of the sides of the intersection is allowed to flow for a fixed duration of 64s. Our agent chooses which side (phase) to allow at each time step. As mentioned earlier, the intersections are operated cyclically, and hence, the available actions at any time t depend upon the intersection that is being operated. It should be noted that at any given time, all the intersections will have one of their signals as green.

$$A = \{a1, a2, a3, \cdots, a14, a15\} \tag{2}$$

where, ai denotes giving green to signal i.

The reward R ranges from -40 to 40 depending on the effectiveness of the performed action in clearing the waiting traffic and mitigating future traffic jams. Each time step t is equal to 16s. The problem is formulated as a continuing task, where the objective is to maximize the long-term discounted return G_t, with the discount factor $\gamma = 0.6$.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \tag{3}$$

IV. DESCRIPTION OF THE DEEP RL APPROACH USED

The technique used in this paper is based on distributional reinforcement learning. We call our traffic controller ITS-QRDQN. We first introduce the concept of distributional RL, and how it is more effective in tackling certain types of problems.

What we are trying to achieve in the traffic problem is to find the optimum action at any given state to maximize the long-term reward. In other words, we are dealing with the *control problem* of reinforcement learning. The parameter that is used to measure the effectiveness of an action a at state s is the Q-value $Q(s, a)$. The higher the Q-value, the higher is the effectiveness of the action at the state s. The action corresponding to the maximum Q-value at a given state is termed the optimal action for state s

$$a^* = argmax_a Q(s, a) \tag{4}$$

The Bellman optimality equation provides a method to obtain $Q^*(s,a)$. It would provide us with $|S| \times |A|$ number of equations, where, $|S|$ and $|A|$ are the cardinalities of the state and action spaces.

$$Q^*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma max_{a'} Q^*(s',a')] \quad (5)$$

However, it is easy to see that for a system with a large number of states, this system of equations quickly becomes unsolvable even with the aid of modern computing resources. This calls for the need of a method to estimate $Q^*(s,a)$ indirectly. Methods such as SARSA, Q-learning, Monte Carlo, and Dynamic programming are not suitable for us since they either rely on the environment dynamics being known or are constrained to finite-state systems.

Since we have an infinite number of states in our problem, we need techniques that output an estimate of $Q(s,a)$ given the characteristics of the state s and the action a. Many such proven techniques already exist ranging from Linear function approximation to non-linear approximators such as Deep Q-Networks, Deep Recurrent Q-networks, etc.

The basic principle of the above techniques is the use of Neural Networks (NNs) to estimate the Q-values. Then, Generalized Policy Iteration (GPI) can be used to iteratively estimate the optimum policy. Fundamentally, distributional RL estimates the value distribution, $Z(s,a)$, instead of $Q(s,a)$.

The neural networks can be designed to take as input the *features* of the state s and the action a and to output the estimate $Z(s,a)$. They can also be designed to take only the features of the state s as input and output $Z(s,a)$ for all possible actions a in the state s. We use the latter architecture in our approach (see Fig. 3). The features of a state are the observable and unique characteristics of the environment when it is in a state. Choosing the characteristics to model into the feature of a state can hugely influence the performance of the algorithm, and is a specialization on its own called feature engineering. In our approach, we use a simple model where the state is represented by the number of vehicles waiting at the 15 signals.

Correlating the neural network in Fig. 3, with our problem, the inputs $x1$ to $x15$ correspond to the number of vehicles standing at the 15 signals (state features). The outputs $y1$ to $y75$ correspond to the estimates $Z(s,a1)$ to $Z(s,a15)$, where ai denotes the action of giving green to signal i (5 outputs per action).

We now focus on the significance of the value distribution and methods to represent the distribution. The value distribution $Z_\pi(s,a)$ is a random variable representing the sum of discounted rewards observed along one trajectory of states while following policy π. For a fixed policy π,

$$Z_\pi(s,a) = \sum_{t=0}^{\infty} \gamma^t R_t \quad (6)$$

Taking the expectation of the value distribution gives the much familiar Q-value,

$$Q_\pi(s,a) = E[Z_\pi(s,a)] \quad (7)$$

This brings us to the question of the advantage of estimating the complete distribution over estimating the expected value. There are two main advantages. Firstly, it is important to realize that the expected value masks certain valuable information of the distribution such as the negativity of the reward for a certain action, the range of the reward that can be expected, etc. For example, in the sample distribution shown in Fig. 4, the expected value (marked in red) turns out to be completely unindicative of the actual rewards that are to be expected. This problem can exist in any multimodal distribution but the severity can vary. It can be argued that the expected value is misleading the agent by ignoring the positive reward peak, even though sometimes it may be favourable to take the risk of taking that action, in anticipation of the significant positive reward.

This brings us to the second advantage - risk sensitivity. Even though the expected value of two actions may be identical, their variances might be very different. The higher the variance, the higher is the uncertainty and the risk in the outcome of the action. Sometimes, it may be wise to choose an action with lower risk. This is particularly true in the case of our traffic problem, where taking the wrong step in a heavy traffic scenario can snowball into a traffic jam. This helps us avoid risky traffic deadlock situations. Our results show that we can completely avoid deadlocks, while other predictive models such as n-step SARSA suffer from them.

Although the value distribution is a continuous curve, we can practically estimate only a discretized version of it, and in fact, it turns out to be sufficient as well. There are many ways to represent discrete distributions. The most prominent is the one used by the C51 algorithm wherein, the value distribution curve is described by $N = 51$ fixed equally spaced supports z_1 to z_{51} of corresponding heights p_i. The values of p_i are to be learned. Although giving promising results, the caveat with this approach is that the range of the discounted reward is to be known apriori for fixing the positions of the supports z_i. If the supports do not cover the full range of

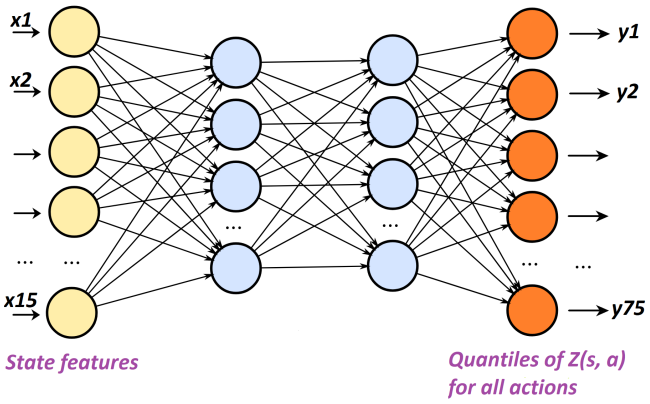

Fig. 3. The Neural network architecture used in our algorithm.

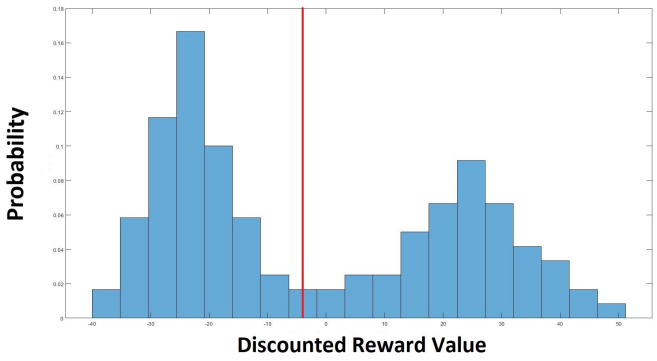

Fig. 4. The value distribution depicts the probability of occurrences of the discounted reward (G_t) values. The probability of occurrences sum to 1.

the discounted reward for all state-action pairs, it can lead to suboptimal performance. QR-DQN circumvents this issue by instead keeping the probabilities fixed, and the positions of the N supports, z_i, adjustable and to be learned. The probabilities p_i are all kept equal to $\frac{1}{N}$, and hence, the supports are termed quantiles as they divide the distribution into N equiprobable regions. Quantile regression is used to estimate the quantiles of the distribution.

We aim to perform quantile regression using neural networks due to their many advantages. Neural networks, which are based on the Stochastic Gradient Descent (SGD) method, need an unbiased sample gradient to guarantee convergence to a local minimum of the loss function. Our original quantile parametrization of z_i, unfortunately, does not guarantee an unbiased sample gradient of the quantile regression loss [13]. To get around this limitation, we perform a re-parametrization, where we map each state-action pair to a uniform distribution on N supports.

Formally, we define a mapping $\theta : S \times A \rightarrow \mathbb{R}^N$ such that the new distribution Z_θ satisfies

$$Z_\theta(s,a) = \frac{1}{N}\sum_{i=1}^{N} \delta_{\theta_i(s,a)} \tag{8}$$

where, δ_x denotes a Dirac at x [13].

To quantize how effective the projection in (8) is, we use the Wasserstein metric with $p = 1$. This metric is a measure of the distance between two probability distributions. Let Z be the Cumulative Distribution Function (CDF) of the value distribution we are trying to approximate, and Z_θ be the projected uniform distribution as defined in (8). Then, the Wasserstein distance with $p = 1$ (termed 1-Wasserstein distance) is defined as

$$W_1(Z, Z_\theta) = \sum_{i=1}^{N} \int_{\tau_{i-1}}^{\tau_i} |Z^{-1}(x) - \theta_i| dx \tag{9}$$

where, τ_i are the quantiles of our distribution with $\tau_i = \frac{i}{N}$ for $0 \leq i \leq N$. And Z^{-1} is the inverse CDF of our distribution.

It can be shown that the values of θ_i that minimize the Wasserstein distance in (9) are those that correspond to the midpoints of the quantiles τ_i [13]:

$$\theta_i = Z^{-1}(\frac{\tau_{i-1} + \tau_i}{2}) \tag{10}$$

We also denote the *quantile midpoints* as $\tau_i^* = \frac{\tau_{i-1}+\tau_i}{2}$. Hence, our original problem of estimating z_i has now been transformed into that of estimating θ_i, where, θ_i are the midpoints of the desired quantiles, and not the quantiles themselves. For example, if we desire $N = 4$ quantiles, then, τ_i = 0.25, 0.5, 0.75 and 1. We would aim to estimate the θ_i corresponding to the quantile midpoints τ_i^* = 0.125, 0.375, 0.625, and 0.875 through quantile regression, which is explained below.

We take a slightly modified version of the standard quantile regression loss called the quantile Huber loss for performing quantile regression, as described in [13]

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} \rho_{\tau_i^*}(z_i - \theta_i) \tag{11}$$

where,

$$\rho_\tau(u) = |\tau - I_{\{u<0\}}| L(u) \tag{12}$$

and $L(u)$ is the Huber loss with $\kappa = 1$ defined by

$$L(u) = \begin{cases} 0.5u^2 & \text{if } |u| \leq 1 \\ |u| - 0.5 & \text{otherwise} \end{cases} \tag{13}$$

Setting $J(u)$ as the loss function of the neural network and training over several episodes would yield θ_i which would uniformly quantize the distribution Z. Note that the loss $J(u)$ is calculated at the quantile midpoints τ_i^* so that the 1-Wasserstein distance gets minimized according to (10).

At this juncture, we need to remind ourselves that our objective was to find the optimal solution for our traffic control problem. We can use DQN to directly find the optimal value distribution using the below distributional Bellman optimality update target, which we recognize as the counterpart of the Q-learning update target.

$$TZ(s,a) = R(s,a) + \gamma Z(s', a^*) \tag{14}$$

$$a^* = argmax_{a'} Q(s', a') \tag{15}$$

In our case of the projected quantile distribution Z_θ, we can transform the update target to

$$T\theta(s,a) = R(s,a) + \gamma \theta(s', a^*) \tag{16}$$

where, a^* is the same as in (15). Also, $Q(s', a')$ can be computed using (7), which in our case can be approximated to be proportional to the average of the quantile values. The proportionality constant can be dropped since it does not affect the *argmax* function.

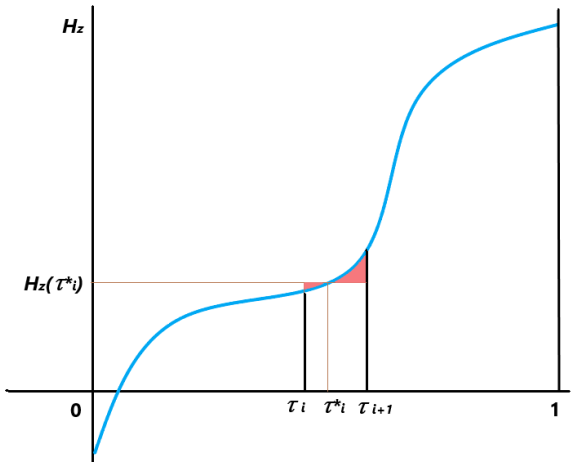

Fig. 5. Depiction of 1-Wasserstein loss (area of the region shaded in red) for the value distribution between quantiles τ_i and τ_{i+1}. Here, H_z is the inverse CDF, Z^{-1}.

$$Q(s', a') \approx \frac{1}{N} \sum_{i=1}^{N} \theta_i(s', a') \tag{17}$$

During the training phase, we choose a^* in an epsilon-soft manner, with the epsilon decaying at the rate of $\frac{1}{t}$ every 1000 steps.

The error term in (11) is now replaced by $T\theta(s,a) - \theta(s,a)$ according to the update rule in (16). The $T\theta$ and θ are computed by two different but identical neural networks to avoid instability. The weights of the two networks are synchronized every 500 time steps. After training, the output of the θ network should directly yield the $\theta(s,a)$ of the optimum policy.

Now that we have the quantiles of our distribution after the training, we test our algorithm using a *live policy* that takes into account the risk. Note that risk is not considered during the training phase. We measure the risk as the probability of negative reward, which we calculate from the quantile values, $\theta(s,a)$, as the area under the curve below zero reward in Fig. 4. We again note that the area between two quantiles will be $\frac{1}{N}$

$$\mu(s,a) = P(Z_\theta(s,a) < 0) \tag{18}$$

The live policy is as follows:

$$\pi_L(s): \ a^* = argmax_a(\aleph(s,a)) \tag{19}$$

where, $\aleph(s,a)$ is an aggregate score giving weightage to both the expected reward and the risk

$$\aleph(s,a) = Q(s,a) + \left(\frac{1}{\mu(s,a)}\right)^{\omega(s)} \tag{20}$$

Here, $\omega(s)$ is the *risk factor* that decides how much weightage has to be given to risk depending on the median occupancy percentages of vehicles waiting at the intersections. Our justification for weighing it as such is that the more crowded the intersections are, the more the chances of a traffic jam, and the more importance risk should be given.

$$\omega(s) = \begin{cases} 1.5 & \text{median occupancy} > 40\% \\ 1 & 40\% \geq \text{median occupancy} > 20\% \\ 0 & 20\% \geq \text{median occupancy} \end{cases} \tag{21}$$

The neural networks we use are with 2 hidden layers, which are both fully connected Linear layers. Layer 1 size is 15 by 256, and Layer 2 size is 256 by *15N*. A ReLU activation function is used in Layer 1 to introduce non-linearity to the quantile estimates. We take the number of quantiles $N = 5$.

V. Simulation Results

We compare the performance of ITS-QRDQN against SS, LQF, and n-step SARSA using the following metrics:

1) *Percentage Queue occupancy*: This metric is the average percentage lane occupancy of the 15 signals in our 4-junction network.
2) *Waiting Time*: This metric denotes the fraction of the journey time that the vehicle spends waiting at signals.
3) *Time Loss*: This metric denotes the time lost by a vehicle due to it driving below the ideal speed.
4) *Dispersion Time*: This metric denotes the time needed to disperse the traffic present in the road network. To measure this metric, traffic is allowed to flow for about 3.3hrs, after which, the time needed to empty the network is recorded.

The performance plots of ITS-QRDQN against SS, LQF, and n-step SARSA are shown in Fig. 6 to 9. In addition, we have also included the curves of the QR-DQN algorithm without taking into account the risk i.e., with $\omega(s) = 0$. It is evident from the plots that when risk is not taken into account, the agent goes for only immediate gains, and ends up performing poorly similar to LQF. This proves that our risk-sensitive approach has significantly improved the native QR-DQN algorithm.

We can observe that in all the graphs, static signalling shows the worst results. This establishes that conventional signalling systems are not very effective. LQF fares better, but no better than any RL-based approach. In Fig. 6 to 9, we can see SARSA performs better than QR-DQN without risk, but worse than ITS-QRDQN. During the dispersion phase (from 200 minutes onwards), we can see ITS-QRDQN performs similar to SARSA in terms of Queue Length. ITS-QRDQN shows a 18.2% and 13.2% lesser vehicle Waiting time and journey Time loss, respectively, as compared to SARSA.

During the steady phase (50-200 minutes), we can see a 5-10% decrease in Waiting time & Time loss, on average. This signifies an overall smooth flow of traffic in the case of ITS-QRDQN. Fig. 9, compares the dispersion times of the traffic control algorithms. Dispersion time values above $\sim$100 minutes signify a deadlock traffic condition. We can see LQF frequently entering deadlock (spikes), whereas both

SARSA and ITS-QRDQN are able to predictively avoid such conditions. ITS-QRDQN registers about 25% lower Dispersion time than SARSA.

Fig. 10, depicts the variation of Huber Loss across the training episodes. It is seen that the Huber Loss continually decreases and converges rapidly in less than 30 episodes. Hence, our algorithm requires a very short training time, making it easily deployable in real-life traffic systems.

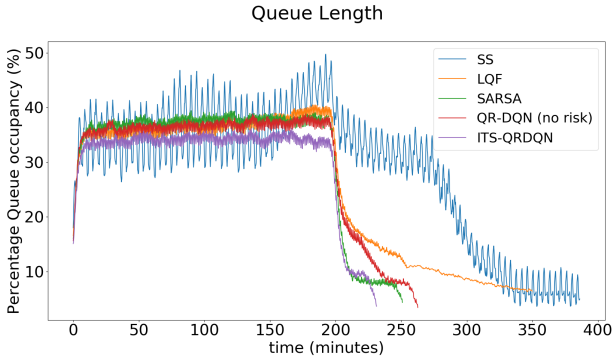

Fig. 6. Queue Length comparison of the traffic control algorithms.

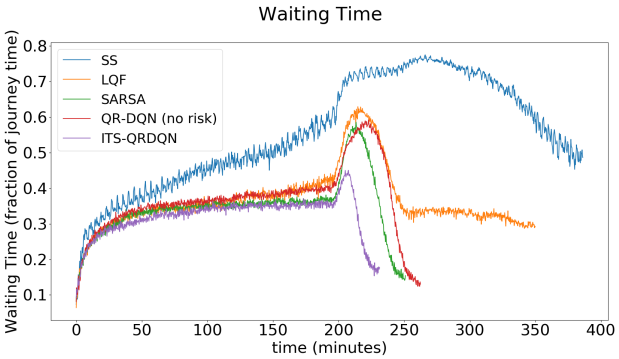

Fig. 7. Waiting Time comparison of the traffic control algorithms.

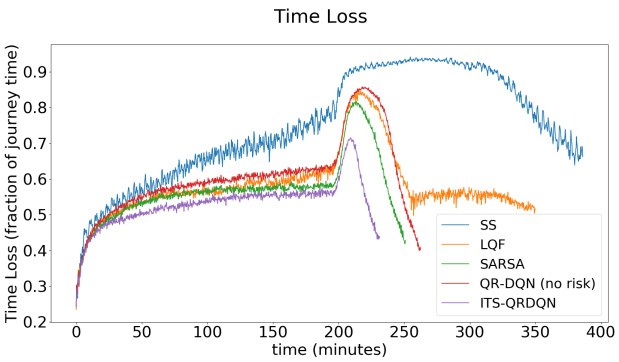

Fig. 8. Time Loss comparison of the traffic control algorithms.

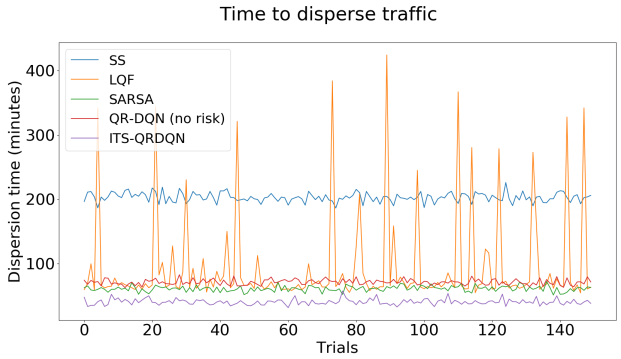

Fig. 9. Dispersion Time comparison of the traffic control algorithms.

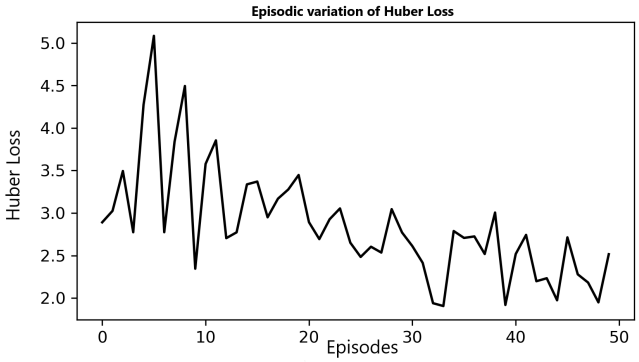

Fig. 10. Episodic variation of Huber Loss.

VI. Conclusion

In this paper, we apply the recently developed deep reinforcement learning method [3] on the traffic control problem and show that it notably outperforms traditional approaches to traffic signal control. The risk-sensitive nature of our algorithm enables it to foresee and avoid traffic situations which may lead to congestion. We believe the combination of deep reinforcement learning methods and AI at the edge systems can improve the efficiency of modern traffic networks.

References

[1] R. S. Sutton, A. G. Barto. Reinforcement learning: An introduction, second edition. MIT press Cambridge, 2018.

[2] Ralph Neuneier, Oliver Mihatsch. Risk sensitive reinforcement learning. In Proceedings of the 11th International Conference on Neural Information Processing Systems, pp. 1031–1037. MIT Press, 1998.

[3] H. Zhong, E. Fang, Z. Yang, and Z. Wang. Risk-Sensitive Deep RL: Variance-Constrained Actor-Critic Provably Finds Globally Optimal Policy, 2021. arXiv:2012.14098.

[4] Mann Shie Mannor and John N. Tsitsiklis. Mean-variance optimization in Markov decision processes. In Proceedings of the 28th International Conference on International Conference on Machine Learning, (ICML'11). Omnipress, Madison, WI, USA, 177–184, 2011.

[5] Peter Geibel. Reinforcement Learning with Bounded Risk. In Proceedings of the Eighteenth International Conference on Machine Learning, 2001 (ICML '01). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 162–169.

[6] NA Urpí, S Curi, A. Krause. Risk-averse offline reinforcement learning. Ninth International Conference on Learning Representations (ICLR 2021). arXiv:2102.05371, 2021.

[7] Yun Shen and Michael J. Tobia and Tobias Sommer and Klaus Obermayer. Risk-sensitive Reinforcement Learning. CoRR, abs/1311.2097, 2013.
[8] Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-learning. arXiv:1509.06461, 2015.
[9] Gu J., Fang Y., Sheng Z., and Wen P. Double Deep Q-Network with a Dual-Agent for Traffic Signal Control. Appl. Sci., 10, 1622, 2020.
[10] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling Network Architectures for Deep Reinforcement Learning. arXiv:1511.06581, 2015.
[11] Xiaoyuan Liang, Xunsheng Du, Guiling Wang, and Zhu Han. Deep Reinforcement Learning for Traffic Light Control in Vehicular Networks. arXiv:1803.11115, 2018.
[12] A. Kekuda, R. Anirudh and M. Krishnan, Reinforcement Learning based Intelligent Traffic Signal Control using n-step SARSA, 2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS), 2021, pp. 379-384, doi: 10.1109/ICAIS50930.2021.9395942.
[13] Will Dabney, Mark Rowland, Marc G Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. arXiv:1710.10044, 2017.
[14] The QR-DQN implementation of Arsenii Senya Ashukha was used as the reference for our implementation. https://github.com/senya-ashukha/quantile-regression-dqn-pytorch.