CITY UNIVERSITY OF HONG KONG
SCHOOL OF DATA SCIENCE

SDSC 8006 Reinforcement Learning
Individual Project

Deep Q-Learning Based Regional Adaptive Traffic Signal Control

Name (Student ID):
Xiao HAN (56190707)

Supervisor:
Chin Pang HO

Abstract

As we know, traffic signals work together to expedite the movement of cars through intersections. Traditional transportation methods use pre-calculated offsets between two intersections to establish cooperation. However, an offset computed in advance does not work well in dynamic traffic situations. Thus, on the basis of the group project, I continued to extend the single-intersection-based signal control method to the multi-intersection situation, and a simulation is used for the experiment.

1 Problem Definition

In the field of traffic management, there is a lot of study taking on. In this project, I aim to apply Deep Q-Learning (DQN) method [1], which is mentioned in the group project, to solve the traffic management problem. I am trying to figure out the best policy to route traffic so that I can reduce regional traffic congestion and waiting times. In this project, I have taken a 4-intersection road network as shown in Figure 1 to test the DQN algorithm.

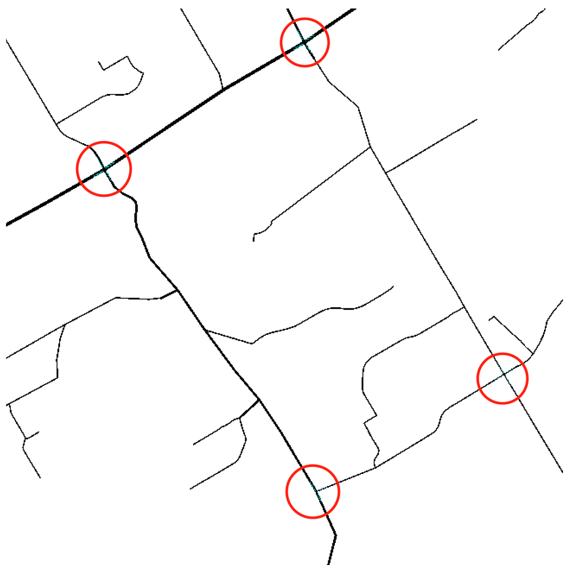


Figure 1: The sketch of the 4-intersection road network

2 Methodology

2.1 Quantile Regression

Rather than learning the anticipated return, distributional RL focuses on discovering the complete distribution of the random variable Z. RL distributions may be represented in a variety of ways. This distribution is represented by DQN as a homogeneous mixture of N supporting quantiles:

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

where δ_x denotes a Dirac function at $x \in \mathbb{R}$, and each θ_i is an estimation of the quantile corresponding to the quantile level $\hat{\tau}_i = (\tau_{i-1} + \tau_i)/2$ with $\tau_i = i/N$, $0 \leq i \leq N$.

The state-action value $Q(s,a)$ is then approximated by Equation (2) as follow:

$$Q(s,a) = \frac{1}{N}\sum_{i=1}^{N} theta_i(s,a), \tag{2}$$

which is referred to as quantile approximation [2]. Then the distributional Bellman optimality operator for signal control as shown in Equation (3) is used, which is similar to the Bellman optimality operator in mean-centered RL.

$$\mathcal{T}Z(s,a) = R(s,a) + \gamma Z(s', \arg\min_{a'} \mathbb{E}_{p,R}[Z(s',a')]),\ s' \sim p(\cdot|a) \tag{3}$$

Based on the distributional Bellman optimality operator, proposed to train quantile estimations (i.e., q_i) via the Huber quantile regression loss. To be more specific, at each time step t, the loss function is:

$$L = \frac{1}{N}\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N}[p_{\hat{\tau}_i}^{\kappa}(y_{t,j} - \theta_i(s_t,a_t))] \tag{4}$$

where $y_{t,j} = r_t + \gamma\theta_{i'}(s_{t+1}, \arg\max_{a'} \sum_{i=1}^{N} \theta_i(s_{t+1},a'))$ and $\rho_{\hat{\tau}_i}^{\kappa}(x) = |\hat{\tau}_i - \mathbb{I}\{x < 0\}|\mathcal{L}_{\kappa}(x)$, where $\mathbb{I}$ is the indicator function and $\mathcal{L}_{\kappa}$ is the Huber loss:

$$\mathcal{L}_{\kappa}(x) = \begin{cases} \frac{1}{2}x^2, & \text{if} x \leq \kappa \\ \kappa(|x| - \frac{1}{2}\kappa), & \text{otherwise} \end{cases} \tag{5}$$

2.2 Deep Q-Learning (DQN)

Q-Learning is based on an iterative update of the Bellman optimality equation, as shown in Equation (6).

$$Q_{i+1}(s,a) \leftarrow \mathbb{E}[r + \gamma \max_{a'} Q_i(s',a')] \tag{6}$$

where it can be shown that this converges to the optimal Q-function, i.e., $Q_i \to Q^{\star}$ as $i \to \infty$

It is not realistic to describe the Q-function as a table of values for each possible combination of s and an in most situations. A function approximator, like a neural network [3], is trained to estimate Q-values ($Q(s,a|\theta) \approx Q^{\star}(s,a)$), rather than the actual Q-values themselves. It's possible to do this by reducing the following losses in step i:

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s' \sim \rho(\cdot)}[(y_i - Q(s,a|\theta))^2] \tag{7}$$

where $y_i = r + \gamma \max_{a'} Q(s', a'|\theta_{i-1})$

Here, y_i is the temporal difference (TD) target, and $y_i - Q$ is the TD error. ρ denotes the behavior distribution over transitions s, a, r, s' collected from the environment.

Note that the parameters from the previous iteration θ_{i-1} are fixed and not updated. Instead of using the most recent iteration's network settings, I utilize a snapshot from a few iterations back. This snapshot is called the *target network*.

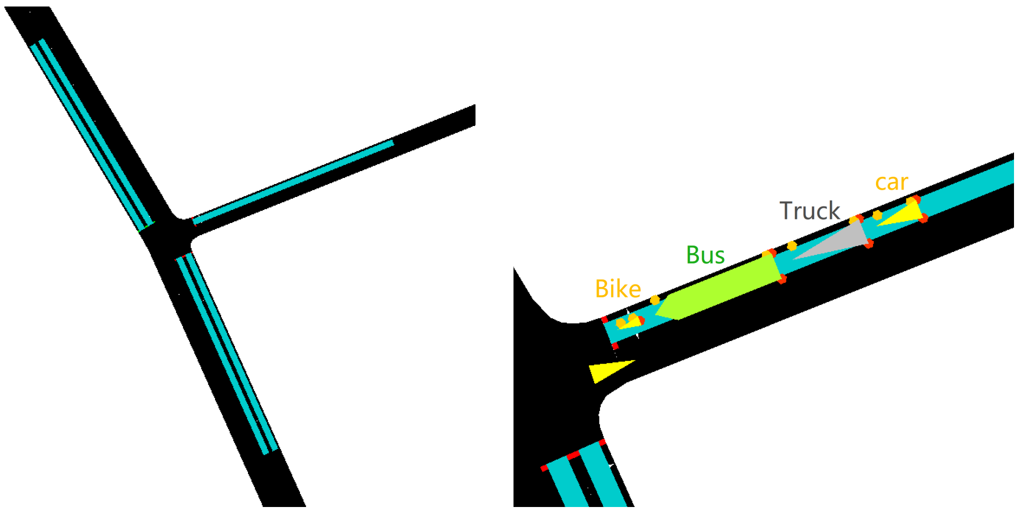
As an off-policy algorithm, Q-Learning is one that learns about the greedy policy of $a = \max_a Q(s, a|\theta)$ while behaving in the environment and gathering data with a separate behavior policy. This behavior policy is generally a greedy strategy that picks the ϵ–greedy action with a probability $1 - \epsilon$ and a random action with a probability ϵ to provide excellent coverage of the state-action space.

3 Experiment

3.1 Experimental Setting

The traffic lights shown in the Figure 3 are in use in our environment. The four crossroads have a total of 15 traffic lights (N1–N15). At a 16-second interval, intersections in my work are cycled through (INT1 $>$ INT2 $>$ INT3 $>$ INT4 $>$ INT1). The RL agent determines which signal to use at each junction. Reduce Queue Lengths, Waiting Time, Time Loss, and spread traffic over the network as quickly and efficiently as possible are the RL agent's goals.

I used Simulation of Urban Mobility (SUMO) software to simulate the traffic scenarios, which can generate different traffic patterns as the input data related to the given road network. The SUMO environment details are as follow:



(a) Induction loops of one intersection (b) Different vehicles in the road network

Figure 2: The settings of SUMO environment

- **Induction loops**: To measure the quantity of traffic that is waiting at the intersections, I employ E2 detectors (lane area detectors) that span a distance of up to 70 meters, as the blue area shown in Figure 2(a). Then, the amount of space occupied by moving vehicles is calculated using the data collected by these sensors.
- **Types of vehicles**: In the simulation, the traffic consists of 4 types of vehicles: bikes (41.7%), cars (37.5%), trucks (12.5%), buses (8.3%), which is modelled close

to real-world urban traffic, as shown in Figure 2(b), The top speeds, acceleration, and braking times of each of these vehicles vary.

- **Traffic density and pattern**: Binomial distribution with $n = 5$ and $p = 0.15$ is used to generate vehicles entering the network. Besides, the number of lanes of a road also affects the distribution of cars, with multi-lane routes seeing more traffic.

Then I compared Deep Q-Learning algorithm to the following ones and found that it performed better:

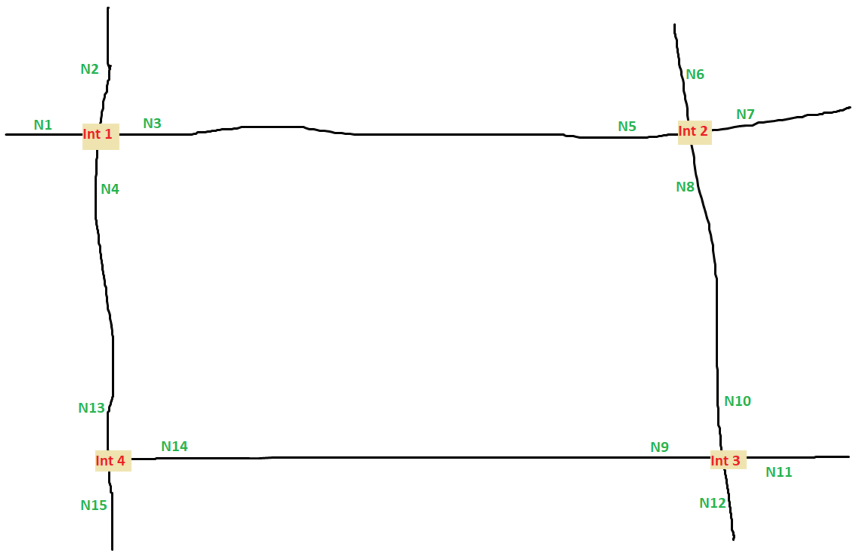


Figure 3: The road network

- **Fixed Signaling**: Each intersection has its own set of traffic lights that are controlled in a round robin method (e.g. every 16 seconds, the signals change). In my experimental setting, the green light order was (N1, N5, N9, N13), (N2, N6, N10, N14), (N3, N7, N11, N15), (N4, N8, N12, N13) etc.

- **Longest Queue First**: Decisions are made at intersections based on the side with the longest queues, and more green time is given to the side that has the longest queues. Again, in accordance with our context, I work on intersections in a recursive manner.

As for the DQN I used, the details of this model is as follows:
States: A 15-side congestion level is used to represent the whole road network states, which means for each intersection, I use a percentage vehicle occupancy to describe the intersection congestion.

Actions: I have a total of 15 actions, which correspond to the signal that is to be made Green, or keeping Red.

Reward: I have experimented with two different reward structures:

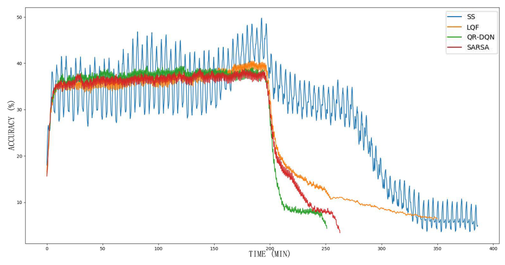
- Queue lengths based reward, at each time step t, it shown as follow:

$$R_t = -\sum_{j=1}^{M} q_{t,j} \tag{8}$$

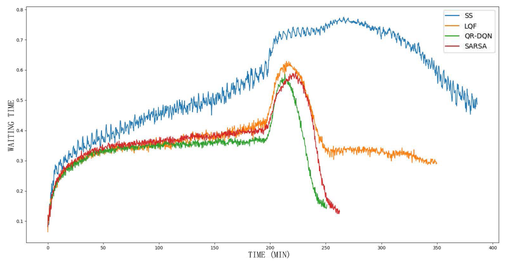
  where $q_{t,j}$ is the queue length of side j at time t.

- The change in percentage vehicle occupancy.If traffic from side N_i goes to side N_j, and if both sides are crowded, then N_i shouldn't't be opened. This could gives consistent results.

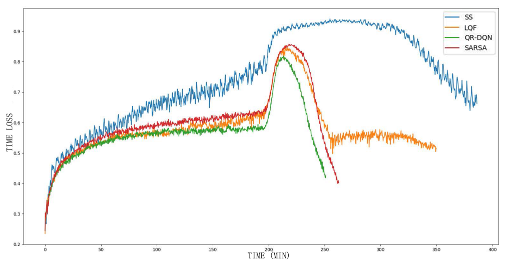
3.2 Experimental Result



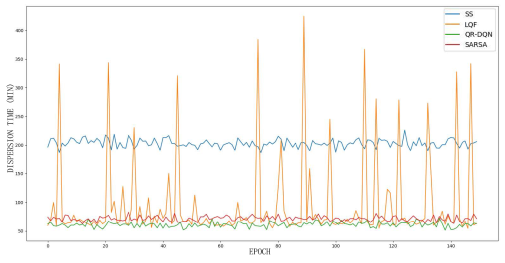
(a) Percentage vehicle queue



(b) Waiting time



(c) Time loss



(d) Dispersion time

Figure 4: The experimental results

I train the Deep RL agent under different traffic patterns each run. Based on the trained NN, I run a Live algorithm to evaluate the performance, that chooses greedy actions based on the trained weight for a simulation. The result can be shown as follow in four metrics: Percentage vehicle queue occupancy, waiting time, time loss and dispersion time.

4 Conclusion

In all measures, DQN surpasses Static Signalling (SS), Longest Queue Factor (LQF), and n-step SARSA (SARSA), as seen in the Figure 4 above.

Static signaling, which is the most common kind of traffic signaling used in cities, performs poorly. Because it is not adaptable, it has a large variance.

The network takes roughly 150 minutes to attain steady state traffic flow in terms of queue length. When DQN reaches steady state, it begins to outperform both SARSA and LQF. In steady state, it improves by 7.5% when compared to LQF. DQN disperses traffic 28.5 percent quicker than SARSA during the dispersion phase (200 minutes and above).

The network takes roughly 100 minutes to establish steady state traffic flow in terms of Time Loss. When DQN reaches steady state, it begins to outperform both SARSA and LQF. In both the steady state and dispersion phases, it outperforms LQF and SARSA by 12.9%.

Finally, when it comes to dispersion time, we see that LQF has many spikes. When traffic comes to a full stop owing to congestion, this is referred to as a stalemate scenario. Both DQN and SARSA never reach a stalemate condition (0%) because to their capacity to detect and avoid such circumstances, as seen in the charts. On all of the experiments, DQN had the shortest dispersion times.

References

[1] Y. Liao and X. Cheng, "Study on traffic signal control based on q-learning," in *2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery*, vol. 3, 2009, pp. 581–585. DOI: 10.1109/FSKD.2009.539.

[2] W. Dabney, M. Rowland, M. Bellemare, and R. Munos, "Distributional reinforcement learning with quantile regression," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[3] L. Prashanth and S. Bhatnagar, "Reinforcement learning with function approximation for traffic signal control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 2, pp. 412–421, 2010.