Traffic Signal Control based on Reinforcement Learning

Chixuan Zhang, Hairui Yin, Yida Qiu, Yukun Qi,

ShanghaiTech University
393 Middle Huaxia Road, Pudong, Shanghai
zhangchx@shanghaitech.edu.cn, yinhr@shanghaitech.edu.cn
qiuyd@shanghaitech.edu.cn, qiyk@shanghaitech.edu.cn

Abstract

We realize and compare several traffic controlling methods based on Random, Greedy Algorithm, naive Q-learning, Periodic Cycle and deep Q-network. The simulation environment of our project is built on Simulation of Urban Mobility (SUMO) where Road traffic can be visualized and vehicles and road information can be easily extracted. In our methods, the evaluation criteria are set according to the total waiting time of all cars in the simulation and the length of different lanes. Greedy Algorithm modulates traffic light according to lanes length. Naive Q-learning updates q value according to state and action. Periodic Cycle is the same as our real world where the period is determined by experience. In our code, the period is selected after tests. Deep Q-network (DQN) is the combination of Reinforcement Learning (RL) and Deep Learning (DL). The achieved results show that DQN algorithm can produce intelligent behavior.

1 Introduction

Traffic congestion is a complex and pervasive issue facing highly urbanized countries around the world. By addressing traffic congestion, it is possible to reduce fuel consumption, decrease delays, and improve logistics transportation. Traditional traffic light control policies often rely on historical data and are adjusted according to specific times of day, but these methods are often ineffective and struggle to reach optimal performance. As a result, drivers may find themselves waiting at red traffic lights even when no other vehicles are present. However, the development of vehicular communication technologies offers an opportunity to gather more detailed information about vehicles, which can be used in conjunction with computational methods to improve traffic flow through more efficient traffic light control policies. In the future, the integration of autonomous driving technologies may further enhance the performance of these policies.

Markov Decision Processes (MDPs) and Traditional Reinforcement Learning (RL) frameworks are popular approaches for optimizing traffic light control policies as they can achieve optimal performance through techniques such as policy iteration, value iteration, and linear programming. However, both MDPs and RL frameworks suffer from issues related to high dimensionality, resulting in large state spaces and slow convergence in real-world problems. The optimal performance of these approaches is therefore difficult to attain due to the significant computational power required. To address these challenges, we propose the use of Deep Q-Networks to optimize a realistic traffic light control problem and achieve a local optimal solution. We explored the potential of Deep Q-Networks in this application by comparing them to other methods (including Random, Period Control, Greedy Algorithm, ϵ-Greedy, Naive Q-learning) and identifying other potential ways to improve traffic light control strategies.

1.1 External Libraries

SUMO SUMO, Simulation of Urban Mobility, is an open source, micro, multimodal traffic simulation software developed in 2000. It is purely microscopic and can be controlled individually for each vehicle, so it is very suitable for the development of traffic control models. SUMO can give each vehicle a unique identifier, departure time, and route, as well as customize its lane, speed, or location.

PyTorch PyTorch is one of the most popular deep learning frameworks. It is simple, efficient and fast. To keep our project relevant to the course, PyTorch was used only to help us implement a Multi-Layer Perceptron (MLP) and assist in training the model.

2 Related Work

There has been growing interest in using reinforcement learning (RL) to optimize traffic signal control methods. However, many existing machine learning approaches do not adequately compare with classical transportation methods, which have been largely ignored. Hua Wei (2020) conducted a survey that takes a comprehensive view of both machine learning and transportation engineering in order to facilitate interdisciplinary research and bridge this gap.

Several classical transportation methods have been identified for comparison as baselines for RL-based methods, including Webster, GreenWave, Maxband, Actuated, SOTL, Max-pressure, and SCATS. These methods use various inputs, such as traffic volume, phase sequence, lane length, speed limit, and queue length, to generate cycle-based signal plans for individual intersections and offsets between adjacent traffic signals. Webster generates a signal plan for a single intersection using traffic volume and phase sequence, while GreenWave and Maxband determine offsets for a cycle-based signal plan using traffic volume, lane length,

speed limit, and a cycle-based signal plan at individual intersections. Actuated and SOTL use traffic volume, phase sequence, and a phase change rule to determine the transition to the next phase. Max-pressure control aims to minimize the "pressure" of phases at an intersection in order to balance queue length between neighboring intersections and reduce the risk of over-saturation. SCATS uses pre-defined signal plans and performance measurements to adjust signal plans for all intersections based on traffic volume.

Several artificial intelligence techniques have been proposed for use in traffic signal control, including fuzzy logic algorithms, swarm intelligence, and reinforcement learning. Examples of these approaches include the use of fuzzy logic algorithms [Gokulan and Srinivasan 2010], swarm intelligence [Teodorović 2008], and reinforcement learning [ElTantawy 2012; Kuyer 2008; Mannion 2016; Wiering 2000].

3 Problem Statement

We model the road intersections and formulate our optimization problem. For the sake of transparency, we consider an admittedly stylized simulation that only aims to capture the most essential features that govern the dynamics of contending traffic flows at road intersections. We throughout adopt a discrete time formulation to simplify the description according to our iteration and allow comparision of all methods based on the same total cars amount and lane-based speed control.

3.1 Crossroad

Our simulation starts with a single-intersection scenario to facilitate the validation of RL algorithms by comparing with all used approaches. We consider the simplest meaningful setup with directional crossroads traffic flows as schematically depicted in the picture.

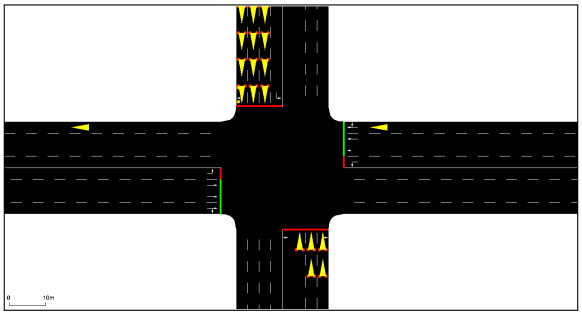


Figure 1: Crossroad

This is a traffic intersection with a total of four roads at the junction, of which each road has eight lanes, four heading towards the junction and four heading outwards. Vehicles in the outward lane can pass directly, while vehicles in the inward lane need to follow the traffic signal. There are four kinds of inward lanes with different legal actions:

- 0: turn right or go ahead
- 1,2: only go ahead
- 3: only turn left

3.2 Traffic Light Control

In reality, most traffic lights change in the same direction, meaning that the opposite road of an intersection will change green at a time. So in our simulation, we follow the same guidelines and have four traffic light operations:

- Lights in straight lanes of left and right roads are green, the rest are all red
- Lights in leftturn lanes of left and right roads are green, the rest are all red
- Lights in straight lanes of up and down roads are green, the rest are all red
- Lights in leftturn lanes of up and down roads are green, the rest are all red

Note that we only consider the red and green light conditions, not the yellow light condition. Traffic lights also change in an instant.

3.3 Evaluation criteria

During our simulation iteration, we can obtain information of all vehicles and roads. Considering our real life, traffic congestion happens when queue is long and every vehicle has to wait for a long time. And the car at the front of congestion may have shorter waiting time while waiting time of cars at the end is larger. As a result, in our code, we consider the total waiting time of every vehicle and the variance of cars' waiting time as evaluation criteria.

$$T_S = \sum_{i=1}^{N} T(i)$$

$$\sigma^2 = \frac{\sum_{i=1}^{N}(T(i) - \bar{T})}{N}$$

where T_S is the total waiting time, $T(i)$ is the waiting time of vehicle i, σ is the variance, N is the total amount of vehicles and $\bar{T}$ is the average waiting time.

3.4 Optimization Goal

We assume that after all iterations, the total waiting time T_S and the waiting time varianceσ are calculated. The goal is to a dynamic control policy which selects actions over time so as to minimize T_S and σ.

4 Method

4.1 Q-Learning

Q-leaning is a model-free reinforcement learning technique proposed by Watkins el al. in 1989. It has been proved that Q-learning will eventually find an optimal strategy for any finite Markov Decision Processes (MDPs).

In Q-learning, actions are selected based on their expected reward r_t. The goal is to choose the action that will result in the maximum reward for a given state. The iterative process of value function is the core of Q-learning.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot \Big[r_t + \gamma \max_{\pi} Q(s_{t+1}, a_t) - Q(s_t, a_t) \Big]$$

Where α is the learning rate and γ is the discount factor. α helps to avoid getting stuck in a locally optimal solution.

Higher alpha values result in faster changes to Q-values. And γ determines the importance of future rewards by multiplying the estimated optimal future value in the Q-learning equation. In the Q-learning equation, the new Q-value for a state is calculated by adding the selected action's Q-value to the old Q-value.

However, when facing state space in high dimesion, it's easy for Q-learning to meet problem of Dimensional curse and memory difficulties. Unfortunately, most data in the real world is high dimensional. For example, in our task, we only consider the number of cars on each road (16 roads in total), and suppose there are at most 15 cars on one road at a time, the state space will be $15^{16} \approx 6.57 \times 10^{18}$. At least right now, it's impossible for us to solve this problem. And the real problem is more complex and even has continuous features. Hence, we need to consider some approximate methods.

4.2 Q-learning with ϵ-Greedy

The ϵ-greedy action selection method combines exploitation, in which the agent makes use of prior knowledge to maximize reward, with exploration, in which the agent searches for new options. Most of the time, the ϵ-greedy approach will select the action with the highest estimated reward. However, with a small probability epsilon, the agent will choose to explore by selecting an action randomly, regardless of its estimated reward. This allows the agent to try new things and potentially discover better options. Through infinite trials, the ϵ-greedy action selection policy is able to discover the optimal actions. With this idea, we can update the value iteration equation as below:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot [r_t + \gamma \cdot \mathbb{Q} - Q(s_t, a_t)]$$

$$\mathbb{Q} = \begin{cases} \max_\pi Q(s_{t+1}, a_t), & \text{with probabilty } 1-\epsilon \\ Q(s_{t+1}, a_{\text{random}}), & \text{with probabilty } \epsilon \end{cases}$$

4.3 Deep Q-Network

To deal with the weakness of Q-learning about high dimensional data. We can use a neural network to approximate the value function. $\hat{Q}_\theta(s,a) \approx Q(s,a)$ The value iterative process can be denoted as:

$$\hat{Q}(s_t, a_t) \leftarrow \hat{Q}(s_t, a_t) + \alpha \cdot \Big[r_t + \gamma \max_\pi \tilde{Q}(s_{t+1}, a_t) - \hat{Q}(s_t, a_t) \Big]$$

Where $\hat{Q}(s_t, a_t)$ is the main neural network in training, and $\tilde{Q}(s_{t+1}, a_t)$ is the targetNet (more contents below). A good model should be able to achieve the most likely output action is the decision that makes Q-value be best.

Model Snapshot In order to avoid the problem that model doesn't convergence, we do not use the real-time update model, but use a snapshot of model. In other words, every K steps, the model is stored and fixed as a snapshot for computing ideal Q, which is also called target network.

Experience Replay MultiLayer Perceptron (Deep Neural Network) is a supervised learning model, which asks the data should be independent and identically distributed. Experience Replay breaks down the correlation by the store-sampling method. Save the data from the environment agent and then sample from the data.

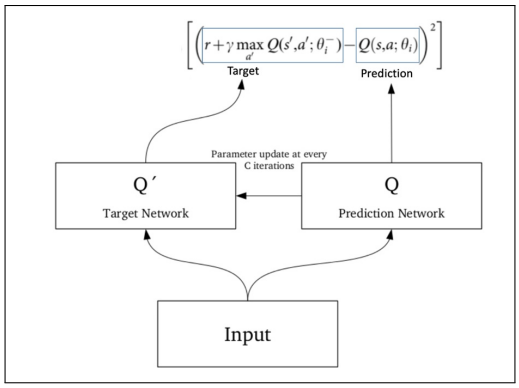


Figure 2: Deep Q-Learning Framework

4.4 DQN Algorithm Design

We provide a detailed pseudo-code and detailed parameters setting of the DQN algorithms for the scenarios of a single intersection as described in the previous section.

To ensure relevance to the course content, we will use a simple Multi-Layer Perceptron as the neural network to fit the $Q(s_t, a_t)$. The MLP has four hidden layers of 400 nodes, and the input is a 16-length tensor, which is the number of cars on the 16 roads. And the output is a 4-length tensor to give the score of 4 actions.

$$Loss(\theta) = \mathbb{E}\left[\Big(r + \gamma \max_{a'} \tilde{Q}(s', a') - \hat{Q}(s, a) \Big)^2 \right]$$

The loss function is as above, where r is the reward in the current step, r is calculated by the difference between the total waiting time of the current time step and the next. γ is the discount factor for acceleration of convergence. And $\tilde{Q}(s', a')$ is the target model, which is a copy for the main model $\hat{Q}(s, a)$ for each 1 step.

Algorithm 1: Deep Q-Network Algorithm for SUMO

Require: Q_θ: the model need to be learned;
Ensure: Q-value interative equation is satisfied
initial target model $\mathbb{Q} = Q_\theta$;
for epoch $<$ MAXEPOCH **do**
  while SUMO is runing **do**
    $\epsilon = 1 - \text{epoch}/\text{MAXEPOCH}$
    a_t = epsilonGreedy(ϵ)
    s_t = getfeatures()
    $q_t = \mathbb{Q}(s_t, a_t)$
  end while
  for i $<$ MAXITERATION **do**
    for batch in Experience Replay **do**
      LOSS($Q_\theta(s_t), a_t, q_{t+1}$)
      OPTIMIZER(Q_θ, learning_rate)
    end for
  end for
  Every 1 Epoch, Update $\mathbb{Q} = Q_\theta$.
  SAVE CheckPoints Q_θ
end for

5 Experiment

In this section, we quantitatively evaluate traditional method based on empirical periodic setting. And four different periods are used. Following existing period work, we test all proposed methods as well as the competitive baselines. We first present the general experiment setups, and then describe task-specific evaluation protocols and discuss the results in each section.

5.1 Experiment Setup

Baselines The traditional method in traffic light control field is Periodic Cycle control. Four different Periodic Cycle parameters are set. The parameters with the best simulation performance is chosen as the baseline.

5.2 Experiment Detail

Evaluation Criteria As mentioned in 3.3, the total waiting time T_S of all vehicles and the variance of time σ between each car are recorded and set as the Evaluation Criteria. T_S measures the overall performance of the method. And σ measures fairness between vehicles, a higher σ means there exists some vehicles having to wait for a long time.

Multiple experiments For every method, multiple tests with different random seed are operated in order to reduce the impact of randomness. The result taken into consideration is the average result of each method.

DQN HyperParameters The complete Deep Q-Network Algorithm and loss function is in section4.4. Here are some other concrete hyperparameters. MAXEPOCH is set as 100. MAXITERATION is set as 800. SUMO will be run for 5400 ticks in one epoch. And we choose Adam algorithm (which is the only content in our project that is not related to the course) to be the optimizer with learning rate $1e-3$. One epoch needs about 40s on a 6900xt gpu.

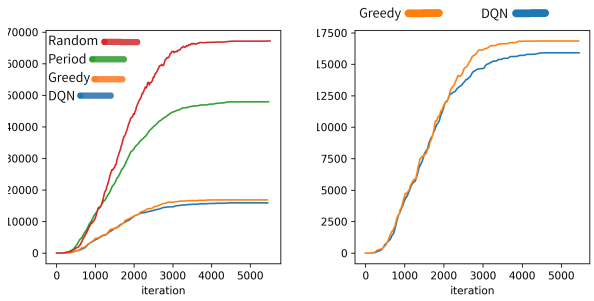
5.3 Experiment Result

Table 1: Results on different method

Method Name	T_S	σ
Deep Q-Network	15908.0	461.22
Greedy	16850.0	834.02
Period Cycle	47942.0	1681.86
Random	62800.0	868.04
SUMO buildin	75467.0	839.59

The result is summarized in Table 1 and has been sorted according to their performance. In the result, under the same vehicle circumstance, Deep Q-Network performs best, both in total time and variance. Greedy has a similar total time, but a much larger variance. Period Cycle, as a baseline, has a relatively average performance, of which total time is approximately three time larger than these two methods above but better than completely python determined random and SUMO buildin random. And the periodic method has the largeset variance.

Waiting Time Analysis During iteration, the accumulative waiting time T_S is in Figure 3. A figure with only performance of DQN and Greedy is intended to show the difference between them more clearly.



(a) All methods (b) DQN and Greedy

Figure 3: Performance of total waiting time

Waiting time demonstrates the total cars waiting time of each control policy. Slope of Periodic Cycle and Random is much higher than slope of Greedy and Deep Q-Network, meaning that the congestion problems in these two policy are more serious. There is a little difference between Greedy and Deep Q-Network and Greedy has a worse performance. Greedy considers only the current optimal situation, but ignoring some cars which have waited for a long time. Intuitively, because the total number of cars in left turn lane is less than the main road, hence greedy algorithm sacrifices a few cars for the good of all. With gui of simulation, the left turn lane's traffic light is always red during rush hour.

Variance Analysis During iteration, the variance σ is in Figure 4. It's clear that DQN method achieves quite excellent results, compared with other methods.

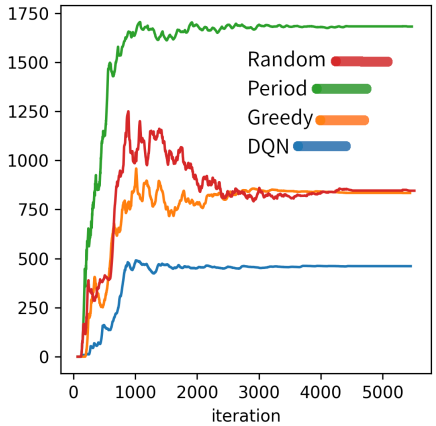


Figure 4: All methods variance

Variance demonstrates fairness for each vehicle. Above all methods, period method performs worst. It is largely due to the sudden increase of some lanes, but the traffic lights have no reaction to that instant change. Greedy have a lower variance because it can deal with the problem of instant longest vehicle lanes Random performs poorly for every vehicle. And Deep Q-Network performs best, which shows that it can distributes the burden waiting time to all vehicles in a more just way. With gui of simulation, we can clearly find that both greedy and random are hard to properly handle rush hour. Especially greedy algorithm will ignore the left turn lane during rush hour due to there are fewer cars.

6 Conclusion

We have studied the potential of Q-learning, Q-learning with epsilon-greedy, and deep Q-networks (DQN) to optimize real-time traffic signal control policies in intelligent transportation systems. By comparing the performance of these algorithms in a single-intersection scenario, we found that DQN has the best performance in controlling the overall waiting time and variance of waiting time. Naive Q-learning, on the other hand, is impossible to deal with large state spaces and is difficult to converge. While greedy algorithm can achieve good performance, it may sacrifice the waiting time of drivers in the left turn lane (a large variance of waiting time) when there are more cars on the other roads.

In this paper, we focused on the optimal performance achieved in a single-intersection scenario. In reality, it is also important to optimize the signal timings for adjacent Intersection's traffic signals, because the traffic flow at adjacent intersections is correlated. Failing to do so can lead to decisions at one signal that negatively impact traffic operations at another. In the future, we plan to investigate the use of reinforcement learning for more realistic and wide-scale scenarios of traffic lights control. Additionally, different vehicle types (e.g. bus), different speeds of vehivles, pedestrains, security problem (countdown before the light change) and so on, all of these factors are needed to be taken into consideraton when facing a complex reality problem.

Appendix

As required, at the end of our report, we will list all the external resources (e.g., code, library, tools) that we use and explain how/why we use them.

SUMO SUMO, Simulation of Urban Mobility, is what we use to simulate urban transportation. It provide the Graphical User Interface (GUI) and Application Programming Interface (API). SUMO provide a python library "traci" as the API. It plays the role of "environment" in reinforcement problem.

PyTorch As mentioned in section 1.1. PyTorch is one of the most popular deep learning frameworks. It is simple, efficient and fast. To keep our project relevant to the course, PyTorch was used only to help us implement a Multi-Layer Perceptron (MLP) and assist in training the model.

Numpy NumPy, Numerical Python, is an open source extension of Python for numerical computation. It provides a large number of mathematical function libraries for array operations. We use this library for some common function "argmax" function.

tqdm "tqdm" module is a python library for progress bar. Without the progress bar, although the code can run normally, but can not provide real-time feedback, blind waiting will cause anxiety, we use "tqdm" to keep us happy.