A Reinforcement Learning Approach to Optimize Traffic Signal Control

Xiangyu Li, Stephen Tseng, Gabriel Gomes

Abstract*—This study presents a comparative analysis of reinforcement learning (RL) algorithms for optimizing urban traffic signal control. With increasing post-pandemic traffic, unoptimized signals have worsened congestion, resulting in delays and higher emissions. Using the traffic simulation tool SUMO, we implemented and evaluated three approaches: the Max Pressure (MP) algorithm, Deep Q-Learning (DQN), and Independent DQN (IDQN). MP serves as a baseline, dynamically adjusting signal phases based on local traffic pressure. DQN, an RL algorithm, uses a neural network to learn optimal traffic signal strategies for reducing vehicle waiting times. IDQN extends DQN by allowing multiple independent agents to control different intersections without communication. Simulation results show that DQN performs similarly to MP for single intersections but is less effective in multi-intersection networks. This paper proposes a novel method to reduce vehicles' average waiting time via traffic signal control for a chosen multi-intersection road segment with real-world traffic flow data. The proposed deep reinforcement learning (DRL) based signal timing algorithm is compared against the classical Webster method as the control group. Specifically, the control group adopts the Webster method for traffic signal timing; experimental groups 1 and 2 use DRL-based methods DQN and A2C, respectively. In contrast, IDQN outperforms MP by reducing waiting times and CO2 emissions by up to 80%. Future work could explore more sophisticated multi-agent systems and apply the algorithms to real-world networks to validate their effectiveness.

***Index Terms*—Traffic Signal Control, Reinforcement Learning, Urban Traffic.**

I. Introduction

WITH more travel coming back in the post-COVID world, traffic is once again a pervasive and universal issue. This is especially true in urban areas, with unoptimized traffic signals contributing to this issue. The purpose of this project was to build a traffic control system that could handle traffic pressure efficiently for multiple intersections. To accomplish this task, several algorithms were used to run traffic simulations on a software tool called SUMO(Simulations of Urban MObility). In SUMO, traffic networks are constructed with demand and can be simulated to model the performance of traffic controllers. The algorithms utilized in this study were the Max Pressure algorithm, Deep Q-Learning (DQN), and Independent DQN(IDQN). The Max Pressure algorithm deterministically selects the traffic lights with the highest pressure, while the DQN is a reinforcement learning (RL) algorithm that allows an agent or multiple agents to select the phase to maximize a reward. The IDQN is an extension of the DQN that has multiple independent agents that each control one intersection, but have a shared memory pool. The programming language Python was used to create both controllers and interact with SUMO.

Xiangyu Li is the Department of Civil and Environmental Engineering, University of California, Berkeley, United States

Stephen Tseng is the Department of Industrial Engineering and Operations Research, University of California, Berkeley, United States

Gabriel Gomes is Department of Mechanical Engineering, University of California, Berkeley, United States

Once trained, the agents were run in a traffic simulation, and the waiting time, CO2 emissions, and total vehicles per step were extracted. Results from experimentation show that the DQN performed similarly to the Max Pressure algorithm for a single intersection, but approximately 3 times worse when extended to multi-intersections. The IDQN in the multi-intersection network outperformed the Max Pressure algorithm by 80%. For future work, experimentation on the size of the neural network and design of the reward system could be attempted. The algorithm could also be extended to a more complicated and robust multi-agent algorithm to control a larger traffic network. Lastly, the best performing algorithm could be implemented into a real traffic signal network.

The code used in this project can be found in the following Github repository: https://github.com/Sunglass2225/AI-Traffic

II. Problem space

Emerging and developed countries alike are plagued with urban traffic problems, such as traffic congestion, accidents, and environmental problems caused by the concentrated populations and economic growth. (Hiroshi, 2018) These problems are only becoming worse and will need to be addressed in the near future. According to Vital Signs, a study done by the Metropolitan Transportation Commission (MTC) and the Association of Bay Area Governments (ABAG), time spent in highway congestion in the Bay Area has been gradually increasing over the years as shown in Figure 1. This results in higher CO2 emissions and decreased air quality that impacts public health.

Researchers from areas such as mathematics, physics and engineering have attempted to find solutions to traffic by using advanced methods and techniques for their own fields and areas of expertise. (Orosz, 2010) However, there has been no consensus on the most optimal way to solve this issue.

In recent years, due to the development of computation capabilities and algorithms, more complicated and sophisticated models have been developed and used. Reinforcement learning algorithms and neural network structures provided novel approaches to solving complex problems, such as traffic congestion. Not only have these advances provided various possible solutions, but combined with traffic simulations, they are cheap and easy to be experimented with, without negatively impacting actual transportation networks. In addition to being more optimized for constant demand traffic flows, these algorithms show a more dynamic feature, allowing them to react to ever-changing traffic levels, which strengthen its ability as a more well-rounded solution.

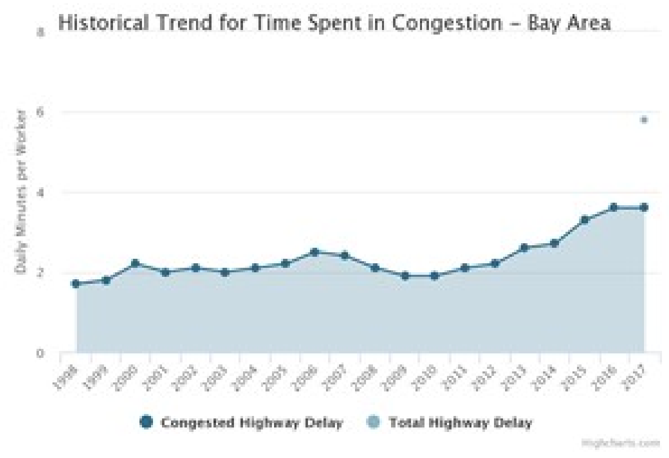


Fig. 1. Historical Trend for Time Spent in Highway Congestion - Bay Area (Source: https://www.vitalsigns.mtc.ca.gov/time-spent-congestion) .

By experimenting and training different reinforcement learning algorithms in the traffic simulation, this paper compares it with traditional traffic algorithms, demonstrating the advantages and disadvantages and also evaluating the practicality of applying reinforcement learning algorithms to solve traffic congestion issues.

1) A main road controlled by three traffic lights is chosen as the studied scene
2) States are selected as vehicle position and velocity at intersection exits
3) A total of 4 actions can be chosen from, with 4 transition intervals between each pair of actions
4) Rewards are set as per vehicles' average waiting time
5) MSE is used as the loss function with Adam as the optimizer

III. Review of Literature

A. Max Pressure

A significant source of traffic is unoptimized traffic signals. Traffic lights often run on a fixed timing plan, with the signal giving a predetermined amount of green time to each phase. However, in 2013, the first use of the Max Pressure Algorithm (MP) in the context of traffic was done by Varaiya; the journal article describes MP as a means to stabilize the traffic network using only local information at an intersection(Varaiya, 2013). Varaiya describes that MP traffic controller selects the active traffic phase by subtracting the number of downstream vehicles divided by the road segment length from the number of upstream vehicles by the road segment length. Liu and Gayah improved upon the version of the Maximum Pressure algorithm presented by Varaiya by integrating a reward and punishment function (Liu and Gayah, 2023). This version of MP maximizes vehicle performance (overall average speed, queue length, and minimum speed) throughout the system and minimizes the overall network travel time.

In the article, "Investigating Max Pressure Traffic Signal Timing", Minnesota's Transportation Research group describes that the max pressure algorithm detects real-time data, including vehicle's queue length, enter time, and exit time, to optimize the signal control. This algorithm also adapts a mathematically proven model to calculate a pressure value for each phase, activating the one with the greatest pressure and dynamically adjusting the phase sequence within seconds based on real-time traffic measurements (Michael, 2023). Ben Hao, a traffic operation engineer at Hennepin County, stated that this method has been used in seven county intersections, and has demonstrated decreased travel delays and increased vehicle throughput with an adaptive signal control strategy (Hao, 2021). Many improved MP algorithms have been raised recently to further improve performance of the original MP algorithm, including Total-delay-based Max Pressure (also known as a decentralized signal control algorithm). The results show that this proposed algorithm outperforms all three baseline models, including the original MP, Position Weighted Back-Pressure, and Delayed-Based MP algorithms in both reducing traffic delay and increasing delay equity when the penetration rate is less or equal to 60% (Liu and Gayah, 2023). Despite these benefits, MP still requires a fixed refresh rate and can only obtain current information, meaning the solution may not be truly optimal. This was a driving factor for the consideration of another approach to traffic signal control for better performance and efficiency in this study.

B. Deep Q-Learning Network (DQN)

According to "Reinforcement Learning With Deep Q-Learning", Loeber states that the Q-learning algorithm is a value-based approach based on the concept of a Q-Table (Loeber, 2022). This table contains values of expected rewards from the states and actions taken. To make the process more scalable, Deep Q-learning replaces the Q-Table with a Deep Neural Network (DNN), with the differences shown in Figure 2. A neural network is a machine learning method that allows a program to learn, similar to a human brain, with nodes and layers connected by linear abstractions.

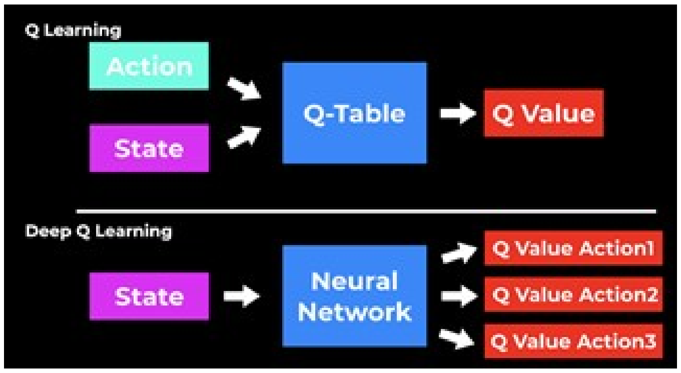


Fig. 2. The difference between Q-Learning and Deep Q-Learning visualized(Loeber, 2022).

Further explained, Q-learning uses an explanatory policy to generate samples, to maximize the Q-value (Q), given the inputs are the state (S) and action (A), for obtaining a deterministic optimal target policy. The equation is formatted as

$$Q\left(S,A\right)=R\left(S,A\right)+\gamma*max_{A}Q\left(S',A\right) \quad (1)$$

where the next state is denoted as S' and discount rate is denoted as γ(Sutton and Barto, 2018). In general, there are three steps to setup the Q-learning network: initialize the Q-table, choose an action based on the reward and punishment functions within the system, update the Q-table based on the Bellman equation, denoted as

$$Q\left(S_t,A_t\right)=\left(1-\alpha\right)Q\left(S_t,A_t\right)+\alpha*\left(R_t+\lambda*max_{\alpha}Q\left(S_{t+1},a\right)\right) \quad (2)$$

with a being the learning rate, Rt as the reward, λ as the discount factor, and S_{t+1} as the next state (Wang, 2021). The steps are repeated until the maximum traffic performance is achieved.

The use of the neural network in the DQN can help estimate the function $Q^*\left(S,A\right)$ without having to calculate all Q-values of the actions. The network design takes the input state, S, from the observation in the environment. After the input layer, there are several hidden layers with neural nodes. These are used to capture the weights of the neural network, and combining them results in the policy of the algorithm. The output of the network are Q-values, which are combinations of expected rewards of the present action and the future action. As the neural network processes the input received, it outputs the Q-values related to the action. The agent then selects the action that results in the highest Q-value to perform in the simulation.

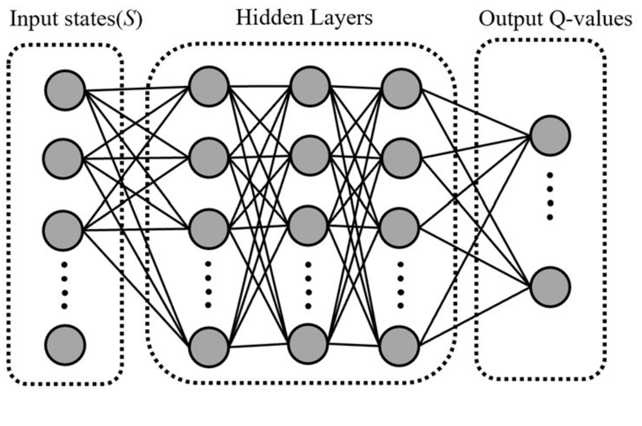


Fig. 3. DQN neural network.

After initializing the neural network, the agent starts the training process and collects data from the simulation. To start a learning step, the agent retrieves the current state data, (S_t,A_t,R_t,S_{t+1}),, puts it in the neural network and then selects the highest Q-value action, A_t. After simulating with the action, the agent retrieves the reward R_t and the next state data S_{t+1} . The agent then stores these data as memory, (S_t,A_t,R_t,S_{t+1}), in the replay buffer, which finishes one learning step.

The deep neural network (DNN) learns to map inputs to outputs given a training dataset of examples from the replay buffer. With the training data, the agent learns to minimize the mean square error (MSE), given by

$$MSE=\frac{1}{N}\sum_{i=1}^{N}\sum_{i=1}^{N-1}\left(y_i-\hat{y}_i\right)^2 \quad (3)$$

where $y_i$and $\hat{y}_i$are the predicted value of the model and true value, and N is the number of samples (Mishra, 2022). When the model takes all the training data and updates the weights of the neural network, a large N, makes the required computational resources of the MSE significantly large.

Since DQN can be integrated into most traffic models without heavily modifying the existing system or structure, it is expected that such a method can be implemented in real-life applications and obtain optimal results once its reward and punishment functions are fully developed.

C. Independent Deep Q-Network (IDQN)

Independent Deep Q-Networks (IDQN) is a reinforcement learning algorithm that extends the popular Deep Q-Network (DQN) algorithm for multi-agent settings, combining it with the concept of independent Q-learning. The IDQN algorithm is a relatively new method and was initially introduced in a paper called "Deep Reinforcement Learning for Multiagent Systems: A Review of Challenges, Solutions, and Applications," by Thanh et al. (2017). The authors proposed a decentralized approach to multi-agent reinforcement learning, where each agent has its own Q-network that is trained independently. Unlike centralized approaches, where all agents share a single Q-network, IDQN allows each agent to learn its own Q-function based on its own observations. In the IDQN algorithm, agents treat other agents' actions as part of the environment and do not require communication or coordination with other agents. This approach can simplify the learning process and scale to larger multi-agent systems. Therefore, it is believed by the researchers in this paper that by applying such computational algorithms to existing traffic systems, more sophisticated and efficient training agents can be designed to control traffic signals.

Another article used by the researchers in this paper discussing the extended application of DQN is called "Multiagent Cooperation and Competition with Deep Reinforcement Learning" by Ardi Tampuu et al. (2015). In this article, the authors propose two rewarding schemes, which are fully competitive and fully cooperative (Tampuu, 2015). In the result, it shows that IDQN is suitable in certain situations, which it is capable of reaching an acceptable result in the fully cooperative scenario but failed in the fully competitive scenario. By integrating such rewarding schemes into the computational algorithm, the authors of this study believe that greater efficiency and accuracy during the agent-training procedure will be obtained, along with more optimal training statistics.

D. Related Works

There has been a significant amount of research done using RL for traffic controllers. LA and Bhatnagar (2011) found that RL algorithms outperformed traditionally used methods for traffic controllers such as fixed timing. Troia et al (2021) found that the DQN algorithm was the best RL controller. Prabuchandran et al. (2014) found that multi-agent Q-learning algorithms worked best for large systems

IV. System Model

The above figure depicts the multi-intersection road segment adopted in this study. It comprises a two-way four-lane main road, with the right lane for through and right turns and the left lane for through and left turns, and two-way two-lane side roads whose lanes are used for left turns, through and right turns. Vehicles can enter and exit the main road through the side roads.

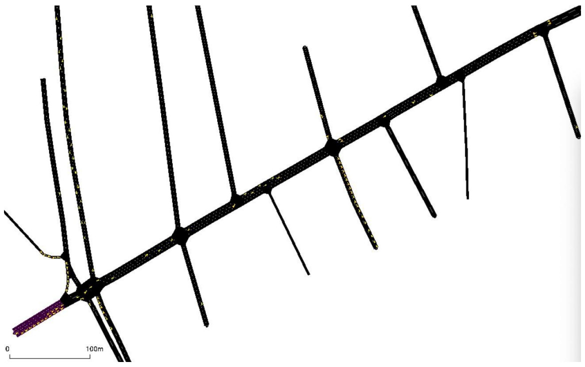


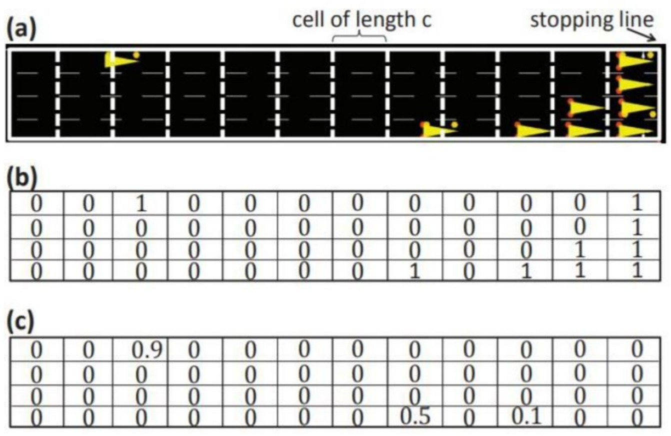
Fig. 4. multi-intersection road segment.

A. High-level design of DRL paradigm

The traffic signal control problem is formulated as a reinforcement learning problem, where an agent interacts with the intersections at discrete time steps t=0,1,2 with the goal of reducing the long-term duration that vehicles stay at intersections. Concretely, this agent first observes the intersection state St (defined later) at the beginning of time step t, and then selects and triggers a traffic signal At. Once the vehicle has moved under the activated traffic signal, the intersection state becomes a new state St+1. For its decision on the choice of the traffic signal, the agent also receives a reward Rt (defined later) at the end of time step t. This reward can serve as a signal to guide the agent towards its goal. In terms of a time sequence, the interactions between the agent and the intersection can be defined by ..., St, At, Rt, St+1, At+1.... The following sections will provide the definitions of the intersection state St, agent action At and reward Rt.

B. State

The state configurations of the intersections are illustrated in the figure above. For each edge, the position and velocity matrices in (b) and (c) are generated, with the velocity normalized by speed limit. Hence, the intersection state is defined by two matrices, one for position and one for velocity, which are given by:



0	0	1	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	1	0	1	1	1

0	0	0.9	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0.5	0	0.1	0	0

Fig. 5. The state configurations.

$$P = \begin{bmatrix} P_0 \\ P_0 \\ P_0 \\ P_2 \end{bmatrix} V = \begin{bmatrix} V_0 \\ V_2 \\ V_3 \end{bmatrix} \tag{4}$$

C. Action

The intelligent traffic light (agent) observes the current state and chooses one of the following two actions: (0) turning on the green light for through and right turns in the east-west direction; (1) turning on the green light for through and right turns in the north-south direction; (2) turning on the green light for left turns in the east-west direction; (3) turning on the green light for left turns in the north-south direction. Every time a state change is required (i.e., the horizontal road is on green light and the agent decides to turn on the green light for the vertical road, or vice versa), transition to the subsequent state is necessary prior to the execution of the state-changing action. Transition will involve the following:

1) Changing light for through and right-turn vehicles to yellow.
2) Changing light for through and right-turn vehicles to red.
3) Changing light for left-turn vehicles to yellow.
4) Changing light for left-turn vehicles to red.

Among them, the green light lasts 10 seconds and the yellow light lasts 3 seconds. As such, at each time step, the agent may decide to stay in the same state or change to a new state.

D. Reward

The agent is rewarded or penalized at each time step for its attempt to reduce congestion by lowering vehicles' waiting time at intersections. Two observations are made at each time step, one at the beginning of the first time step and one at the end of the second time step. Vehicles' waiting time is recorded in each of the two observations, resulting in two observations values r1 and r2. Eventually, the total reward R is given by R=r1-r2. With the described settings, the agent is penalized if its action leads to an increase in waiting time, and is rewarded for decreased waiting time.

E. Agent objective

In the long run, the agent aims to reduce vehicles' waiting time at intersections. By observing the state, the agent decides to act according to a certain action policy . The experience of each agent is stored in an experience replay buffer, from which the agent randomly samples the experience of all agents. When selecting an action, each agent makes its selection in accordance to its own observation state. Given its goal of reducing vehicles' long-term waiting time, the agent ought to find an action policy that maximizes the following cumulative future reward Π^*:

$$\begin{aligned} a_i(s,a) &= \frac{1}{e}\left[n_i + n_1 R_{+1} + i^2 R_{n+2} + 2\cdots|S_1 = 0,\right. \\ A_1 &= a_1, \pi_3 = \sum_{1}^{\infty}\sum_{k=1}^{n} R_{R^2_{(-1)}s_i=3, A_1=0, \pi_i} \end{aligned} \tag{5}$$

F. DQN algorithm

There apparently exist an infinite number of available states to be used. Hence, it is infeasible to store the Q value for each state, and an approximation technique is needed to estimate the Q value for a given state. In general, the optimal Q value satisfies the following recurrence relation, known as the Bellman optimality equation:

Shown in the figure above, the network takes as input the observed states (i.e., P, V, L). The first layer of convolutional network has 16 4x4 filters with a stride of 2 and ReLU activation. The second layer has 32 2x2 filters with a stride of 1 and ReLU as its activation function. The third and fourth layers are fully connected layers of size 128 and 64, respectively. The last layer is a linear layer that outputs Q-values corresponding to each possible action taken by the agent.

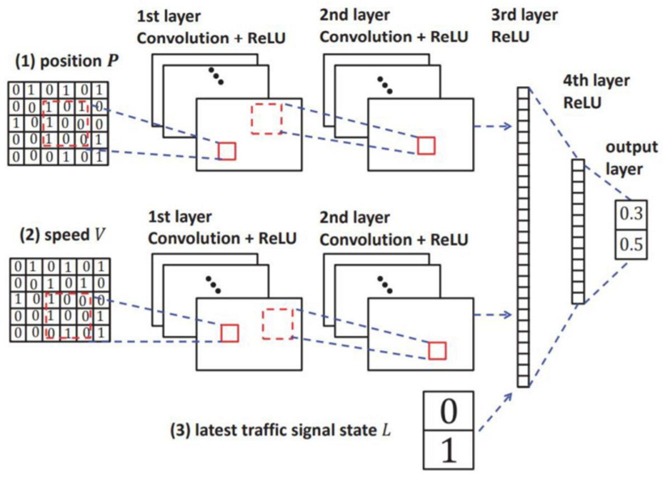


Fig. 6. DQN Networks.

For network training, the neural network is first initialized with random weights. At the beginning of each time step, the agent observes the current time step St, forms the input to the neural network, and performs the action At with the highest cumulative future reward. Upon performing an action, the agent receives the environmental reward Rt and the next state St+1. The agent stores this experience (St, At, Rt, St+1) in memory. Due to the limited memory size, the oldest data will be deleted as the memory space becomes full. The deep neural network (DNN) is trained by fetching training samples of type (St, At) from memory. With the fetched training data, the agent learns the feature as the DNN network is trained to minimize the mean square error (MSE), given by:

$$MSE(\theta) = \frac{1}{m}\sum_{i=1}^{m}\{(R_t + \gamma_{mi}\alpha_i(S_{i+1}, a'; \theta')) - Q(S_t, A_t; \theta)\}^2 \tag{6}$$

where m is the size of the input dataset. Due to a large value of m, the computation of $MSE(\theta)$ is relatively expensive. Hence, we choose the RMSProp optimizer with a minibatch size of 32.

V. Research Approach

As mentioned in the previous section, the dynamic signal control algorithm has the potential to further resolve traffic congestion issues and be more flexible for different kinds of scenarios. Multiple algorithms had been accepted to solve this issue in the field and there had been more new methods developed that also had the potential. This section illustrates more details on how methods mentioned in the previous section were utilized in this paper, and also includes several additional implementations of the reinforcement learning algorithms.

The Max pressure algorithm is chosen as a baseline model in this paper since it is already a fairly robust and simple algorithm that can deal with congestion problems by dynamic traffic signals. This paper explores the potential of reinforcement learning, DQN and IDQN, and compares their performance with Max pressure.

A. Max Pressure

The basic idea behind the max pressure algorithm is to give priority to the movement with the highest "pressure" at each intersection. As stated previously, the pressure of a turning movement or traffic phase is a measure of the number of vehicles that are waiting to make a movement, such as turning left or going straight through an intersection, minus the number of vehicles downstream from the traffic signal. The more vehicles that are waiting upstream and the less vehicles downstream, the higher the pressure. In this paper, the pressure was implemented by the following following formula:

$$P_i = (Up\_Num_i \ / \ Up\_Len_i) - (Down\_Num_i \ / \ Down\_Len_i) \tag{7}$$

where P denotes pressure, i denotes the pair of lanes selected to calculate, Up_Num_i and Up_Len_i denotes the number of vehicles and length of the upper lane in the pair, and $Down\_Num_i$ and $Down\_Len_i$ denotes the number of vehicles and length of the down lane in the pair.

At each intersection, the algorithm calculates the pressure for each movement and then selects the movement or phase with the highest pressure to be given a green light. This process is repeated for each refreshment of the traffic signal, where the refresh is set by the user. For this project, the refresh time was set to 10 seconds. Advantages of the max pressure algorithm are the straightforward approach and adaptability, as the signal timing changes as demand changes. However, the cycle lengths are fixed, thus improvements can be made on this controller.

The MP traffic controller serves as a baseline model to which the DQN and IDQN controllers are compared.

B. Reinforcement Learning – single-agent DQN

The Agent objective during the training progress is to maximize the overall reward it receives in each epoch. By retrieving state information from the traffic simulation, the agent selects an action according to a certain action policy. The experience (state, reward, and action) is stored in a replay buffer, which has max length at 500 experiences. Instead of using the latest state to adjust the action policy, the policy is changed by using a random sample set from the replay buffer, establishing a new weight in the neural network to fit the input and output. To update the neural network more efficiently, this research team chose a smaller batch size at 32 for the model training.

To further enhance the performance of the DQN algorithm, this research project created two separate models, one for doing predictions and one for tracking "target values". To prevent overestimation during the process of updating the neural network, the target network remained the same, while the main network was updated according to the reward it received. This was to ensure when doing the training at each time step, the Q-function would not fluctuate dramatically. The target network was updated to the same state as the main network after the update. A decay exploration process was also established during the start of each training process. The agent started with an exploration rate of 100% and gradually decreased over the process, encouraging the agent to explore random action when it is initiated.

The reward for this agent is calculated by cumulating each lane's vehicle waiting time in every time step after the action is assigned. After 10 time-steps, before the agent assigns its next action, the algorithm records the reward that the agent receives in this training step and the current state to the replay buffer. Agents would alway receive a negative reward in this setting and to increase the total reward it receives in each epoch.

C. Reinforcement Learning – Independent DQN

For Independent DQN (IDQN) agents, the objectives are still to maximize the overall reward it receives in each epoch. However, instead of allowing one single agent to control the whole network, the IDQN contains several agents which control one intersection independently. The reason IDQN was included is that the agent only controls its own intersection, so the number of actions for each agent is decreased, thus making the action space simpler and the controller was expected to have a faster learning rate. The learning rate was also expected to be faster because agents share the same replay buffer and memory, allowing them to learn from each other's experience. After iterations of memorizing previous actions, taking actions and getting reward, each agent finds the policy that maximizes the reward in each episode.

The neural network design for Independent DQN is similar to that of Single DQN; the main difference is that IDQN is a decentrial structure, where each agent G observes the partial state of the environment, stG, selects an individual action, atG, and receives a individual reward, rtG. By extending DQN with independent Q-learning, each agent G independently and simultaneously learns its own policy during the training process. The memories generated by all agents are then stored in the same replay buffer, so when the agent is updating its neural network it can learn from others' experience. The same decay exploration process as DQN was also set up in the IDQN to allow random exploration when it was initiated.

The reward function for this algorithm was initially the waiting time (the same as the single-agent DQN). However, upon training the IDQN, the algorithm did not converge. To remedy this, the difference in waiting time was used as the reward function. This is further elaborated in Section 5.3.

VI. Reinforcement Learning Algorithm Design

This section illustrates the process that was taken to set up the DQN and IDQN algorithm in this study. The design of the simulation environment and action space for the agent to train in were first established. There are two environments set up in this project, the one-intersection and multi-intersection. The first one is for testing whether the reinforcement learning algorithm is able to perform as good as the Max pressure in the simplest environment. The second one is to test the reinforcement learning algorithms' performance in a more complicated environment. The neural network and reward function of the agent were then constructed. The algorithms were then trained in repeated simulations and finally tested.

A. Traffic network design in SUMO

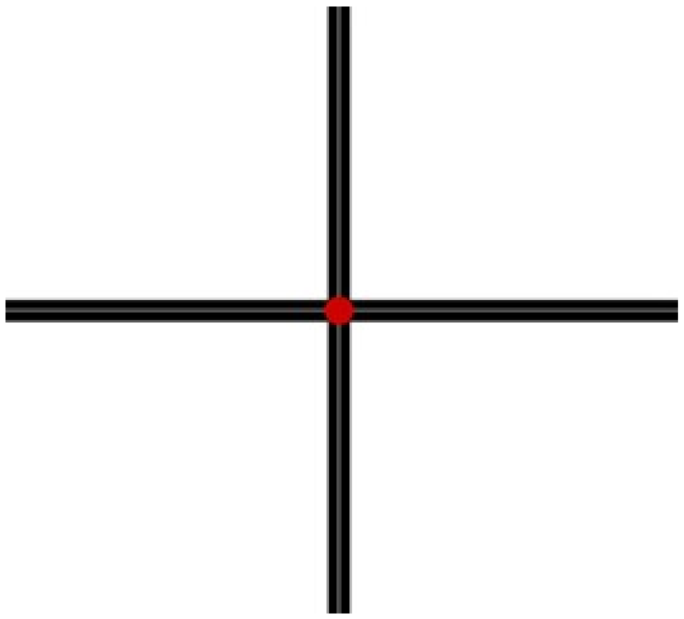


Fig. 7. SUMO one-intersection network design.

1) One-intersection traffic network design: Figure 4 presents the one-intersection road network adopted in this study for the first training stage of reinforcement learning.

It contains 4 edges, 5 nodes, and 1 traffic light intersection. Each edge's length is roughly 200 meters, containing two through lanes. Vehicles can enter and exit the system through the nodes.

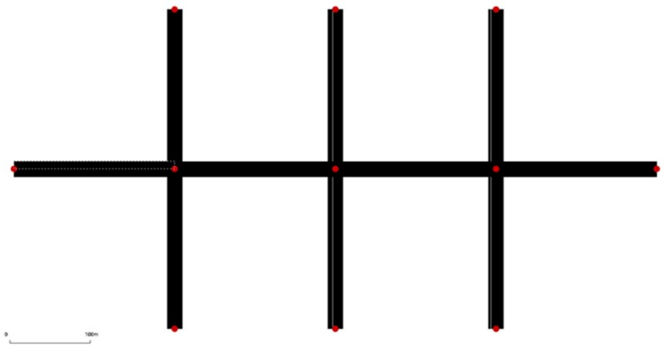


Fig. 8. SUMO Multi-intersection network design.

2) Multi-intersection traffic network design: Figure 5 presents the multi-intersection road network adopted in the second stage of this study, training the agent in a more complicated situation similar to a corridor on a real traffic network. It contains 10 edges, 8 nodes, and 3 traffic light intersections. Each edge's length is 200 meters, containing two through lanes and one left-turn lane. Vehicles enter and exit the system through the nodes.

B. State and Action design

1) One-intersection DQN's action space: For the one-intersection network, the signal design of the traffic light assigns each through lane together, resulting in all 8 lanes that the traffic light intersection control being grouped into 2 different traffic light phases as shown below, only having North-South through and East-West through.

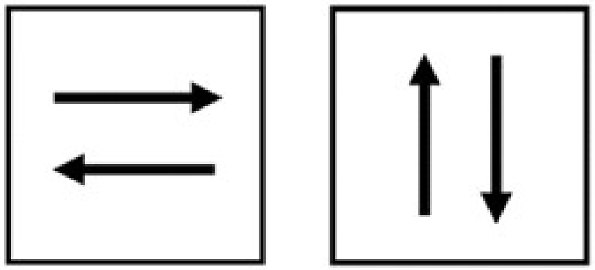


Fig. 9. One-intersection action space.

For the multi-intersection network, the signal design for each traffic light intersection each pair of directions assigned into a traffic light phase, grouped by through and left-turn directions. This results in all 12 lanes traffic lanes being controlled by different traffic light phases as shown below, including turning green for East-West through roads, East-West left turning roads, North-South through roads, and North-South left turning roads.

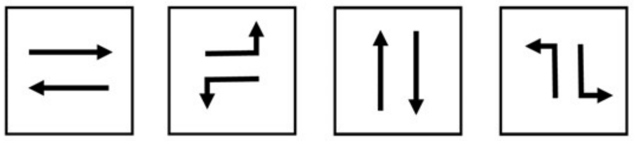


Fig. 10. Multi-intersection action space.

The states of the intersections include the number of vehicles in each lane in the system; for the three intersection network, there exists 60 lanes in the simulation and the prior action that the agent chose.

$$vehclieineachlane = \begin{bmatrix} D_0 \\ D_1 \\ D_5 \end{bmatrix}. \quad (8)$$

$$-DQNprioraction = \begin{bmatrix} A_0 \\ A_1 \\ A_6 \end{bmatrix} \quad (9)$$

$$ADQNprioraction = \begin{bmatrix} A_1 \\ A_1 \\ A_3 \end{bmatrix} \quad (10)$$

2) Multi-intersection DQN's action space: Since the network model has 4 traffic phases in every three intersections, the agent has a total combination of 64 actions to select based on the state it observes. The agent can determine specific traffic light phases in each intersection by constructing a logical table as below.

TABLE I
MULTI-INTERSECTION DQN'S ACTION SPACE LOGIC TABLE

A	0	1	2	3	4	5	7	...	63
T0	0	1	2	3	0	1	2	...	3
T1	0	0	0	0	1	1	1	...	3
T2	0	0	0	0	0	0	0	...	3

3) Multi-intersection IDQN's action space: In IDQN, each agent controls their responsible traffic light signal independently, resulting in each having only 4 traffic phases to select in each action. The agent can determine specific traffic light phases in their responsible intersection by the Q-values they calculated without considering the action taken by other agents.

After determining which action is chosen, the model identifies the responding traffic light phase for each traffic light intersection, then changes the traffic light signal accordingly. The duration of each phase is 10 time-steps, and the yellow phase is ignored in this simulation.

C. Neural network design

1) One-intersection DQN: The agent's neural network design in this network takes the observed states as input, including the number of vehicles in each lane(D) and the last action(A). The agent design contains two layers on neural nodes. The first and second neural layers are fully connected layers of size 64 and 32 neural nodes. The last layer is a linear layer that outputs Q-values corresponding to the 2 possible actions taken by the agent.

2) Multi-intersection DQN: The agent's neural network design in the DQN takes the observed states as input, including the number of vehicles in each lane(D) and the last action(A). The first, second, and third neural layers are fully connected layers of size 512, 256, and 128 neural nodes. The last layer is a linear layer that outputs Q-values corresponding to the 64 possible actions taken by the agent.

3) Multi-intersection IDQN: The agent's neural network design in the IDQN takes the observed states as input in all three agents, including the number of vehicles in each lane(D) and the last action(A). The first, second, and third neural layers are fully connected layers of size 128, 64, and 32 neural nodes. The last layer is a linear layer that outputs Q-values corresponding to the 4 possible actions taken by each agent. The three agents share the same replay buffer to store memories.

D. Training evaluation

To monitor each agents' training, several values were tracked. Table 2 lists the hyper-parameters used in the training process.

1) Deep reinforcement learning (DRL)
2) Python, TraCI and SUMO
3) Deep learning models in Keras (TensorFlow-based)
4) TraCI (TCP socket) real-time connection

TABLE II
REINFORCEMENT LEARNING TRAINING PARAMETERS

Parameter	Values
Epochs	2000
Learning rate	0.001
Exploration rate	1.00
Min Exploration rate	0.001
Target network update	10 time step
Discount factor	0.75
Replay buffer size	500
Minibatch size	32

Two metrics in the training process were selected to evaluate the performance of the agent, total halting vehicle number and total waiting time in each episode. SUMO defines a vehicle hauling as its speed below 0.1 m/s and calculates its waiting time as the time since the last time it was faster than 0.1 m/s.

After the agent training reaches convergence, four metrics were selected to compare its performance against the max-pressure algorithm, including the waiting time, CO2 emissions, and total vehicle number in each lane every 10 time-step. The waiting time was calculated by cumulating each vehicles' total time stopped in the network. This metric was used to evaluate whether the algorithm was able to avoid long waiting times and congestion in the traffic network. CO2 emissions were calculated using a built in SUMO function to further evaluate whether the algorithm was able to reduce greenhouse gas emissions. Total vehicle number was calculated by cumulating every vehicle in the system at the moment, to further track traffic flow changes in the system.

VII. Research Result and Comparison

A. One-intersection DQN

Shown in Figure 8 is the training progress of the DQN agent in the one-intersection environment. The learning process was interrupted several times, creating several spikes at around epochs 40, 50, and 65 due to the limitation of the computation abilities in the early stages of the agent training. Based on the simple network and small action space, the agent remained stable after training for around 70 epochs. It is proven by using both metrics mentioned in the previous section (total halting vehicle number and total waiting time).

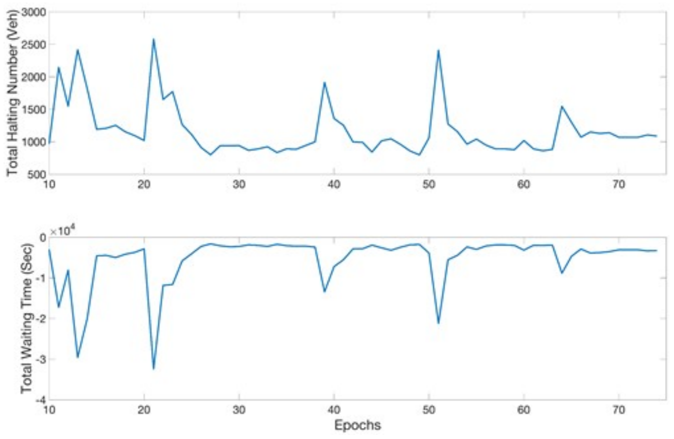


Fig. 11. One-intersection DQN training result.

After finishing the training process, the agent was applied to the same simulation network it was trained in and compared to the MP. As Figures 10A and 10B show, both algorithms had similar performances in the simplified one-intersection simulation, with the DQN having a slightly lower waiting time in both the lane averages and total vehicle number in most simulation time steps. By using the waiting time and the vehicle number metrics, the comparison shows that DQN was able to allow vehicles to wait slightly less than the MP in the simulation. However, both the CO2 emission metric shows very similar results between two algorithms. The reason causing this result could be that reducing just a few seconds of waiting time in the network was not enough to lower the CO2 emissions significantly.

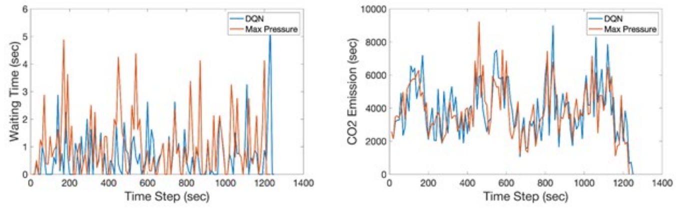


Fig. 12. Comparison between One-intersection DQN and Max Pressure for Per Lane.

B. Multi-intersection DQN

Figure 11 below shows the training progress of the DQN agent in the multi-intersection environment. Because of the

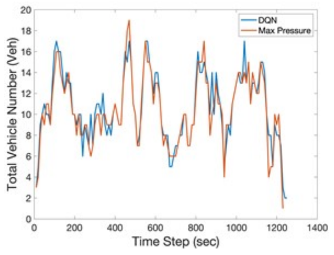


Fig. 13. Comparison between One-intersection DQN and Max Pressure for Total Vehicles.

more complicated network and larger action space, the agent took more epoches to reach its optimal policy, gradually reaching a relatively stable status after 50 epochs. This was because of the larger and more complicated DNN used. Despite training the agents over 100 epochs to ensure optimal performance of the algorithm, the performance in each epoch in the training process is relatively poorer than the other reinforcement learning algorithm in this research. There was no interruption during the training to cause the much larger spikes as seen in the single agent DQN. Using the metrics mentioned in the previous section (total hauling vehicle number and total waiting time) it was determined graphically when the agent reached its optimal solution.

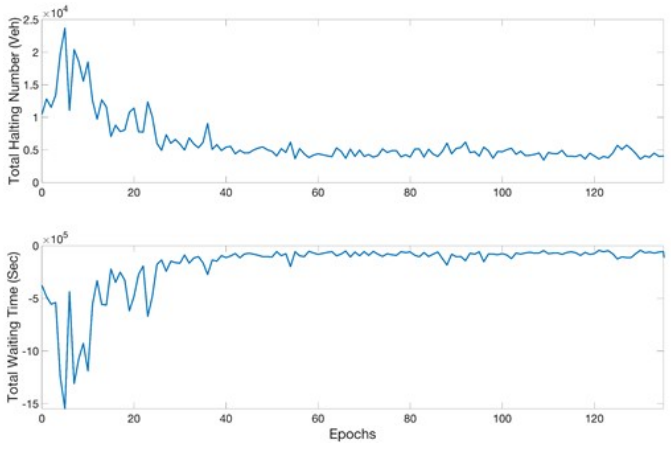


Fig. 14. Multi-intersection DQN training result.

After finishing the training process, the trained agent was applied to the same simulation network it was trained in, and its performance with MP was compared. As Figure 12A and B shows, the MP had better performance than the DQN agent. For the waiting time metric, despite showing a sharp decrease during the middle of the simulation, the overall performance was dramatically worse than MP. The significantly longer waiting time also was reflected in the CO2 emission, causing longer and higher peaks throughout the simulation. From Figure 12B, the DQN performed closer to MP for vehicle number, but was still not able to reach the same performance. The total vehicle number decreased at around step 400 in Figure 12B, which also aligns with the jump in the waiting time metric in Figure 12A. This may indicate the DQN agent was not able to serve the network fast enough. Using the MP, vehicles on average spent 5.826 time steps in the network, whereas with the DQN, vehicles spent 9.158 time steps in the network. Another deficiency of the DQN was that it could not determine the few last vehicles in the end of the simulation, making it longer to then finish the simulation, at around 900 time-step.

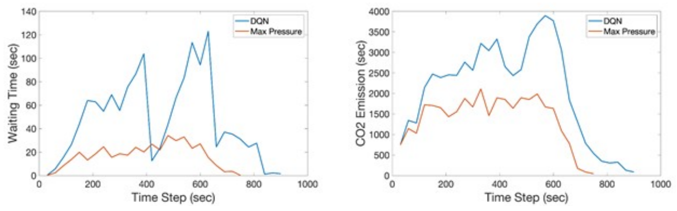


Fig. 15. Comparison between Multi-intersection DQN and max pressure for Per Lane.

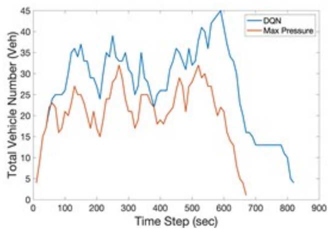


Fig. 16. Comparison between Multi-intersection DQN and max pressure for Total Vehicles Number.

C. *Multi-intersection IDQN*

Upon training the IDQN algorithm using the same reward as the single-agent DQN, convergence did not occur, even after 200 episodes. To solve this issue, a new reward was used. The reward for this agent was calculated by subtracting the present action's cumulative lane vehicle waiting time from the last action's cumulative lane vehicle waiting time. The design was to determine whether the present action relieved congestion in the system. After calculating the reward and before each agent assigns its next action, the algorithm records the reward that the agent receives in this training step along with the current state to the replay buffer. The reward is positive if the agent decreases the cumulated waiting time compared to the last action. The reward stored in the replay buffer was identical for all the agents, meaning that each agent takes memories from others to train itself.

Figure 13 shows the training progress of the IDQN agent in the multi-intersection environment. Based on the same complicated network but smaller action space, the IDQN agent took a shorter training process, gradually reaching a relatively stable status after 20 epochs. The agents were trained for approximately 100 epochs to ensure optimal performance of the algorithm. The performance in each epoch in the training process was more stable than the DQN agent. Judging by the experience with one-intersection agent training, there was no interruption during the training. Using the metrics mentioned in the previous section, total hauling vehicle number and total waiting time, it was determined graphically when the agent reached its optimal solution.

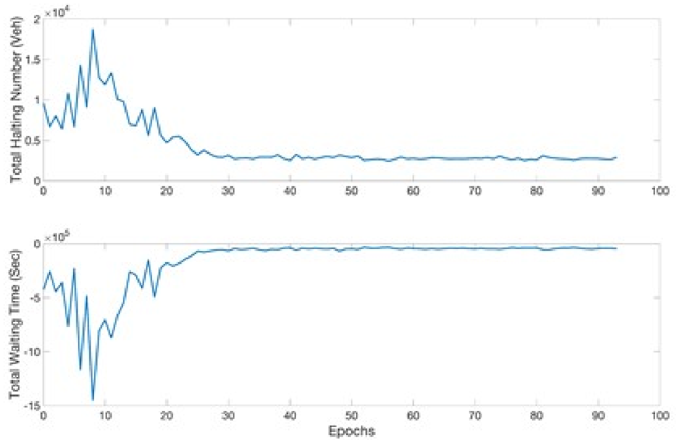


Fig. 17. Multi-intersection IDQN training result.

After finishing the training process, the IDQN was applied to the same simulation network it was trained in and the performance was compared to MP. As Figure 14A shows, the IDQN agent performed significantly better than Max Pressure in both waiting time and CO2 Emission. Results suggest that IDQN was able to allow vehicles to wait less thus, causing the overall CO2 emission to be lower. Also, Figure 14A shows that IDQN both metrics went flat faster than MP, indicating that the controller cleared the system faster than MP. As Figure 14B shows, the IDQN performed slightly better than Max Pressure in total vehicles in every 10 time steps. In general, from both Figure 14A and Figure 14B, IDQN was able to both clear the network faster and prevent congestion better than MP in the three-intersection system.

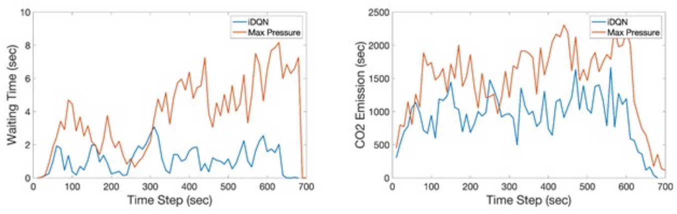


Fig. 18. Comparison between Multi-intersection IDQN and max pressure for Per Lane.

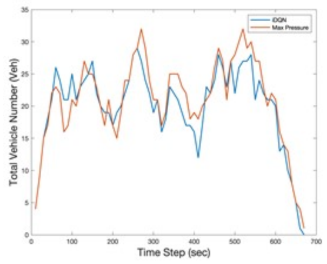


Fig. 19. Comparison between Multi-intersection IDQN and max pressure for Total Vehicles.

Judging by the performance in four of the metrics, DQN was able to solve the traffic congestion problem in the simple, one intersection traffic network, both by lane and by vehicle. In the multi-intersection, however, the DQN performed worse than the Max Pressure algorithm in all of the three variables being measured. IDQN on the other hand, performed better than the Max Pressure algorithm in the three intersections traffic network.

D. MADQN

The average waiting time of vehicles at intersection exits and the average queue length of vehicles are employed as two straightforward parameters for performance comparison. It can be seen that, compared with the traditional Webster method, DQN reduces the average waiting time by 32.5% and the average queue length by 29.4%, whereas MA2C reduces the former by 60.9% and latter by 50.8%.

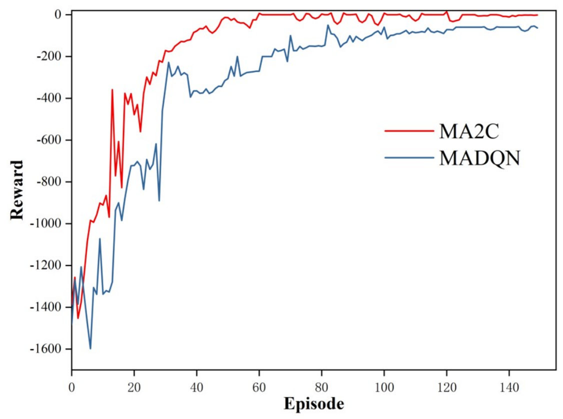


Fig. 20. Rewards.

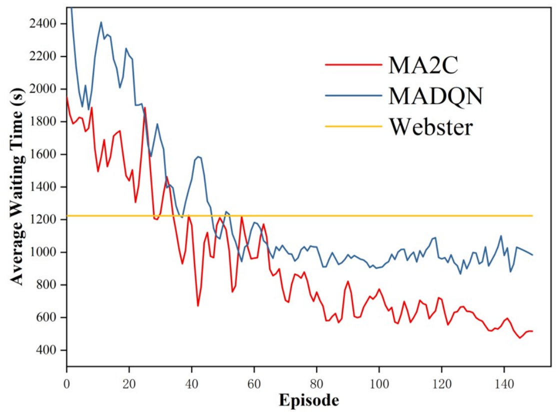


Fig. 21. Average waiting time.

VIII. Conclusion

This capstone project explores the potential application of using reinforcement learning in traffic controllers and compares the results with the well-established traditional algorithm, Max Pressure. The RL methods used were DQN and IDQN and two different traffic networks were tested. Through experimentation, it was shown that the DQN is a suitable algorithm that has the potential to perform as well as Max Pressure

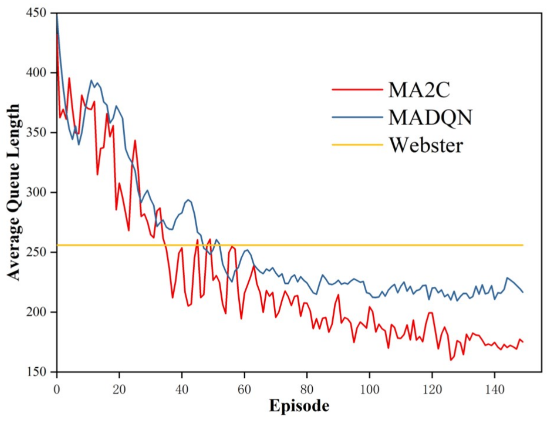


Fig. 22. Average Queue Length.

in a simple one-intersection network system, in regards to waiting time and CO2 emissions. However, when extending the simulation network system from one-intersection to multi-intersection, the DQN was shown incapable of performing as well as Max Pressure. Potential causes were that the states and inputs of the network were too complicated for the agent to process. Extending the DQN to a multi-agent IDQN resulted in the algorithm outperforming Max Pressure, despite the algorithm ignoring most correlation between agents. The results indicate that while dealing with a more complicated network system (even just the three intersections in this paper) a multi-agent algorithm is more suitable then single-agent for dealing with the scale in input state and action space.

Through this research study, it was learned that a variety of methods and techniques exist for traffic signal control, all with their own benefits and drawbacks. The Max Pressure algorithm is a strong, and simple approach, but may not provide truly optimal signal timing due to its myopic approach and fixed refresh rate. With regards to using RL, there are many solutions and optimization techniques to improve the performance of the agent, however evaluating and diagnosing the outcomes is especially challenging, as the agent must be trained for numerous epochs before being evaluated.

In conclusion, although training algorithms may require substantial time to be trained, this project demonstrated the potential advantages to apply reinforcement learning to dynamic traffic signal controllers: less vehicle waiting time, less CO2 emissions, and faster travel times. This suggests the feasibility of utilizing the RL algorithms, especially IDQN, to resolve traffic congestion in urban cities. Future work should include extending the network system to city-wide scale, adding in more traffic conditions, and investigating into other more robust multi-agent reinforcement learning algorithms. In addition, due to the complex nature of neural networks, developing diagnostic codes to supervise the Q-function updating would be beneficial during the training process. Further, a more powerful computation device should be used to improve the learning process and learning speed of the reinforcement learning algorithm.

Acknowledgments

This should be a simple paragraph before the References to thank those individuals and institutions who have supported your work on this article.

References

[1] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente, “Multiagent Cooperation and Competition with Deep Reinforcement Learning,” ArXiv.org, (2015), https://doi.org/10.48550/arXiv.1511.08779.

[2] Elizabeth Hunter, Brian Mac Namee, and John Kelleher, “A Comparison of Agent-Based Models and Equation Based Models for Infectious Disease Epidemiology.” Volume 2259 of CEUR Workshop Proceedings, (2018).

[3] Hao Liu, and Vikash Gayah, “Total-delay-based Max Pressure: A Max Pressure Algorithm Considering Delay Equity,” Transportation Research Record, (2023), https://doi.org/10.1177/03611981221147051

[4] Hiroshi Makino, Kazuya Tamada, Koichi Sakai, and Shunsuke Kamijo, “Solutions for urban traffic issues by ITS technologies”, IATSS Research, Volume 42, Issue 2, (2018), Pages 49-60, ISSN 0386-1112, https://doi.org/10.1016/j.iatssr.2018.05.003.

[5] Hua Wei , Chacha Chen , Guanjie Zheng , Kan Wu , Vikash Gayah , Kai Xu , Zhenhui Li . “PressLight: Learning Max Pressure Control to Co- ordinate Traffic Signals in Arterial Network.” In The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, (2019). https://doi.org/10.1145/3292500.3330949

[6] Jeancarlo Arguello Calvo and Ivana Dusparic, “Heterogeneous Multi-Agent Deep Reinforcement Learning for Traffic Lights Control,” Irish Conference on Artificial Intelligence and Cognitive Science, (2018), Retrieved from Https://www.semanticscholar.org/paper/Heterogeneous-Multi-Agent-Deep-Reinforcement-for-Calvo-Dusparic/ab1ea8e8ca6e3ef75797d3fe6f9364c1d79affaa

[7] Khadijeh Alibabaei, Pedro D. Gaspar, Eduardo Assunção, Saeid Alirezazadeh, Tânia M. Lima, Vasco N. G. J. Soares, and João M. L. P. Caldeira, “Comparison of On-Policy Deep Reinforcement Learning A2C with Off-Policy DQN in Irrigation Optimization: A Case Study at a Site in Portugal,” Computers, (2022), 11(7):104, https://doi.org/10.3390/computers11070104.

[8] Michael W Levin, “Investigating Max-Pressure Traffic Signal Timing,” Crossroads, (2014) Retrieved from https://mntransportationresearch.org/2023/02/14/investigating-max-pressure-traffic-signal-timing/

[9] Mike Wang, “Deep Q-Learning Tutorial: Mindqn.” Medium, Towards Data Science, (2021), https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc.

[10] Orosz Gábor, Wilson R. Eddie and Stépán Gábor, “Traffic jams: dynamics and control”, Phil. Trans. R. Soc. A.3684455–4479,13 (2010), http://doi.org/10.1098/rsta.2010.0205

[11] Patrick Loeber. “Reinforcement Learning with (Deep) Q-Learning Explained,” News, Tutorials, AI Research, (2022), https://www.assemblyai.com/blog/reinforcement-learning-with-deep-q-learning-explained/.

[12] Pravin Varaiya, “Max pressure control of a network of signalized intersections,” Transportation Research Part C: Emerging Technologies, vol. 36, pp. 177-195, (2013), https://doi.org/10.1016/j.trc.2013.08.014.

[13] Richard Sutton and Andrew Barto, “Temporal Difference Learning,” Reinforcement Learning: An Introduction, ed. 2, pp. 119-131, (2018), https://doi.org/10.1109/TNN.1998.712192

[14] Satria A. Ramadhan, Herman Y. Sutarto, Gilang S. Kuswana, and Endra Joelianto “Application of area traffic control using the max-pressure algorithm”, Transportation Planning and Technology, 43:8, 783-802, (2020), Retrieved from https://doi.org/10.1080/03081060.2020.1828934

[15] Shaili Mishra and Anuja Arora, “A Huber Reward function-driven deep reinforcement learning solution for cart-pole balancing problem”, Neural Comput and Applic, (2022), https://doi.org/10.1007/s00521-022-07606-6

[16] Tom Mitchell “Deep Reinforcement Learning – Solving known MDPs.”, (2018) Retrieved from https://www.andrew.cmu.edu/course/10-703/slides/lecture4valuePolicyDP-9-10-2018.pdf

[17] Thanh Thi Nguyen, Ngoc Duy Nguyen and Saeid Nahavandi, ”Deep Reinforcement Learning for Multiagent Systems: A Review of Challenges, Solutions, and Applications,” in IEEE Transactions on Cybernetics, vol. 50, no. 9, pp. 3826-3839, (2020), doi: 10.1109/TCYB.2020.2977374.