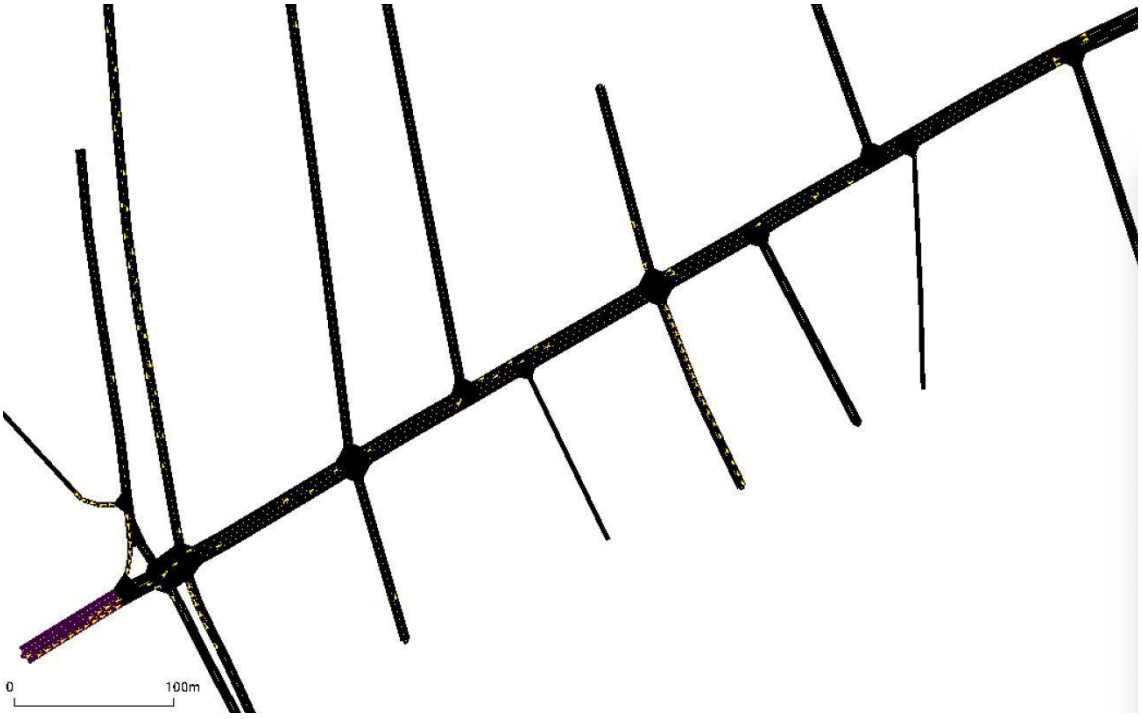
Abstract:
This paper proposes a novel method to reduce vehicles' average waiting time via traffic signal control for a chosen multi-intersection road segment with real-world traffic flow data. The proposed deep reinforcement learning (DRL) based signal timing algorithm is compared against the classical 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.

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.

Tools for simulation:
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

Completed components are detailed as follows:

1. System model

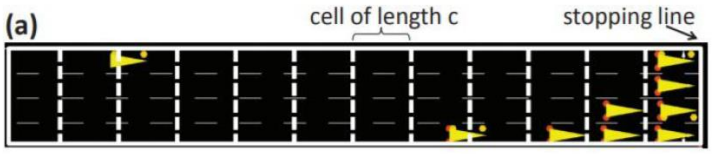


The above figure depicts the multi-intersection road segment adopted in this study. It comprises a two-way four-lane main road, with the right lane for through and right turns and the left lane for through and left turns, and two-way two-lane side roads whose lanes are used for left turns, through and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

2. 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.

3. State



(b)

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

(c)

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

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:

$$P = \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix} \qquad V = \begin{bmatrix} V_0 \\ V_1 \\ V_2 \\ V_3 \end{bmatrix}$$

4. 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.

5. 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.

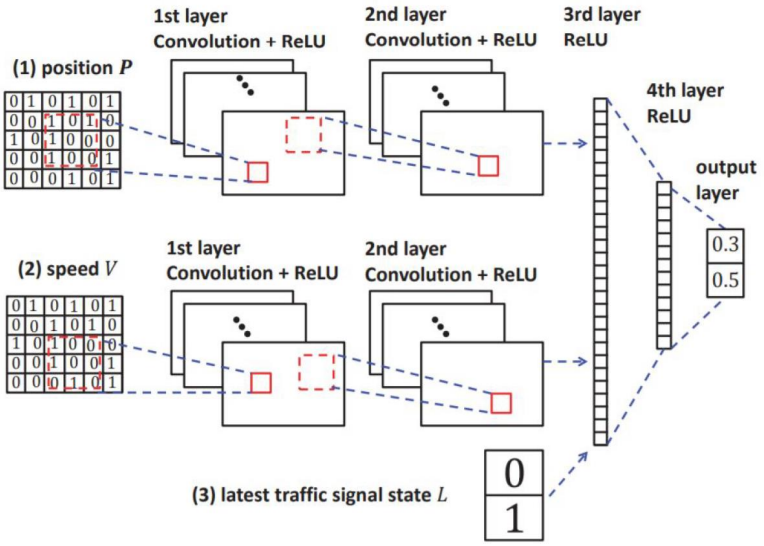
6. Agent objective

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 π. 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^*:

$$
\begin{aligned}
Q_\pi(s,a) &= \mathbb{E}\{R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots | S_t = s, A_t = a, \pi\} \\
&= \mathbb{E}\{\sum_{k=0}^{\infty} \gamma^k R_{t+k} | S_t = s, A_t = a, \pi\}
\end{aligned}
$$

7. 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 the 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.

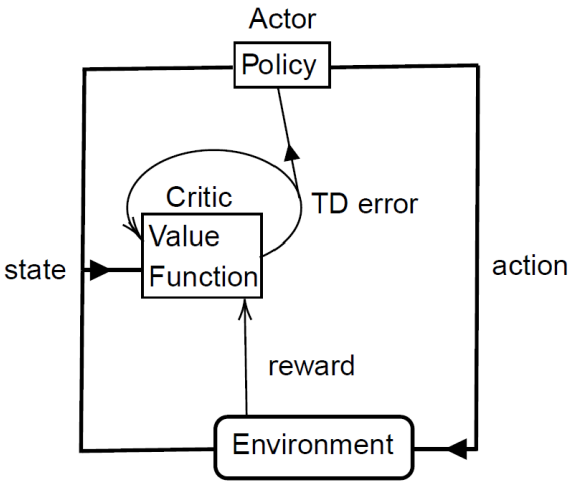
For network training, the neural network is first initialized with random weights. At the beginning of each time step, the agent observes the current time step St, forms the input to the neural network, and performs the action At with the highest cumulative future reward. Upon performing an action, the 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 θ as the DNN network is trained to minimize the mean square error (MSE), given by:

$$MSE(\theta) = \frac{1}{m}\sum_{t=1}^{m}\Big\{\big(R_t + \gamma \max_{a'} Q(S_{t+1}, a'; \theta')\big) - Q(S_t, A_t; \theta)\Big\}^2$$

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

8. A2C algorithm

The Actor-Critic algorithm (A2C) is a Reinforcement Learning agent that combines value optimization and policy optimization approaches (Value Based and Policy Based). More specifically, the A2C combines the Q-learning and Policy Gradient algorithms. The resulting algorithm obtained at the high level involves a cycle that shares features between Actor and critics.



The output of the actor is the probability of executing each action from state s_t at time t. The second model is the critic, which estimates the value function V and evaluates all actions performed by the actor.

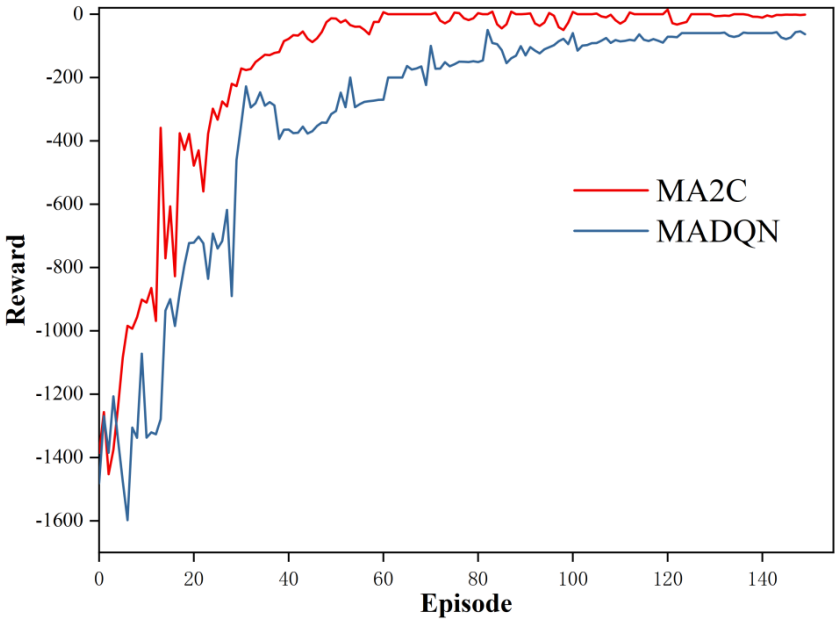
The differences between DQN and A2C are:

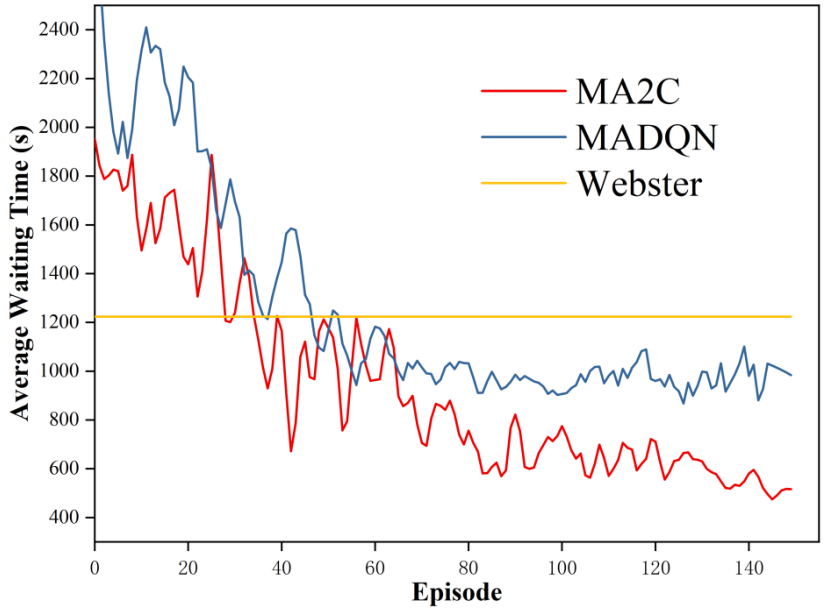
- The DQN model is an off-policy method, and the A2C model is an on-policy method.
- Unlike DQN, A2C does not use the Replay Buffer but learns the model using state(s), action(a), reward(r), and next state(s') obtained at each step.
- In DQN, the function Q is estimated; however, in A2C, the value function V and policy p are estimated.

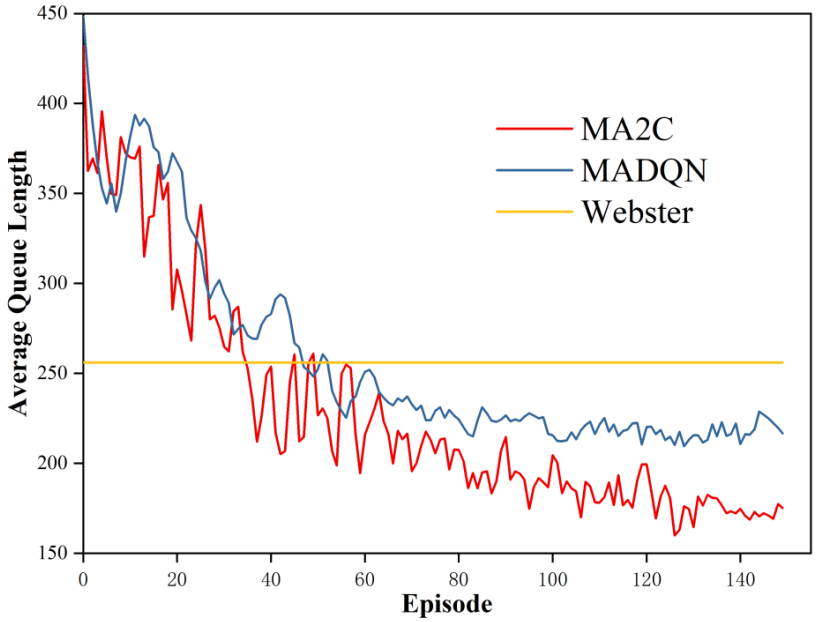
(Citation: Alibabaei, K.; Gaspar, P.D.;Assunção, E.; Alirezazadeh, S.;Lima, T.M.; oares, V.N.G.J.;Caldeira, J.M.L.P. Comparison of On-Policy Deep Reinforcement Learning A2C with Off-Policy DQN in Irrigation Optimization: A Case Study at a Site in Portugal. Computers 2022, 11, 104. https://doi.org/10.3390/computers11070104)

9. Experimental results

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.


Reward
Episode
MA2C
MADQN
0
-200
-400
-600
-800
-1000
-1200
-1400
-1600
0
20
40
60
80
100
120
140


Average Waiting Time (s)
Episode
MA2C
MADQN
Webster
2400
2200
2000
1800
1600
1400
1200
1000
800
600
400
0
20
40
60
80
100
120
140



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%.