Deep reinforcement learning using neural network configuration for traffic light signal control

Youcef Hassani*, Bilal Tolbi, Hicham Zatla*****

**LECM laboratory, University of Sidi Bel Abbes, POBox 89, 22000, Sidi Bel Abbes Algeria (youmvs98@gmail.com)*
***LECM laboratory, University of Sidi Bel Abbes, POBox 89, 22000, Sidi Bel Abbes Algeria (bilalcdsi@gmail.com)*
****IRECOM laboratory, University of Sidi Bel Abbes, POBox 89, 22000, Sidi Bel Abbes Algeria (hicham.zatla@gmail.com)*

Abstract: Traffic congestion in Algerian cities poses a major challenge to urban mobility. It increases waiting times, energy consumption, and pollution emissions. Deep Reinforcement Learning (DRL) offers promising solutions for intelligent traffic management. In this paper, a novel intelligent sequential methodology, called Self-Reaction Deep Q Network (SR_DQN) is introduced. This approach takes into account critical traffic states, such as tram arrivals, numbers of vehicles and random pedestrians. It also reduces the influence of irrelevant characteristics. Real-world experiments were conducted on several traffic scenarios. The results show that SR_DQN adapts effectively to different traffic profiles. The method reduces waiting times and queue lengths. It also improves average vehicle speeds and decreases CO_2 emissions. The performance of SR_DQN surpasses that of the Deep Q Network (DQN), Double Deep Q Network (DDQN), Gated Recurrent Unit Deep Q Networks (GRU DQN) and GRU DQN with dueling output methods. This study confirms the potential of DRL for the development of intelligent, efficient and sustainable urban transport systems.

Keywords: Neural network configuration; Transportation systems; Single agent; Deep reinforcement learning; Optimization; Timing traffic lights

1. INTRODUCTION

Population growth in Algerian cities has led to a significant increase in the number of vehicles. This development has exacerbated urban traffic congestion, causing long waiting times and widespread user dissatisfaction. Traffic lights generally operate according to fixed cycles, often ill-suited to actual traffic conditions. Congestion occurs when demand exceeds the capacity of the road network, particularly during peak hours. Traditionally, traffic management relied on the intervention of traffic officers. While this approach contributes to road safety, it presents operational limitations and human risks (Chu et al., 2019).

Machine learning now offers promising solutions for optimizing urban traffic management. Among its branches, reinforcement learning (RL) allows an agent to learn through interaction with its environment. Recent advances in Deep Reinforcement Learning (DRL) have demonstrated their effectiveness in several traffic control problems (Liu et al., 2023). Furthermore, a comparative study of DRL algorithms showed that value-based methods offer better performance for traffic light control (Wu et al., 2023). These advances pave the way for intelligent, adaptive, and sustainable transportation systems capable of reducing congestion and improving urban mobility in Traffic Signal Control (TSC) experiment simulations.

P. Wang and W. Ni (Wang and Ni, 2024) aimed to improve Dueling Double Deep Q-Network (Z. Wang et al., 2016) by using Convolutional Block Attention Module (CBAM) (Woo et al., 2018) for Traffic Signal Optimization, and the optimization algorithm employs the Double Deep Q Network (Double DQN) (H. van Hasselt et al., 2015) called CBAM_D3QN. Or for Fixed Timing Strategy metrics as a Maximum Pressure algorithm (Varaiya, 2013). Yanliu Zheng et al. (Zheng et al., 2025) proposed a method called Pri-DDQN, which presents a priority-based dynamic experience replay mechanism to enhance the sampling rate of important samples with double DQN. Bao, J., Wu, C., Lin, Y. et al. (Bao et al., 2023) have introduced a method based on Federated Learning (FL) with reinforcement learning (RL) for TSC. FL is effectively achieved by integrating the parameters of local RL models into a cloud federated model, passing intersection variations with a integrate agent state information. Wang, B. et al. (Wang et al., 2022) propose the EP-D3QN algorithm based on MP, Self-organizing traffic lights (SOTL), and 3DQN algorithms (Liang et al., 2019).

In other work, Changjian Cai & Min Wei (Cai and Wei, 2024) present a new method called PN_D3QN, which combine a comprehensive strategy involving dueling network, noisy network (Shuai Han et al., 2022), prioritized experience replay (Schaul et al., 2016), and double q-learning. Swapno et al. (Swapno et al., 2024) developed a sophisticated Deep Q-Network (DQN) (Mnih et al., 2013) and smoothly integrated it into the system intersection, and the result shows that the method proposed reduced queue lengths by 49% in traffic management. C.-J. Choe et al. (Choe et al., 2018) propose a method called DRQN-TLC, which presents deep recurrent Q networks based on the traffic light signal. The method presents the combination of LSTM (Long-Short Term Memory) with DQN, and the results show

that DRQN-TLC reduces the average traveling time by 23% and the overall vehicle waiting time by 10% when compared to the DQN-based method.

In this paper, a method with two major improvements is proposed. The first concerns the neural network architecture, where a new architecture called a Self-Reaction Network (SRN) is introduced, whereas most methods use a standard neural network as an example (Pri-DDQN (Zheng et al., 2025), FL-base RL (Bao et al., 2023), EP-D3QN (Wang et al., 2022), or simply a combination of existing techniques such as CBAM D3QN (Wang and Ni, 2024) and PN_D3QN (Cai and Wei, 2024). In our modified architecture, a new mathematical equation (eq. 6) is introduced with an improvement to the focus gate equation (eq. 6) and the use of the GRU concept (Cho et al., 2014) before the output layer for more simplification; this modification offered us significantly faster calculations compared to LSTM (Hochreiter and Schmidhuber, 1997), and greater stability in terms of result display compared to all the previously mentioned techniques. The second part concerns the improvement of the agent training where Dueling DQN is used in our own optimization algorithm, to update weight parameters. These contributions enhance the performance and stability in reinforcement learning (RL) tasks. Experimental results demonstrate performance improvements with different metrics and scenarios."

The second section of this paper presents the methodology process and introduces the new neural configuration. The subsequent section consists of giving a detailed description of each intersection scenario, independently, of the real environment in Aidi Bel Abbes city. Afterwards, in the fourth part, several experiments were performed using the proposed method (SR_DQN), and the most successful techniques are compared, namely the experimental method, fixed by the municipal authorities of Sidi Bel-Abbes city, Deep Q-Network (DQN), Double DQN, GRU DQN, GRU DQN with dueling output and the proposed SR_DQN approach. In the final section, a conclusion is presented to summarize the main points and results of the study.

2. METHODOLOGY

A discrete action space is employed in this research for the purpose of selecting the phase and duration of traffic lights, using deep Q-networks (DQNs) (Youcef et al., 2024). These networks are widely used in deep reinforcement learning frameworks.

Regarding the DRL algorithms, the reward represents the evaluative feedback that is given to an agent in accordance with the action it performs in a specific environmental state, while a value function expresses the long-term advantages (Sutton, R.S. & Barto, A.G., 2018). The same metrics are utilized to define the reward function r_t when addressing urban traffic scenarios. This function (Kodama et al., 2022) is given as:

$$r_t = W_{t-1}(J) - W_t(J) \quad (1)$$

Here $W_t(J)$ represents the waiting time of all vehicles (cars, buses, motorcycles) at time t for intersection (J).

In real-world urban traffic scenarios, each traffic light cycle adds to the overall waiting time (Youcef et al., 2024), and intersections experience significant congestion likely giving rise to longer waiting periods.

The primary purpose of the study consists in finding the best steps in the synchronization cycle by determining the optimal minimum waiting time through simulation steps or real-time scenarios.

It should be emphasized that the agent primarily focuses on mitigating congestion until optimal conditions are attained.

The sequence method, which was applied in our previous work (Youcef et al., 2024), was also used in the experimentation, where the GRU concept (Cho et al., 2014) is integrated, its fundamental composition is characterized by two distinct types of gates, namely update gates and reset gates. The inspired formulations of this work are given as follows:

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr}) \quad (2)$$

$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz}) \quad (3)$$

$$\tilde{h}_t = tanh(W_{in}x_t + b_{in} + W_{hn}(r_t \odot h_{(t-1)}) + b_{hn}) \quad (4)$$

$$h_t = (1 - z_t) \odot h_{(t-1)} + z_t \odot \tilde{h}_t \quad (5)$$

With:
σ: Sigmoid function,
tanh: hyperbolic tangent,
r_t: reset gate,
z_t: Update gate,
$\tilde{h}_t$: Candidate cell,
$h_{(t-1)}$: Previous hidden state,
h_t: Hidden cell
x_t: Input at step time *t*,
W_{iz}, W_{hz}: Weights for the update gate.
b_{iz}, b_{hz}: biases for the update gate.
W_{ir}, W_{hr}: Weights for the reset gate.
b_{ir}, b_{hr}: biases for the reset gate.
W_{in}, W_{hn}: Weights for the update gate.
b_{in} , b_{hn}: biases for the candidate hidden state.

It should be noted that not all GRU weight parameters were used for backpropagation in the previous article, nor were they updated at each step. To this end, all weights using a standard normal distribution are randomly selected, and then set the biases to zero; basing only on the weight values. To avoid instability, it was deemed necessary to multiply all weights of the reset gate, update gate, and candidate gate by $1/\sqrt{hidden\ size}$ (hidden size is equal to (*1/N*) size of the fully connected gate). In this work, all the weights and biases of the update gates, reset gates, and modification candidate gates are updated during backpropagation for each training batch. In addition, the optimized weights are directly used without applying any scaling or multiplication.

In the previous method (Youcef et al., 2024), a compressed value with $softsign(W_{if}x_t + b_{if})$ have been chosen, however, in this paper, a new gate called focus gate is introduced, the formula is as follows:

$$\text{Focus gate} = softsign(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}) \quad (6)$$

Where:
x_t: Input at step time *t*.
W_{if}, W_{hf} : Weights for the Focus gate.
b_{if}, b_{hf} : biases for the Focus gate.

$h_{(t-1)}$ represents the hidden previous state of the sequence method, *Softsign* is identified as a continuous and differentiable function, whose values range from -1 to 1. The mathematical representation of *Softsign* is given as follow:

$$softsign(x) = \frac{x}{1+|x|};\ \forall\, x \in R \quad (7)$$

Where x presents the input of *softsign* function.

The focus gate determines how much importance the model should give to the new input based on what it remembers from the past *h(t-1)*. The *softsign* function manages this variability by providing smooth gradients and avoiding saturation issues that are quite common with other activation functions. The reaction equation is inspired from our previous work (Youcef et al., 2024) by replacing the compressed value by focus gate

$$reaction = \frac{h_t}{\delta + log(2+Focus\ gate)} \quad (8)$$

Where h_t presents the Hidden cell from the sequence method, the term *log(2+Focus gate)* adds non-linear scaling**.** The *log* increases slowly, which helps to prevent overreaction, and the model then becomes more reflective and stable. Note that δ represents a constant value, it's set in this work, as a default value, equal to 1, this value helps to prevent the term *log (2+Focus gate)* from taking the value 0.

The GRU is integrated over other RNN methods, because it offers a simplified design with rest gate, update gate and candidate gate parameters with potentially faster computation than LSTM. Also, it provides powerful sequential modeling capabilities with relatively low computational overhead; however it can sometimes lead to unpredictability in specific operational environments. The focus gate is integrated to get stable resulting values from reaction equation.

After introducing the new equation, the modified distinct architecture (see Figure 1) is denoted as the Self-Reaction Network (SRN).

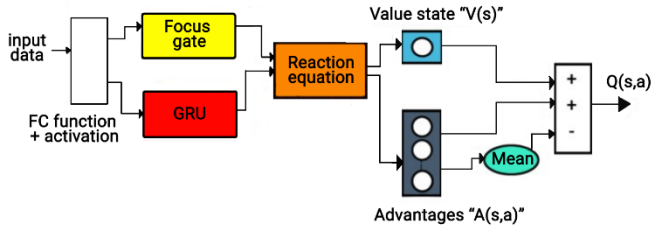


Fig. 1. the proposed neural networks of our SRN approach.

The network architecture was simplified into two hidden layers, where the first use fully connected with the activation function ReLU (Rectified Linear Unit) and second use the new equation." Regarding the new Neural Network (NN), inspiration was deducted from *V(s)* as an assessment of the value state and the advantages of *A(s, a)* from 2DQN (Cho et al., 2014). Then, the action quality value at the considered state is given as:

$$Q(s,a) = V(s) + A(s,a) - \frac{1}{Z}\sum_{i=1}^{Z} A(s,a_i) \quad (9)$$

Here *Z* represents the number of all possible actions.

The Value function *V(s)* represents the overall value of vbeing in a state *s* without taking into account any specific actions. As for the advantage function *A (s, a)*, it introduces the action dimension which indicates to what degree an action is better than all other possible actions in that same state. Furthermore, *V(s)* can change its perspective in accordance with the chosen formulation (new equation). This change enhances the expressiveness and accuracy of representing a state and enables the network to capture more nuanced and complex traffic behaviors. In addition, the resulting values of *A(s, a)* is now derived from the new equation. When the new equation is employed, the model reflects, in a better way, the variations and dynamics of complex environments, enabling more informed decision-making. In addition, once the advantage values *A(s, a)* are fixed, using this new equation.

The simplification of neural networks helps to reduce the network complexity, ensure the consistency of the network architecture and minimize the computation time.

In the field of RL algorithms, a fusion between the deep Q network (DQN) and the SRN architecture was established, resulting in the SR_DQN approach and the loss function is set with mean square error MSE (Rashidi HH, et al., 2023). The full algorithm of the Self-Reaction Deep Q-network (SR_DQN) is given:

Self-Reaction Deep Q-network Algorithm

Create online network and target network with the architecture SRN
Initialize weights.
Initialize replay memory.
Set the discount factor and learning rate.
For episode = 1,2,……episodes:
  Initialize state, hidden
  For step = 1,2……steps:
    Do new state, action a, reward r_i
    Store (state, new state, action, reward, done)
    Sample (state, new state, action, reward, done)
    If done:
      $\hat{y}_i = r_i$
    Else:
      $\hat{y}_i = r_i$ + discount factor.max (Q_{target}(new state, a))
    Calculate the loss function between ($\hat{y}_i$ and Q_{online}(state, a))
    Do Adam optimizer
    Every C steps update target network
    Update hidden
    State = new state
  End for
End for

3. EXPERIMENTAL ENVIRONMENT

The present article aims to study the traffic congestion level of vehicles in Sidi Bel-Abbès, a city located in the northwestern region of Algeria. Over the last years, its population has significantly increased, which has generated high traffic congestion levels in some specific locations of this city. For this, an experimental investigation was conducted in one of the most congested places of the city, namely Abane Ramdane Boulevard (see Figure 2).

Fig. 2. The selected map of the city of Sidi Bel-Abbès (Abane Ramdane Boulevard).

Two software packages were used during our experiments. The first, SUMO (Simulation of Urban Mobility) (Lopez et al., 2018), was used to generate urban traffic maps, including intersections, lanes, traffic lights and vehicle simulation, while the second, Python, was utilized in implementing all deep reinforcement learning algorithms. Afterwards, the "traci" package was installed, by running the "pip install traci" command in the Windows command prompt, in order to establish a connection between the two software packages SUMO and Python. The SUMO simulation package includes a graphical program called netedit which provides tools for creating elements in the virtual environment. In our experimentation, the selected Google map was converted into a SUMO map (see Figure 3) using SUMO OpenStreetMap.

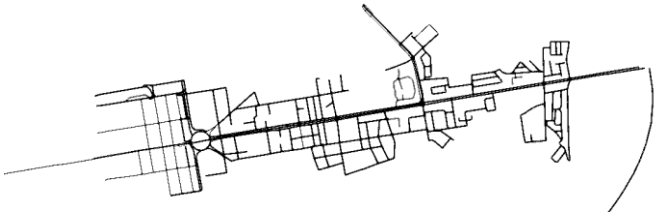


Fig. 3. The converted map of Abane Ramdane Boulevard using OpenStreetMap.

Then, four intersections were selected (see Figure 4), as explained below:
J1: Intersection of Beghdadi Benaissa Street and Abane Ramdane Boulevard
J2: Intersection of Chaib Abdelkader Street and Abane Ramdane Boulevard
J3: Intersection of Naimi Ahmed Street and Abane Ramdane Boulevard
J4: Intersection of Lakhmes Ahmed Street and Abane Ramdane Boulevard

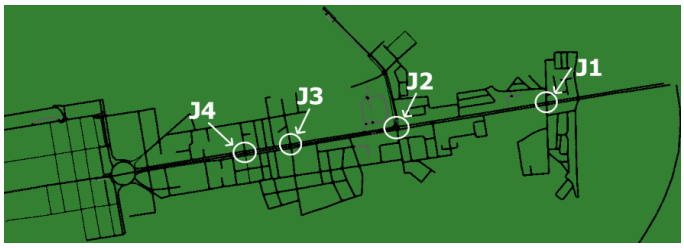


Fig. 4. The selected intersections from Abane Ramdane Boulevard map.

In this work, a map was created using OpenStreetMap and SUMO. This map was then modified and fixed in a way to match the same real intersection, with real distances, number of lanes, etc. As the information about all intersections, i.e. J1, J2, J3, and J4, becomes available, it was decided to explore each intersection independently, without considering the other intersections. The purpose was to avoid any possible problems with the number of vehicles in order to have the real waiting time values of vehicles, and to obtain the most suitable values of the timing phases.

Each vehicle is carefully recorded at these intersections. In addition, a specific distance is assigned to each lane, as indicated in Appendix 1.

A. Setting the number of tramways

It is to be noted that 24 trams travel through the intersection every hour (12 from Amir Abdelkader Station to Idaa Station and the other 12 in the opposite direction, from Idaa Station to Amir Abdelkader Station).

B. vehicle speed settings

In the Python program, a maximum allowed speed of 30 Km/h on the lanes that allow only vehicles (buses, cars, and motorcycles) is chosen for each junction.

In the experiment conducted in this work, a discrete action space for the selection of phases and durations of traffic lights is used. The real environmental phases are presented in Appendix 2.

For the purpose of covering all spatial actions, it was deemed appropriate to consider the following framework for each junction as:

Select action = [[duration 5s, phase 1], [duration 5s, phase 2], [duration 10s, phase 1] ... [duration 20s, phase 2]]

The total number of potential actions is therefore estimated at 8. The letter G is used for the default green phase type in SUMO traci, which allows vehicles to pass through the intersection.

Note that a yellow signal is required between two phases to ensure safety, which means that all vehicles ought to stop before the signal turns red. In our experiment on deep reinforcement learning, the yellow signal is activated when the agent decides to move to another phase for the next step. The following agent for yellow activation is explained in the following algorithm.

The Algorithm for yellow activation:

```
for step in steps:
    if selected phase of step==  selected phase of previous step:
       Don't activate the yellow signal
    else
       Activate the yellow signal
```

In reality, the time of yellow traffic lights is set to 4 seconds. Since durations of 5, 10, 15, 20 seconds, etc., can be selected for each step in our experimental method, then the time can be easily defined. This mainly depends on the presence or absence of the tramway. This time is defined according to the loop given below. This experimental method has been verified with tramways and cars.

Experimental method used with tramways and other vehicles

Tram arrival detection.
If the tram is present and green is <= 20:
Continue until the maximum phase, green = t1, then switch to yellow, then move to the tram time at least 8s.

If the tram is present and green is > 20:
Stop the green phase, switch to yellow, then move to the tram time, at least 8s.

If the tram has not arrived:
Continue until the maximum phase, green = t1, then switch to yellow only.

A series of 0 to 100 episodes, consisting of 0 to 1000 steps for the first and second scenario, 0 to 3600 steps for the third scenario, was employed for each single agent studied in this study. Then, a target network update frequency of 1000 was utilized to make the learning process more stable. In addition, in order to fully leverage an epsilon-greedy algorithm, a decision-making approach that contributes to improving our results, it is highly recommended to follow the next two-step process. First, a high value is assigned to epsilon to make the thorough exploration of the environment by the agent easy before the commencement of epsilon decay. Afterwards, a minimum value of epsilon is defined (Youcef et al., 2024), so that from this value the decomposition process stops in order to prevent the agent from succumbing to excessive greed, such as:

$$\text{epsilon} = \max\left(\text{epsilon}_{\min}, \left(\text{epsilon}_{\max} - \text{epsilon}_{\text{decay}} \cdot \text{t}\right)\right) \quad (10)$$

The prescribed hyperparameters are detailed in Table 1:

Table 1. Implementation parameters.

Parameter	Value
Max memory size	50000
Batch size	64
Optimizer	Adam (Kingma, D.P., Ba, J., 2017)
Target network update frequency	1000
$\text{Epsilon}_{\max}$	1
$\text{Epsilon}_{\min}$	0.01
$\text{Epsilon}_{\text{decay}}$	0.02
Learning rate	0.0001
Discount factor	0.95

4. SIMULATION OF METHODS AND COMPARISON

To evaluation of the proposed method effectiveness for TSC in multiple intersections requires a comprehensive set of performance or evaluation metrics. In this work, the performance of the following different methods has been tested and evaluated, and the neural network architectures for each DRL methods were implemented, trained, and tested using **PyTorch**. All methods were conducted and performed on an **Intel(R) Core(TM) Ultra 7 155H** CPU with 16 GB RAM:

A. Experimental Method

A traditional reinforcement-free strategy is employed in this part of the study in order to analyze the impact of the time cycle on the operational aspects of traffic signals at each junction.

The signal sequence consists of alternating the green phases for Phase 1 and Phase 2. Each green phase is followed by a 4 second yellow phase.

The sequence follows this pattern for each step:
Phase 1:
Green → Yellow (4s) → Phase 2: Green → Yellow (4s)**.**

For intersection J1, the time duration of phase 1 is t1=18s while for phase 2, it is t2 = 24s.

For intersections J2, J3, and J4, the parameters are set for maximum timing, and the time durations of phase 1 and phase 2 may be variable, as was the case of the experimental method where cars and tramways were considered.

For intersection J2, the maximum time duration for phase 1 is t1: 35s, while for phase 2, it is t2: 30s. The time duration can change depending on the presence of the tramway at the intersection, as in the experimental method.

For intersection J3, the maximum timing for phase 1 is t1: 36s, while for phase 2 it is t2: 46s.

For intersection J4, the maximum time duration for phase 1 is t1: 20s, while for phase 2, it is t2: 39s.

The execution of this conventional methodology gave a consistent total waiting time, indicating a consistent use of the environment, with identical vehicles and identical tramways, and using the same traffic light strategies in each episode.

B. DQN method

This technique combines Q-learning with neural networks. The neural network component has a simple design with two hidden layers with ReLU activation functions. The first hidden layer contains 256 fully connected nodes, and the second one contains 256 fully connected nodes as well.

C. Double DQN Method

This method (DDQN) (Hasselt, et al., 2015) uses two crucial elements: firstly, the online network selects the optimal action for the next state, based on the action with the highest Q value. Secondly, the target network calculates the target Q value for the execution of this action in the next state. The neural network architecture follows the methodology used in the previous section with DQN.

D. GRU DQN method

GRU DQN presents a combination of GRU and DQN (Figure 5). The neural network component has a simple design with two hidden-layer functions. The first hidden layer contains

256 fully connected nodes with the ReLU activation function, and the second hidden layer uses GRU with 256 nodes.

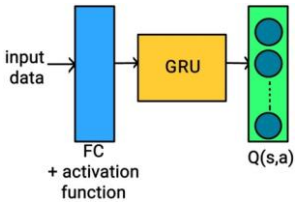


Fig. 5.the architecture neural network used for experimentation GRU DQN.

E. GRU DQN with dueling output

In this combination, the first hidden layer contains 256 fully connected nodes with ReLU activation, while the second includes 256 fully connected nodes (GRU), the result feeds a dueling architecture divided into two separate linear heads: one estimates the state value function *V(s)*, while the other calculates the advantage of each action *A(s,a)*. The final values *Q(s,a)* are computed using an inspired equation from dueling DQN method.

F. SR_DQN method

Our novel approach, i.e. SR_DQN, is introduced in this part of the study. This technique integrates a specific DQN algorithm with the innovative SRN architecture. It is worth mentioning that, in the SRN framework, the dimensions of the hidden layers are set as follows: 256 neurons for the first layer and 256 neurons for the second layer (GRU (Cho et al., 2014) + focus gate). In addition, the common default value σ is also taken as equal to 1.

G. Comparison study

Here, the experimental method used by the municipal authorities of the city of Sidi Bel-Abbès is compared with the DQN, Double DQN, GRU DQN, GRU DQN with dueling output and SR_DQN methods in terms of total waiting time (Figures 6-17); minimum waiting time, vehicle average speed, average queue length, and emission Co2 for each intersection (Table 2, 3, 4 and 5) respectively. A statistical validation for multi scenarios is included:

Scenario 1. In the first scenario, 70 cars were designated for each traffic direction per lane in edge for J1, J2 and J4 and 45 cars for J3 based on situation of intersection from 0 to 1000 seconds of simulation time. The results of total waiting time per episode are presented in figures (6-9) bellow:

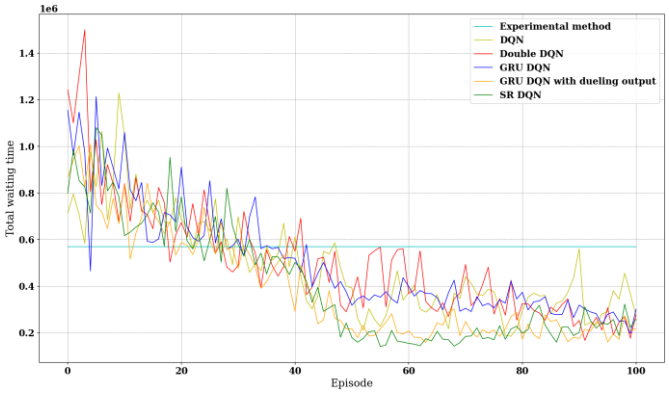


Fig. 6. Comparison of total waiting time per episode using the experimental, DQN, Double DQN, GRU DQN, GRU DQN with dueling output and SR_DQN methods at J1 over scenario 1.

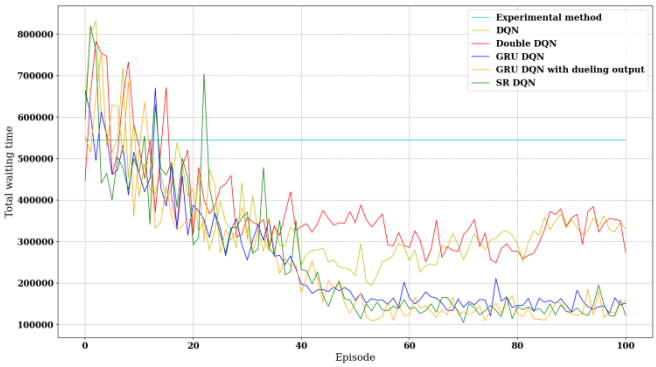


Fig. 7. Comparison of total waiting time per episode using the experimental, DQN, Double DQN, GRU DQN, GRU DQN with dueling output and SR_DQN methods at J2 over scenario 1.

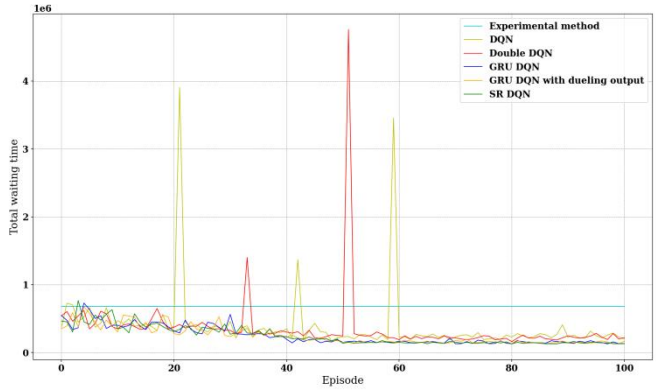


Fig. 8. Comparison of total waiting time per episode using the experimental, DQN, Double DQN, GRU DQN, GRU DQN with dueling output and SR_DQN methods at J3 over scenario 1.

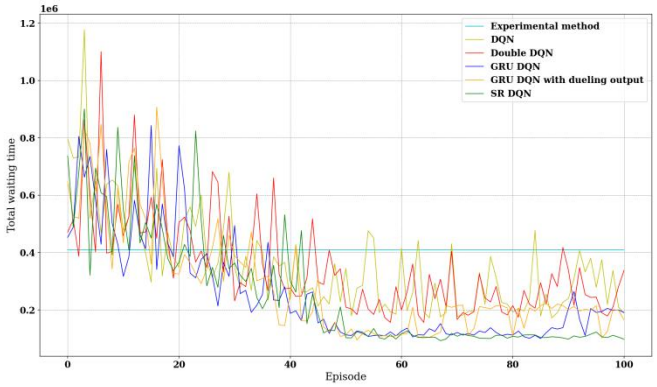


Fig. 9. Comparison of total waiting time per episode using the experimental, DQN, Double DQN, GRU DQN, GRU DQN with dueling output and SR_DQN methods at J4 over scenario 1.

Scenario 2. in this scenario, 50 cars are used for each lane direction in edge for different intersection and randomtrip.py (Guastella and Bontempi, 2023) is used for motorcycles, with: python randomTrips.py -n map1.net.xml -r routes_motos.rou.xml --prefix moto_ -e 1000 --period 25 --vehicle-class motorcycle. This process generated motorcycle-class and vehicles at random locations with a period of 25 seconds during a total simulation time of 1000 seconds. The results of total waiting time per episode are presented in figures (10-13) bellow:

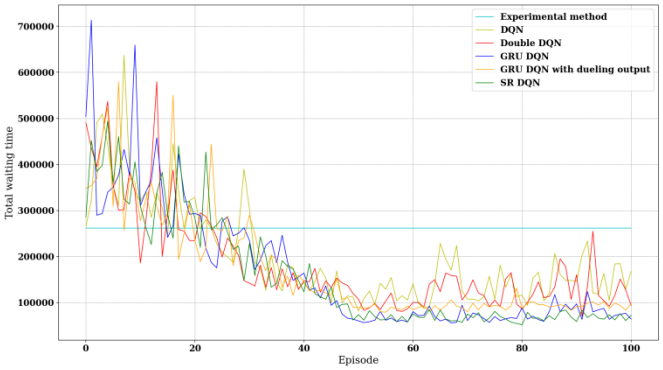


Fig. 10. Comparison of total waiting time per episode using the experimental, DQN, Double DQN, GRU DQN, GRU DQN with dueling output and SR_DQN methods at J1 over scenario 2.

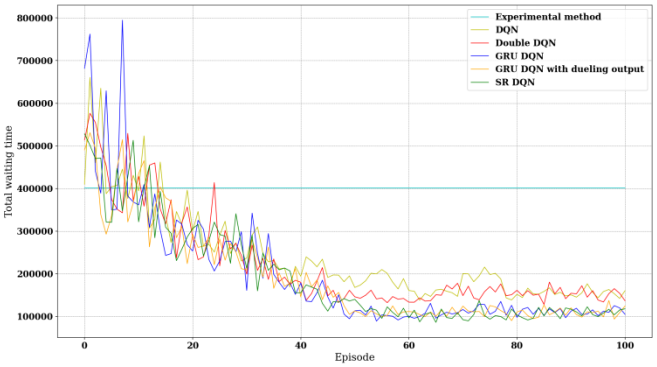


Fig. 11. Comparison of total waiting time per episode using the experimental, DQN, Double DQN, GRU DQN, GRU.

DQN with dueling output and SR_DQN methods at J2 over scenario 2.

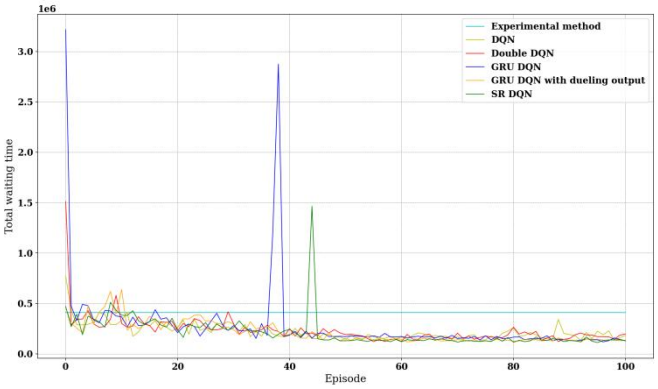


Fig. 12. Comparison of total waiting time per episode using the experimental, DQN, Double DQN, GRU DQN, GRU DQN with dueling output and SR_DQN methods at J3 over scenario 2.

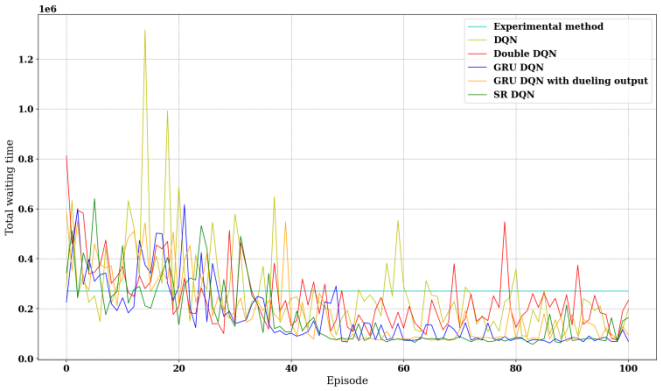


Fig. 13. Comparison of total waiting time per episode using the experimental, DQN, Double DQN, GRU DQN, GRU DQN with dueling output and SR_DQN methods at J4 over scenario 2.

Scenario 3. For the third scenario, the real vehicle data (cars, motorcycles, and buses) are collected on both sides of the road during a one-hour observation of each intersection, these data are presented in detail in Appendix 3(A, B, and C). The results of total waiting time per episode are presented in figures (14-17) bellow:

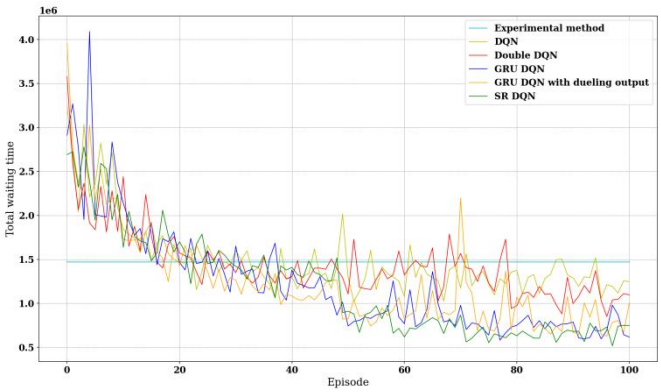


Fig. 14. Comparison of total waiting time per episode using the experimental, DQN, Double DQN, GRU DQN, GRU

DQN with dueling output and SR_DQN methods at J1 over scenario 3.

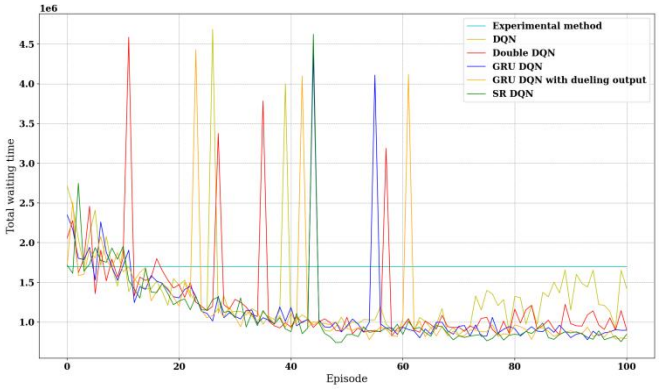


Fig. 15. Comparison of total waiting time per episode using the experimental, DQN, Double DQN, GRU DQN, GRU DQN with dueling output and SR_DQN methods at J2 over scenario 3.

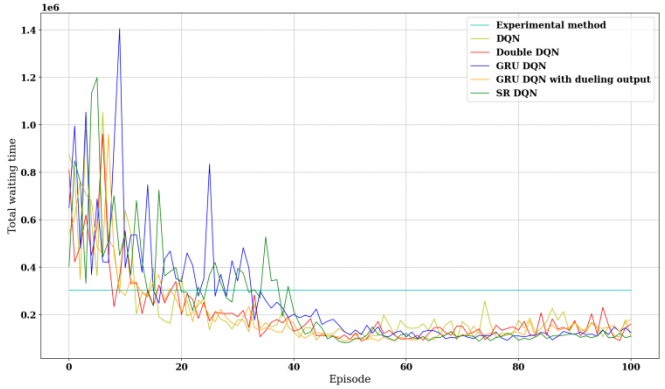


Fig. 16. Comparison of total waiting time per episode using the experimental, DQN, Double DQN, GRU DQN, GRU DQN with dueling output and SR_DQN methods at J3 over scenario 3.

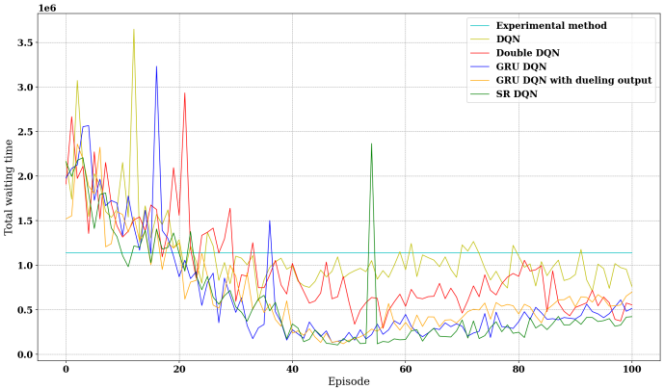


Fig. 17. Comparison of total waiting time per episode using the experimental, DQN, Double DQN, GRU DQN, GRU DQN with dueling output and SR_DQN methods at J4 over scenario 3.

The next tables present the measured metrics (minimum waiting time, average queue length, vehicle average speed, and Co2 emission) for each intersection.

Table 2. Measure metrics for J1.

Measure metrics	Scenarios	Methods					
		Experimental method	DQN	Double DQN	GRU DQN	GRU DQN With dueling output	SR DQN
Minimum Waiting time	1	570120	185657	167081	196331	159986	140892
	2	261703	81681	70457	55819	79127	51600
	3	1471826	1024774	849333	581724	628885	517383
Vehicle average speed	1	1.7671	1.9086	1.9613	1.8917	1.6461	1.9729
	2	2.3972	2.7861	2.7980	2.7993	2.8619	3.0396
	3	2.0209	2.0939	2.0884	2.1417	1.8241	2.2226
Average queue length	1	39.1128	29.4235	28.5574	32.1698	28.9540	27.4355
	2	22.013	14.9630	13.9920	12.5004	13.4315	12.052
	3	26.4346	36.3096	23.7234	24.5767	27.7945	22.8444
Emission Co2 (g)	1	170399.83	164897.82	168035.80	164475.32	187733.34	163015.51
	2	109333.44	98447.86	95990.42	94924.68	95398.15	86880.81
	3	425129.98	424679.15	424733.46	406820.04	475333.22	387528.86

Table 3. Measure metrics for J2.

Measure metrics	Scenarios	Methods					
		Experimental method	DQN	Double DQN	GRU DQN	GRU DQN With dueling output	SR DQN
Minimum Waiting time	1	544576	194399	248655	120768	108872	104886
	2	401197	138041	128040	88952	90136	86140
	3	1694353	838407	823231	775911	774697	741863
Vehicle average speed	1	1.6832	1.7093	1.7333	1.6867	1.7265	1.7338
	2	1.8675	2.0020	1.9720	2.0027	2.0010	2.0595
	3	1.5882	1.5031	1.5387	1.5037	1.5226	1.6067
Average queue length	1	36.4575	29.3046	31.4945	23.5924	23.8671	23.3836
	2	26.5754	18.9900	18.5684	19.0819	18.9130	18.3306
	3	29.9053	25.5677	25.3060	28.2410	28.5195	24.4384
Emission Co2 (g)	1	153317.52	151010.10	149186.83	147734.16	148915.11	146535.57
	2	115640.23	114430.40	110608.13	113188.57	111457.46	109754.71
	3	477144.60	474200.46	471742.68	481285.08	487311.74	459866.55

Table 4. measure metrics for J3.

Measure metrics	Scenarios	Methods					
		Experimental method	DQN	Double DQN	GRU DQN	GRU DQN With dueling output	SR DQN
Minimum Waiting time	1	680373	173736	160763	129434	129783	127990
	2	410467	134558	120932	118554	117003	111637
	3	294255	93228	89040	91642	87400	81309
Vehicle average speed	1	1.5930	1.6235	1.5959	1.5930	1.5438	1.6288
	2	2.1559	2.3402	2.4145	2.2902	2.1416	2.4626
	3	4.3042	4.8314	4.8657	4.9302	4.9413	4.9439
Average queue length	1	31.8031	23.5654	23.5134	21.4505	21.6953	21.1838
	2	20.1518	15.3326	14.9360	18.1808	18.7852	14.3996
	3	4.8347	3.1449	3.0033	2.9544	2.9366	2.9100
Emission Co2 (g)	1	128453.10	133798.77	133883.09	133173.68	136155.40	128350.42
	2	84718.74	82440.67	79546.02	83127.31	87272.43	75828.50
	3	122871.97	105926.67	105996.54	104521.54	104211.83	103887.81

Table 5. Measure metrics for J4.

Measure metrics	Scenarios	Methods					
		Experimental method	DQN	Double DQN	GRU DQN	GRU DQN With dueling output	SR DQN
Minimum Waiting time	1	409130	171805	156351	100350	95037	92790
	2	270213	77869	73978	62325	67685	57084
	3	1138118	599634	291011	131608	117889	104143
Vehicle average speed	1	1.6615	1.7045	1.7890	1.6419	1.6348	1.8355
	2	2.4800	2.9158	2.4837	2.2278	2.4597	3.0345
	3	2.010	2.8438	3.9079	4.3888	4.5395	4.7829
Average queue length	1	22.9840	21.6493	21.0009	19.499	19.5474	18.2057
	2	13.6583	9.6303	10.5204	11.8341	11.9350	8.7732
	3	18.3654	12.5845	6.8397	4.4607	4.1291	3.5901
Emission Co2 (g)	1	110844.06	113382.43	111629.66	115382.94	111435.24	100477.99
	2	63212.86	56686.15	59045.19	63764.53	63092.57	51981.72
	3	308836.25	250526.50	171220.99	139638.53	134034.79	125530.44

During closer examination of these results, the congestion has been reduced by using DRL methods (DQN, Double DQN and GRU DQN) compared to the experimental method. Our

SR DQN method provides, in general, better results in reducing congestion according to measured metrics (waiting time, vehicle average speed, average queue length, and emission Co2) than DQN, Double DQN, GRU DQN, and GRU DQN with dueling output methods.

Based on the above findings, it can therefore be asserted that the SR_DQN method proposed in this study gives quite good results regarding the measured metrics at junctions J1, J2, J3 and J4, with much fewer computational steps as compared to the experimental method.

5. CONCLUSION

In summary, the experimental method, used by the municipal authorities of Sidi Bel-Abbes city, and our method, i.e., SR_DQN, proposed in this study, was investigated to determine the optimal traffic light synchronization plan for this city. The proposed method introduced in this paper is expected to calculate all possible states, depending on the tramway position, which means that a large number of steps must be considered. It turned out that when our method is compared with a deep reinforcement learning method, such as DQN, and DDQN, it generally yields the desired results. It also offers better adaptability in traffic management and helps reduce traffic congestion.

The empirical results of this study demonstrate that the proposed method SR DQN significantly outperforms recent methods DQN, DDQN, GRU DQN and GRU DQN with dueling output, for minimum waiting time and some measure metrics (average speed vehicles, emission co2) over multiple scenarios. Although the minimizing waiting time is used as a reward for the agent, the single-agent DRL frequently changes phase, forcing vehicles into a "stop-start" cycle instead of allowing a gradual reduction in CO2 emissions, and imposes a severe constraint on the average speed of vehicles for a specific scenario at a specific intersection.

Future work will expand the simulation to a city-wide scale utilizing Multi-Agent Reinforcement Learning (MARL) by testing both the independent learning and communication learning which opens critical avenues, particularly regarding system stability and coordination overhead between intersections.

ACKNOWLEDGEMENTS

We gratefully acknowledge the financial support from Directorate General for Scientific Research and Technological Development (la DGRSDT).

REFERENCES

Bao, J., Wu, C., Lin, Y., Zhong, L., Chen, X. and Yin, R. (2023). A scalable approach to optimize traffic signal control with federated reinforcement learning. *Scientific Reports*, [online] 13(1). doi: https://doi.org/10.1038/s41598-023-46074-3

Cai, C. and Wei, M. (2024). Adaptive urban traffic signal control based on enhanced deep reinforcement learning. *Scientific Reports*, [online] 14(1), p.14116. doi: https://doi.org/10.1038/s41598-024-64885-w

Cho, K., Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Schwenk, H. and Yoshua Bengio (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv (Cornell University)*. doi: https://doi.org/10.48550/arxiv.1406.1078

Choe, C.-J., Baek, S., Woon, B. and Kong, S.-H. (2018). Deep Q Learning with LSTM for Traffic Light Control. *2022 27th Asia Pacific Conference on Communications (APCC)*. doi: https://doi.org/10.1109/apcc.2018.8633520

Chu, H-C., Liao, Y.-X., Chang, L. et Lee, Y.-H. (2019). Traffic Light Cycle Configuration of Single Intersection Based on Modified Q Learning, *Applied Sciences*, 9(21), art. no. 21. doi: https://doi.org/10.3390/app9214558.

Garg, D., Chli, M. et Vogiatzis, G. (2020). Multi-Agent Deep Reinforcement Learning for Traffic optimization through Multiple Road Intersections using Live Camera Feed, in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. Rhodes, Grèce : IEEE, pp. 1–8. doi: https://doi.org/10.1109/ITSC45102.2020.9294375.

Guastella, D.A. et Bontempi, G. (2023) *Traffic Modeling with SUMO: a Tutorial*. arXiv preprint arXiv:2304.05982. Disponible sur : https://doi.org/10.48550/arXiv.2304.05982.

Han, S., Zhou, W., Lu, J., Liu, J. et Lü, S. (2022). NROWAN-DQN: A stable noisy network with noise reduction and online weight adjustment for exploration, *Expert Systems with Applications*, 203, 117343. doi: https://doi.org/10.1016/j.eswa.2022.117343.

Hochreiter, S. et Schmidhuber, J. (1997). Long Short-Term Memory, *Neural Computation*, 9(8), pp. 1735-1780. doi: https://doi.org/10.1162/neco.1997.9.8.1735.

Kingma, D.P. et Ba, J. (2017) *Adam: A Method for Stochastic Optimization*. arXiv preprint arXiv:1412.6980. doi: https://doi.org/10.48550/arXiv.1412.6980.

Liang, X., Du, X., Wang, G. et Han, Z. (2019). A Deep Reinforcement Learning Network for Traffic Light Cycle Control, *IEEE Transactions on Vehicular Technology*, 68(2), pp. 1243-1253. doi: https://doi.org/10.1109/TVT.2018.2890726.

Liu, X.-Y., Zhu, M., Borst, S. et Walid, A. (2023) *Deep Reinforcement Learning for Traffic Light Control in Intelligent Transportation Systems*. arXiv preprint arXiv:2302.03669. doi: https://doi.org/10.48550/arXiv.2302.03669.

Lopez, P.A., Wiessner, E., Behrisch, M., Bieker-Walz, L., Erdmann, J., Flotterod, Y.-P., Hilbrich, R., Lucken, L., Rummel, J. et Wagner, P. (2018). Microscopic Traffic Simulation using SUMO, in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. Maui, HI : IEEE, pp. 2575–2582. doi: https://doi.org/10.1109/ITSC.2018.8569938.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. et Riedmiller, M. (2013) *Playing Atari with Deep Reinforcement Learning*. arXiv preprint arXiv:1312.5602. doi: https://doi.org/10.48550/arXiv.1312.5602.

N. Kodama, T. Harada, and K. Miyazaki, 2022. Traffic Signal Control System Using Deep Reinforcement

Learning With Emphasis on Reinforcing Successful Experiences. *IEEE Access*, vol. 10, pp. 128943–128950, https://doi.org/10.1109/ACCESS.2022.3225431

Rashidi HH, Albahra S, Robertson S, Tran NK and Hu B (2023) Common statistical concepts in the supervised Machine Learning arena. Front. Oncol. 13:1130229. doi: https://doi.org/10.3389/fonc.2023.1130229

Schaul, T., Quan, J., Antonoglou, I. et Silver, D. (2016). Prioritized experience replay, in *Proceedings of the International Conference on Learning Representations (ICLR)*. doi: https://doi.org/10.48550/arXiv.1511.05952.

Sutton, R.S. et Barto, A.G. (2018) *Reinforcement Learning: An Introduction*. 2e éd. Cambridge, MA: MIT Press.

Swapno, S.M.M.R., Nobel, S.N., Meena, P. et al. (2024).A reinforcement learning approach for reducing traffic congestion using deep Q learning, *Scientific Reports*, 14, 30452. doi: https://doi.org/10.1038/s41598-024-75638-0.

Van Hasselt, H., Guez, A. et Silver, D. (2015) *Deep Reinforcement Learning with Double Q-learning*. arXiv preprint arXiv:1509.06461. doi: https://doi.org/10.48550/arXiv.1509.06461.

Varaiya, P. (2013). Max pressure control of a network of signalized intersections, *Transportation Research Part C: Emerging Technologies*, 36, pp. 177–195. doi: https://www.sciencedirect.com/science/article/pii/S0968090X13001782.

Wang, B., He, Z., Sheng, J. et Chen, Y. (2022). Deep Reinforcement Learning for Traffic Light Timing Optimization, *Processes*, 10(11), 2458. doi: https://doi.org/10.3390/pr10112458.

Wang, P. et Ni, W. (2024). An Enhanced Dueling Double Deep Q-Network With Convolutional Block Attention Module for Traffic Signal Optimization in Deep Reinforcement Learning, *IEEE Access*, 12, pp. 44224-44232. doi: https://doi.org/10.1109/ACCESS.2024.3380454.

Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M. et de Freitas, N. (2016) *Dueling Network Architectures for Deep Reinforcement Learning*. arXiv preprint arXiv:1511.06581. doi: https://doi.org/10.48550/arXiv.1511.06581.

Woo, S., Park, J., Lee, J.Y. et Kweon, I.S. (2018). CBAM: Convolutional Block Attention Module, in Ferrari, V. et al. (éds) *Computer Vision – ECCV 2018*. Cham : Springer (Lecture Notes in Computer Science, vol. 11211). doi: https://doi.org/10.1007/978-3-030-01234-2_1.

Wu, C., Kim, I. and Ma, Z. (2023). Deep Reinforcement Learning Based Traffic Signal Control: A Comparative Analysis. *Procedia Computer Science*, 220, pp.275–282. doi: https://doi.org/10.1016/j.procs.2023.03.036

Youcef, H., Bilal, T., Walid, S. et Hicham, Z. (2024). An Intelligent Sequential Approach for Traffic Single Agent Lights Signals Control, in *2024 3rd International Conference on Advanced Electrical Engineering (ICAEE)*. IEEE, pp. 1–8. doi: https://doi.org/10.1109/ICAEE61760.2024.10783295.

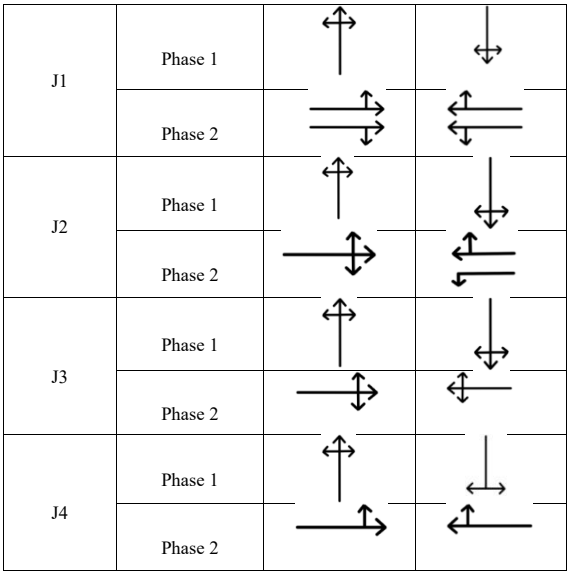
Zheng, Y., Luo, J., Gao, H. et al. (2025). Pri-DDQN: learning adaptive traffic signal control strategy through a hybrid agent, *Complex & Intelligent Systems*, 11, 47. doi: https://doi.org/10.1007/s40747-024-01651-5

APPENDICES

Appendix 1: The measured length of each lane at each intersection.

Intersection id	Geography direction	Measured Distances (m)
J1	From east to west	109.05
	From west to east	216.57
	From north to south	159.75
	From south to north	243.70
J2	From east to west	211.92
	From west to east	157.83
	From north to south	180.29
	From south to north	153.61
J3	From east to west	354.46
	From west to east	152.94
	From north to south	81.98
	From south to north	98.50
J4	From east to west	152.61
	From west to east	288.70
	From north to south	100.22
	From south to north	55.65

Appendix 2: The real phases for each intersection.



Appendix 3-A: The number of cars for each intersection.

Junction		Direction (from)				Total number of cars
		N	S	W	E	
J1	N		126	139	132	397
	S	58		139	132	329
	W	115	126		396	637
	E	115	168	418		701
J2	N		187	72	163	422
	S	133		72	163	368
	W	178	187		487	852
	E	133	250	216		599
J3	N		38	30	103	171
	S	29		30	103	162
	W	38	77		310	425
	E	29	77	60		166
J4	N		30	144	180	354
	S	0		0	0	0
	W	60	15		420	495
	E	60	15	336		411

Appendix 3-B: The number of buses for each intersection.

Junction		Direction (from)				Total number of Buses
		N	S	W	E	
J1	N		1	3	0	4
	S	1		2	0	3
	W	1	1		0	2
	E	1	1	1		3
J2	N		0	2	2	4
	S	0		2	2	4
	W	0	0		2	2
	E	0	0	2		2
J3	N		0	0	0	0
	S	0		0	0	0
	W	0	0		0	0
	E	0	0	0		0
J4	N		0	0	0	0
	S	0		0	0	0
	W	0	0		0	0
	E	0	0	8		8

Appendix 3-C: The number of motorcycles for each intersection.

Junction		Direction (from)				Total number of motorcycles
		N	S	W	E	
J1	N		1	1	2	4
	S	1		1	2	4
	W	2	1		4	7
	E	1	0	2		3
J2	N		1	1	2	4
	S	1		1	2	4
	W	4	1		4	9
	E	1	4	2		7
J3	N		1	0	2	3
	S	1		0	2	3
	W	1	1		6	8
	E	0	1	0		1
J4	N		1	3	2	6
	S	0		0	0	0
	W	1	1		4	6
	E	1	1	5		7

Appendix 4: Pseudocode for focus gate and reaction equation.

```
class networks(nn.Module):
    def __init__(self):
        super(networks, self).__init__()
        self.fc1 = nn.Linear(input_dims, fc1_dims)
        # ----------------------------------------------------------------
        # for a new method
        self.input_1 = nn.Linear(fc1_dims, fc2_dims)
        self.hidden_1 = nn.Linear(fc2_dims, fc2_dims)

        self.input_2 = nn.Linear(fc1_dims, fc2_dims)
        self.hidden_2 = nn.Linear(fc2_dims, fc2_dims)

        self.input_3 = nn.Linear(fc1_dims, fc2_dims)
        self.hidden_3 = nn.Linear(fc2_dims, fc2_dims)

        self.input_4 = nn.Linear(fc1_dims, fc2_dims)
        self.hidden_4 = nn.Linear(fc2_dims, fc2_dims)

        self.value_states = nn.Linear(fc2_dims, 1)
        self.advantage = nn.Linear(fc2_dims, n_actions)

        self.optimizer = optim.Adam(self.parameters(), lr=lr)
        self.loss = nn.MSELoss()

    def forward(self, x, h_prev_1):
        fc1 = F.relu(self.fc1(x))
        # reset gate
        r_t = torch.sigmoid(self.input_1(fc1) + self.hidden_1(h_prev_1))
        # update gate
        z_t = torch.sigmoid(self.input_2(fc1) + self.hidden_2(h_prev_1))
        # candidate gate
        g_t = torch.tanh(self.input_3(fc1) + self.hidden_3(r_t * h_prev_1))
        # ----------------------------------------------------------------
        # Hidden state update
        h_t = (1 - z_t) * h_prev_1 + z_t * g_t
        focus_gate = F.softsign((self.input_4(fc1)) + self.hidden_4(h_prev_1))
        z = h_t / (1 + torch.log(2 + focus_gate))  # reaction gate
        value_state = self.value_states(z)
        advantage = self.advantage(z)

        Q = (value_state + advantage - torch.mean(advantage, dim=1,
keepdim=True))
        return Q, h_t
```