Received: May 4, 2022. Revised: June 14, 2022.

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Cooperative Traffic Signal Control of n-intersections Using a Double Deep Q-Network Agent

Salma El bakkal[1]* **Abdellah Lakhouili[1]** **El Hassan Essoufi[1]**

[1]*Hassan First University of Settat, Faculty of sciences and Techniques, Mathematics, Computer Science and Engineering Sciences laboratory, 26000, Settat, Morocco*
* Corresponding author's Email: s.elbakkal@uhp.ac.ma

Abstract: Intelligent traffic controller leads to manage traffic at intersection in order to minimize traffic congestion and has been intensively researched for a several decades. Multi-intersection cooperative traffic signal control (CTSC) is an efficient system that has received a great deal of attention and development in recent years. One problem with multi-intersection CTSC is that controller's actions are based only on the traffic state or on the decisions taken at previous time step (t-1) at adjacent intersection. To address this problem, in this work a Double Deep Q Network Cooperative Traffic Signal Controller (CTSC-DDQN) is proposed. The CTSC-DDQN algorithm is a reinforcement learning agent that, depending on current traffic conditions, changes the traffic phase distribution order for n-intersection simultaneously. Experimental results under real scenarios show that the proposed approach outperforms other static approach which fixe the time length and the phase order despite the traffic situation and actuated controllers which change the traffic light properties based on the queue length in term of average Q-length and average waiting which eventually leads to mitigate traffic congestion. The results show that our could minimize the average waiting time by up to 79% and the average queue length by up to 80%.

Keywords: Traffic congestion, Cooperative traffic systems, Deep reinforcement learning, Multi-intersection management.

1. Introduction

All Generally, traffic flow at the intersection is managed by traffic light controllers. As a result, controlling traffic light plan at signalized intersection is a critical issue. Furthermore, traffic congestion increases energy consumption, car emission and vehicles traveling time which is a serious problem in large cities. The goal of the intersection management is to find the best signal plan to minimize the average waiting time and the queue length of vehicles.

Based on the literature there are three main types of the traffic light systems. The static controllers that use the historical traffic data collected from sensors to set a fixed periodic model adapted to different times of a day [1], which not responsive to the actual traffic situation. The actuated systems that predict traffic phase based on instantaneous traffic conditions by increasing the traffic time length based on the queue length of the intersection lanes [2] And the adaptive systems that use longer-term information length such as arrival rate, waiting time and traveling time to predict the traffic phases order and traffic phases. Recently, the adaptive systems were widely discussed and has demonstrated a strong potential to efficiently mitigate urban traffic congestion in order to attain acceptable goals compared to the fixed and actuated systems [3]. Scats [4] and Scoots [5] are the famous adaptive traffic systems that control intersection in real time. Scoot aims to minimize the average queue length by modifying the traffic light plan using different data such as traffic flow, queue length, average velocity. This data is gathered from advanced detectors placed near to stop lines. And Scats tries to propose a time plan using only the number of vehicles stacked at stop lines. Subsequently, several adaptive systems were proposed using different methods. Particularly after

the artificial intelligence revolution. Since 1990 researchers started to use machine learning techniques to control traffic signals at intersection [6]. Reinforcement learning (RL) is the most method applied in traffic management systems because it helps the controller to take decision based on the relationship between the actions space and the state space which are learned by interaction of the agent with the environment. But RL techniques were limited only to Q learning table and linear function to predict the Q value which impose to use a small-size state space for example, the number of waiting vehicles or the general data of road traffic [7]. On the other hand, we cannot describe the complexity of the traffic with such limited information. Deep reinforcement learning (DRL) has been developed. And some researchers have proposed to use these techniques to develop new adaptive traffic management methods. The most applied method is the deep Q network (DQN). It consists to enable a neural network to approximate the best actions can take an agent at each given state [8]. This model was adopted in [9] where the authors propose a DQN model that manage an intersection by changing the order phases execution in order to decrease the cumulative vehicle delay between two actions. Also, in [10] authors propose a DQN approach that maximizes the traffic flow through the intersection in real time. Another DRL approach called Dueling DQN was adopted in [11] where the neural network choses the best phase duration based on the position and speed of vehicles at an intersection. In [12] authors propose an adaptive controller based on two agents which are denoted by two different states and change the control of green lights to make the phase sequence fixed and control process stable at intersection using the new version of DQN which called double deep Q network (DDQN).

All the works cited previously, focus only to manage traffic at isolated intersection which signifies that the signal plan at each intersection is unaffected by the signal time plan at other adjacent intersections. Otherwise, congestion at the intersections becomes more serious due to the urban expansion, which has led to an increase in the number of intersections, particularly in large cities. Thus, to manage intersections in an urban environment, it must be considered that all the intersections are not isolated. That is, the state of one intersection is frequently influenced by the state of the neighbouring intersections. As a result, traffic signal systems for multi-intersection becomes complex and difficult to solve. Therefore, to ensure a good traveling experience for vehicles in urban areas, it's necessary to investigate the signal plans at multi adjacent intersections. To solve this problem, researchers use multi-agent systems in order to guarantee the communication between intersection. The idea is describing the intersections network with a multi agent system where each intersection is managed by one agent and each agent can access to the collective representation of the traffic environment collected by all agents installed at adjacent intersection. Subsequently, the agent chose the best action based on different algorithm. For example, in [13] the agent chooses the time length using a deep Q network approach and in [14] adjust the green time using a Knowledge Sharing Deep Deterministic Policy Gradient.

The control approaches for multi-intersection have already been discussed. Most of the existed studies that control signals at multi-intersection adapts the next traffic light plan (which will be executed at the next time step t+1) based on the different data collected from the other intersections at the previous time step t. Since all the agent executes their chosen action simultaneously. Then, the traffic at intersection could be affected by the actions executed at the adjacent intersection. And this case has not been investigated in the literature. Thus, to address this issue, this paper proposes a cooperative traffic light system that change the phase distribution order at n-intersection using a single double deep Q network agent (CTSC-DDQN). The CTSC-DDQN agent learns the at each time step the state of all the intersections and choose the best phase order with a view to minimize the average queue length.

The outline of this work is organized as follows: Section 2 present the background of the method used in this work. The proposed approach is deeply presented in Section 3. In Section 4, simulation setups and parameters are described where the numerical results are discussed in Section 5. The conclusions and the perspectives of this paper are presented in Section 6.

2. Background

This section provides an introduction to RL, DQN, and DDQN. Table 1 provides description of different variables used in this article.

2.1 Reinforcement learning

Reinforcement learning (RL) is one of the most effective proposed methods for solving Markov Decision Process (MDP) problems. The original RL agent learns several state-action pairs in order to maximize its reward (or cost).

Table 1. Annotation list

Parameter	Description
a_t	Action selected by the agent at instant t
s_t	State received from the environment at instant t
Q_t	Expected Q-value at instant t
α	Learning Rate
Argmax()	Function to return the action that gives the max Q-value
R_t	Reward received at instant t
δ_t	Temporal difference between rewards
γ	Discount Factor
Y_t	Output of a neuron
$\varphi()$	Sigmoid function
w_{kj}	Weight value of a neuron
e_t	Experience of the agent at instant t
θ_t	Target network parameter
θ_t^-	Online network parameter

RL system is composed by three sub elements: state space, action space, and reward value. firstly, the RL agent defines the current state $S_t \in S$ at time instant t, and chooses an action $a_t \in A$ based on a learning rate α. This latter is a number between 0 and 1 and defines whether the agent will choose a random action to explore more the environment or whether it will exploit the environment by selecting the action that has the maximum expected Q-value based on its experience. The function used to choose an action is described as following:

$$a_t^* = \alpha \; argmax_a \; Q_t(s_t, a_t) \tag{1}$$

Then the environment executes the chosen action and send to the agent the new state S_{t+1} and the delayed reward R_{t+1} ,subsequently, and using the Q-function, the agent modifies the Q-value of the state-action pair (s_t, a_t) using the following equation [15]:

$$Q_{(t+1)}\ (s_t, a_t\ ) = Q_t\ (s_t, a_t\ ) + \alpha\ \delta_t\ (s_t\ \ , a_t) \tag{2}$$

where $\delta_t(s_t\ , a_t)$ describes the temporal difference of rewards that based on the Bellman equation in term of delayed rewards between two estimations which is presented as following [36]:

$$\delta_t(s_t\ , a_t) = R_{t+1}(s_{t+1}) + \gamma max_{a\epsilon A} Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \tag{3}$$

Where $0 \leq \gamma \leq 1$ is the discount factor that indicates the preference for the discounted reward, and $\gamma max_{a\epsilon A} Q_t(s_{t+1}, a)$ is used to represent the long-term discounted reward, while the delayed cumulative reward $R_{t+1}(s_{t+1})$ defines the short-term reward. And finally, the agent uses a two-dimensional Q-table to store their respective Q-values. The different steps that summarize the RL algorithm are presented in the following algorithm 1:

Algorithm 1: Traditional RL algorithm

1: Procedure

2: Retrieve the environment state $\boldsymbol{S_t \in S}$

3: Choose action $\boldsymbol{a_t \in A}$.

4: Receive the reward $\boldsymbol{R_{t+1}(s_{t+1})}$.

5: Update Q-value $\boldsymbol{Q_{(t+1)}\ (s_t, a_t\ )}$.

6: End Procedure

2.2 Deep Q-network:

The traditional RL algorithms are effective for smaller state-action spaces. However, when the state-action spaces are huge or continuous, the RL converges slowly and fail to find the optimal policy. Thus, to solve this problem, researchers proposed to use the Artificial neural network (ANN). In [16] and [17] the authors combined Q learning and Convolutional neural network (CNN) [18] in order to propose a new method called deep Q-network (DQN). As shown in Fig. 1, in DQN, the state information fed to the CNN network using the input layer. This later represents the state space and it's coupled to a hidden layer that describes the different states given by a nonlinear function. Therefore, the output layer generates the Q-values $Q_t(s_t, a_t)$ of each possible action $a_t \in A$. Note that the neurons of the input and the output layers are fully linked to those in the hidden layer and each link is described with a weight value w_{kj} That define the significance of the x_j compared to the other inputs. The output of a neuron is represented as following:

$$Y_k = \varphi\left(\sum_{j=0}^{m} w_{kj}\ ,\ x_j\right) \tag{4}$$

Where $\varphi()$ is a Sigmoid function that expresses the relation between non-linear and linear functions at every neuron k of the network. The target network and the experience replay present the main features of DQN [16]. During the training process, the agent stores in the replay memory an experience composed by the current state s_t, the chosen action a_t, the received reward value R_{t+1} and the state s_{t+1},$e_t(s_t, a_t, R_{t+1}, s_{t+1})$, and subsequently selects a random experience in order to train the network and calculate the weight θ_t which will be used to predict the best Q-values at time t $Q(s, a; \theta_t)$. In DQN two neural networks are used. The first one is used to estimate the Q-value of each action-state pair and it's called by the online network. And the second one

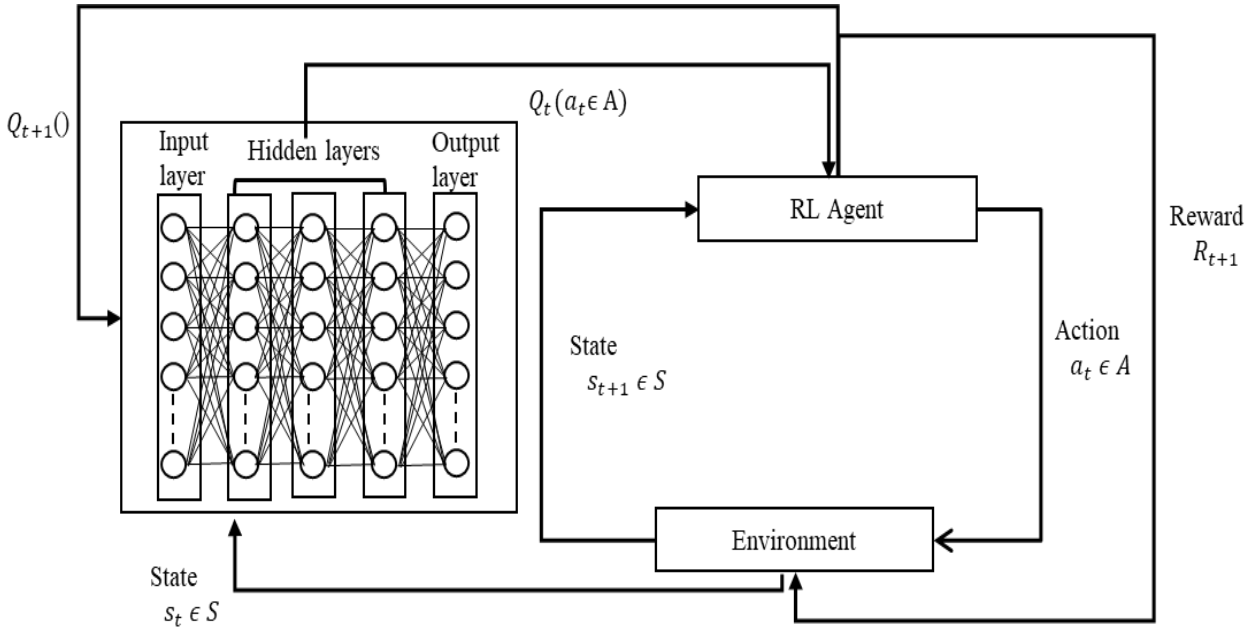


Figure. 1 Deep reinforcement learning agent at t and t+1

which is called by the target network, is used to provide the objective function Y_t . Note that the online network parameter θ_t is updated every step and the target network parameter θ_t^- is copied from the online one every fixed N steps. The objective function is given as following:

$$Y_t = R_t + \gamma max_{a\epsilon A}\varphi(s_{t+1}, a; \theta_t^-) \quad (5)$$

Or it can be transformed into:

$$Y_t = R_t + \gamma\left(Q(s_{t+1}, argmax_a + Q(s_{t+1}, a; \theta_t^-); \theta_t^-)\right) \quad (6)$$

2.3 Double deep Q network DDQN:

In DQN method the expression $max_{a\epsilon A}\varphi(s_{t+1}, a; \theta_t^-)$ used in Eq. (5) describes that the selection and evolution process use the same target network parameter θ_t^- which may lead to overestimation. Thus, to reduce van and haslet in proposed to use the online parameter θ_t for action selection, and the target network parameter θ_t^- is used for action evaluation. Therefore, the new objective function will be transformed in DDQN into:

$$Y_t = R_t + \gamma\left(Q(s_{t+1}, argmax_a + Q(s_{t+1}, a; \theta_t); \theta_t^-)\right) \quad (7)$$

The different steps that summarize the different steps of the DDQN algorithm are provided in the following algorithm 2:

Algorithm 2: Traditional DDQN agent

1: Procedure
2: FOR episode=1 to E do:
  3: Observe current state $\boldsymbol{S_t \in S}$
  4: For t=1 to T do
    5: Select action $\boldsymbol{a_t \in A}$
    6: Receive delayed reward
    7: Store experience
    8: Set target $\boldsymbol{Y_t}$
    9: Perform a gradient descent
    10: Update networks Q-value.
  11: END FOR
12: END FOR.
13: END Procedure

3. Proposal approach

In order to build a traffic light system for multi-adjacent intersections using RL, we need to define the basic elements, especially the state space, actions space and the reward function. In this section, we present how the three elements are defined in our proposed approach.

3.1 State space:

Based on the literature, there are different ways to represent the states space, for example, in [9, 10] describe the state with the presence and velocity of the cars at the intersection's lanes. Otherwise in [11] the state was presented by the position of the vehicles at the lanes. However, in these works the agent mange only one intersection so those representation are sufficient to represent the environment state. But in our case, and to teach our agent to act smartly, we need to give them the best description about the

environment. And to describe deeply a vehicular network, we should extract two information, the number and the velocity of the vehicles at all the lanes. Since we will manage our network with one agent, we need to simplify the state space. In this work we describe the environment with the average velocity at a lane in every intersection of the network, which has not been investigated in the literature. The average velocity presents the ratio of the sum of the velocity and the total numbers of vehicles staying in the lane. If this parameter converges to 0 means that all the vehicles struggling in this lane and if is converging to the maximal velocity value means that we have a fluent traffic at this lane.

Our studied network contains three types of lanes. Arrival lanes where the vehicles enter the network, departure lanes which the vehicles can use it to leave the network, and the internal lanes which connect the intersections. Therefore, if our network contains 4 intersections means that we have 56 lanes which means that our state dimension equal to the lanes number.

Let's consider the Fig. 2 as an example to demonstrate how our builds the state vector values. This figure shows an example of the traffic states at three lanes four-way intersection. To generate the state vector, we translate the intersection to a cell grid and replace the vehicles with their speed values. Thus, we can obtain the state vector by calculating the average speed at each lane and built the vector values as shown in the figure. Denote that in our work we suppose that the maximal velocity value equal to 40 m/s. And we describe the free lanes by -1 in order to differentiate the free lanes from the congested full lanes which also have an average speed equal to 0.

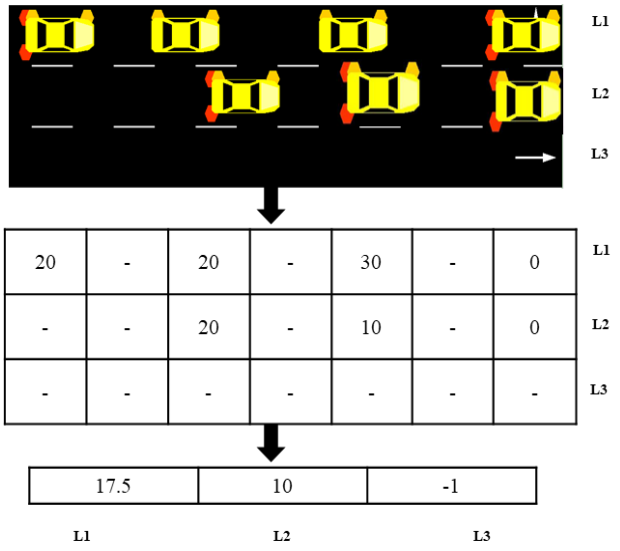
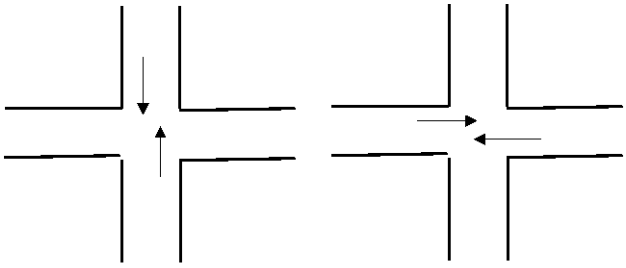


Figure. 2 State detection process



(a) (b)

Figure. 3 Pahse types: (a) North-south phase and (b) East-west phase

3.2 Action space:

After the agent has receive the environment state, it should choose one from the set of all available actions. Based on the literature, authors propose to fix the traffic phase splits and modify the order of the phases such in [19,10], or such in [11] where they proposed to fix the order and change the phases splits. Otherwise, in this work we fixed the green duration for each intersection and we will modify the executed phase every step. Our system contains N intersections managed by a traffic light system. Each one has a program that contains 2 phases, namely by North-South and East-West phase presented in fig 3. The agent should choose an appropriate phase combination for all the intersections to well guide vehicles in the network based on the current traffic state.

To ensure a good learning experience for our agent, we assumed in this work that at each step, the agent has two possibilities, either it chooses the action based on the fact that the number of intersections having the North-South (NS) phase is equal to those having East-West (ES) phase or it will activate the same phase for all the intersections in the next step. For example, if we have 4 intersections the number of possible actions is 8 and the set of possible actions is described in the Table 1.

Due to traffic control security measures, the chosen action using our agent will note be executed immediately. And to address this issue, an additional traffic signal phase configuration will be added before the chosen action.

Table 2. Action space

	I1	**I 2**	**I 3**	**I 4**
A 1	NS	ES	NS	ES
A 2	ES	NS	ES	NS
A 3	ES	ES	NS	NS
A 4	NS	NS	ES	ES
A 5	NS	ES	ES	NS
A 6	ES	NS	NS	ES
A 7	NS	NS	NS	NS
A 8	ES	ES	ES	ES

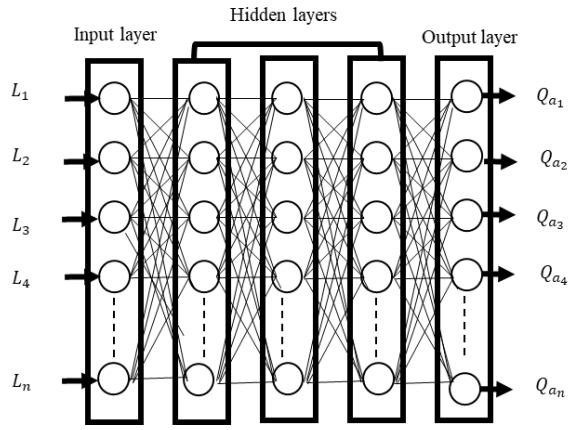


Figure. 4 CNN network architecture

Instead of instantly transitioning from the old traffic signal phase to the new chosen one, a series of yellow traffic signal phases will be executed based on the current phase. All the proposed actions have a yellow configuration which cannot be selected by our agent as an action, but are part of the traffic configuration in order to slow down and stop the traffic before activating the red light.

3.3 Reward:

The most important element of our reinforcement learning agent is the reward function. After the agent has executed the selected action, it receives a value called the reward. The reward is the value that helps the agent to builds an optimal policy which leads to maximize the cumulative long-term reward. In this work, we define the reward as a change in cumulative average queue length in all the intersections between actions. This allows for the reward to be positive which means that the agent will be rewarded or negative, which will be observed as a punishment for the agent. Note that to receive a reward the agent should observe the new environment state which was influenced by the chosen action and calculate the new reward using the following equation:

$$R_t = \sum_{i=1}^{n} AVQ_{i,t-1} - \sum_{i=1}^{n} AVQ_{i,t} \qquad (8)$$

Where AVQ describes the average queue length and n describes the total number of network intersections.

3.4 Algorithm:

In this work we inspired by DDQN and we propose a Convolutional Neural network architecture which will be used to train our agent. The whole network is presented in Fig. 4.

The state is fed from the input layer that has NL neurons; each represents the average speed at corresponding lane $L \in NL$. Subsequently, the information flows forward to a hidden layer which composed with three fully connected (FC) each has 300 neurons connected with Relu function that performs gradient descent, and finally, the information arrives to the output layer that has multiple neurons, each representing the Q values corresponding to all possible action.

Note that our algorithm will use two networks to calculate the Q target based on the traditional DDQN presented in algorithm 2. To effectively manage the traffic signal at intersections, our agent uses the algorithm presented in algorithm 2. At episode $e \in E$ the agent discovers the environment state $S_t \in S$ as part of the initialization. At time instant $t \in T$. Then, the agent selects an action $a_t \in A$ and store experience in a replay memory. Subsequently, the agent randomly selects a mini-batch of experiences from the replay memory to set the target value and update the networks with the new generated value.

4. Simulation setup and parameters:

In order to show the performance of the proposed approach, it is significant to set appropriate traffic network parameters.

As shown in Fig. 5, our network size is 4*4, which means that they are 16 adjacent intersections. Each intersection has 3 lanes. Where the left lane is for left turn, the middle lane allows vars to go straight only, and right lane allows vehicles to turn right. See Fig. 6 for more details.

The method by which vehicles are arriving to the network has a significant impact on the simulation's quality.

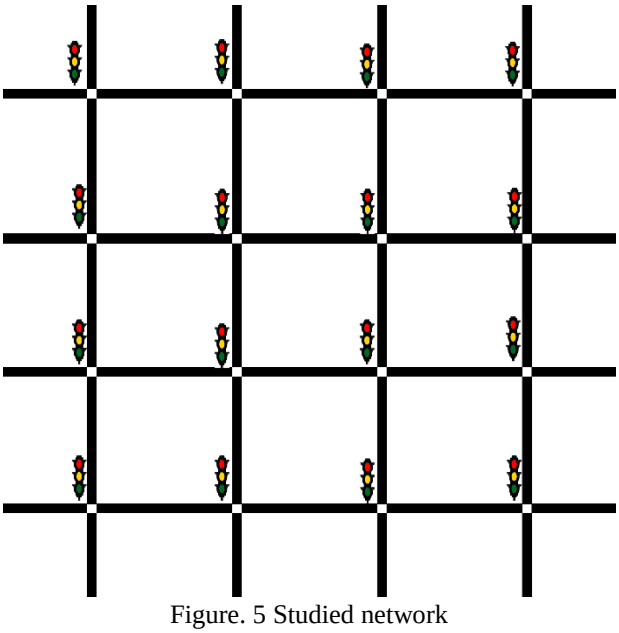


Figure. 5 Studied network

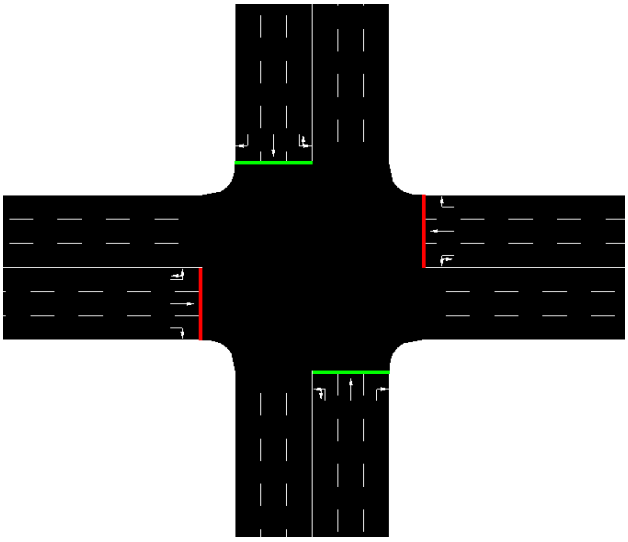


Figure. 6 Intersection configuration

Table 3. Arrival distribution

λ value	Traffic state
$\lambda = 0.1$	free traffic state
$\lambda = 1$	jammed traffic state
$\lambda = [0,1]$	random traffic state

Table 4. DDQN parameters

Parameters	Values
Replay memory size	60 000
Experience sampling	0.5
Discount factor γ	0.7
Learning rate α	0.0002
Mini-batch size	200
Starting ϵ	1
Ending ϵ	0.01
Exploring episodes	100
Exploiting episodes	2000

The most popular method selects a random number using probability distribution that meet the time intervals between vehicles. Experimental research has shown that different vehicle flows are estimated by different probability [20]. In this paper, the vehicles production rat flows the poison process and to test different traffic scenarios we represent in Table 2 the production rate λ setting tested in this work. For example, to generate a free traffic we set $\lambda = 0.1$ which describe that every 10 seconds a vehicle enters

into the network from all the inputs, and for random traffic state we vary the λ values from 0 to 1 randomly during simulation.

In order to get a good traffic signal policy, our DDQN agent is trained for 2000 episodes, each episode lasts 3600s. Detailed parameters of the agent networks are presented in Table 3. In this work, we used the ADAptive Moment estimation (Adam) [21] to update the learning rate during the training process.

To simulate our algorithm, we used Simulation of Urban Mobility (SUMO) (i.e., version 1.9.2). SUMO is an open-source simulator, which simulates in real-time [22]. The RL agent was developed with python, which provides a set of open-source library that help to create our algorithm. And to connect our python code with SUMO program, the TraCI (Traffic Control Interface) was used to facilitate the interaction between the python files and SUMO simulator using TCP/IP protocol.

5. Results and discussions

In this section we evaluate our proposed CTSC-DDQN method by comparing its results with those obtained from the static, the actuated approaches under free traffic state, jammed traffic state and random traffic state. Note that the static and actuated algorithm was implemented by the SUMO simulator. This simulation will lead to compare the tree algorithm using the same constraint and network parameters.

5.1 Cumulative delayed reward:

With a focus to lead our agent to learn the best action value function. We perform two types of tests: exploring and exploiting tests. The exploring test is when the agent takes random actions with no consideration for reward. This test helps our agent to explore more the environment attempting to discover the best action-state pair. The accumulated delayed reward for the CTSC-DDQN under free, jammed and random traffic state during the exploring test is shown Fig. 7.

The figure shows a divergence in term of reward values and this caused because the agent is acting randomly which make them unstable during this test phase.

Otherwise, the agent's behaviour changes during the exploiting tests as shown in Fig. 8.

In this figure the cumulative reward is more optimized and more stable in comparing to the exploring phase and this is because the agent exploits the environment and take into consideration the reward values before taking action during episode time. In addition, the CTSC-DDQN achieves a higher delayed reward in different traffic state as shown in Fig. 8 which explain that our agent adapts with any traffic situation and keeps its stability while time is increasing.

5.2 Average queue length

Fig. 9 presents the average queue length of the three kinds of traffic controllers under free, jammed

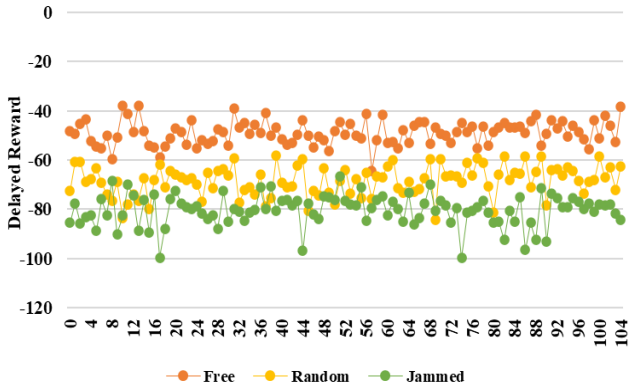


Figure. 7 Cumulative reward during exploring test

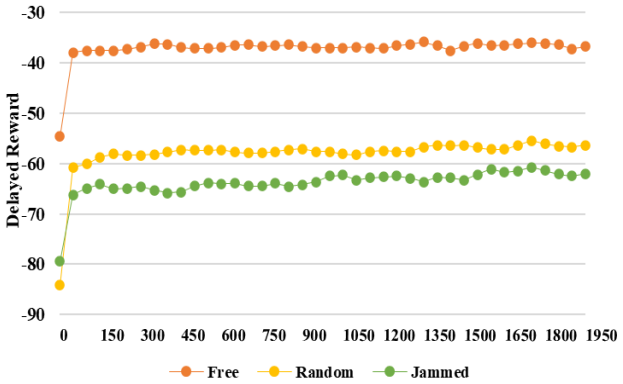


Figure. 8 Cumulative reward during exploiting test

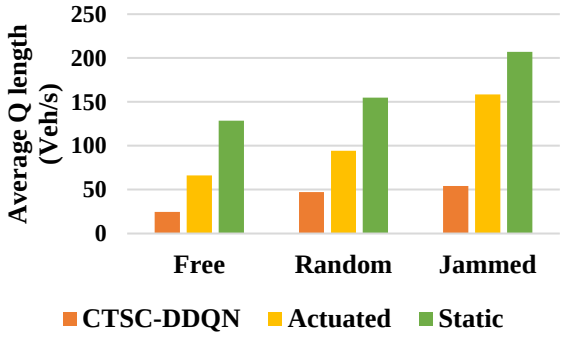


Figure. 9 Average Q-length results

and random traffic state as the time increase. The average queue length for free traffic varies up 129 vehicles for static controllers and 66 vehicles for the actuated controllers. Otherwise, with the CTSC-DDQN, the average queue length is less than 25 vehicles. See Fig. 9.

For the jammed traffic state, the queue length varies up to 207 with static approach, up to 159 vehicles with actuated approach. Otherwise, with our approach, the average queue length is less than 54 vehicles and its stable during the time. See Fig. 9.

Similar behaviour is observed for the random traffic state. The average queue length varies up to 154 vehicles with static approach and up to 94 vehicles with the actuated approach where it is less than 47 vehicles with CTSC-DDQN approach. See Fig. 9.

From these results we can confirm that our proposed approach reduces the queue length by up to 70% and 50% respectively from the static and actuated approaches under random traffic state. And reduces by up to 74% and 66% respectively, compared to the static and actuated approaches under jammed traffic state. And under free traffic, with our approach the average the average queue length is reduced by up to 80% from the static approach and up to 62% from the actuated approach.

5.3 Average waiting time:

The average waiting time in the network of the three types of traffic controllers under free, jammed and random traffic state as the time increases is shown in fig. 10.

For free traffic state, the average waiting time varies to 12 min for actuated approaches. However, the CTSC-DDQN has his average waiting time less than 5 min and it's stable with time.

For jammed and random state, the average varies up to 28min and 18min, respectively with static controllers and with 18 min and 11 min with the actuated controllers. Otherwise, with CTSC-DDQN controllers the average waiting time varies less than 6 min and 4 min respectively under jammed and random traffic state.

From these results we can confirm that our proposed approach reduces average waiting time by up to 78% and 64% respectively from the static and actuated approaches under random traffic state. And reduces by up to 79% and 66% respectively, compared to the static and actuated approaches under jammed traffic state. It can be seen that just modifying the phase distribution order led to an important improvement in term of the average queue and average waiting time in comparison with the

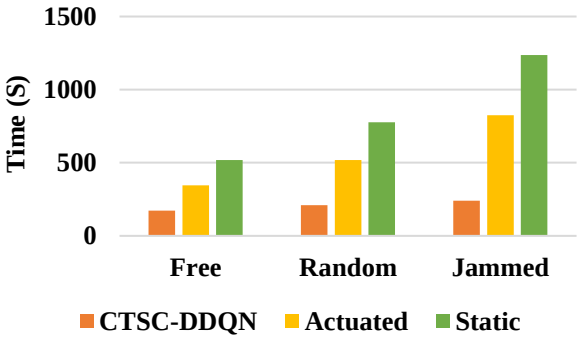


Figure. 10 Average waiting time results

Table 5. Comparison of methods

Adaptive algorithm	Number of intersections	Compared against	Queue length improvement	Waiting time improvement
Our approach	16	Static method	**79%**	**80%**
MADQN [13]	16	Static method	75%	70%
Dual DDQN [12]	Isolated intersection	Static method	57%	72%

static and actuated controllers under different scenarios.

The proposed CTSC-DDQN aims to reduce the average waiting time and average queue length at multiple intersections using a single agent based on the traffic state in real time. Hence, it reduces the average waiting time by up to 79 % and the average queue length by up to 80%. The good performance obtained is explained by the use of the proposed algorithm which manage traffic light by changing the phase distribution based on the adjacent intersection states. When the static controller fixes the signals' time under different traffic scenarios. And the actuated controller changes the time length based on the queue length of the vehicles stacked at the intersection and it's don't take into consideration the arrival vehicles from the adjacent intersection. Also, the existed adaptive system proposed in [12] which manage only one isolated intersection it reduces the average queue length by only 72% and the waiting time by 57% compared to the static controller and this is because it don't take into consideration the adjacent traffic situation. In addition, the multi agent deep Q network approach which proposed in [13] reduces the average queue length by only 75% and the waiting time by only 70% and this is because it is based only on the traffic situation before taking action in addition these results are overestimated because it used the DQN and as shown in section 2 the DDQN resolve the DQN overestimation which means that our results are more significates than the [13] results. For more details we summarized in Table 3 the different results obtained using our proposed approach and the existing approaches in literature.

6. Conclusion

One of the most challenging problems within traffic light systems is the traffic coordination at intersections. The advancements in deep reinforcement learning methods, especially the double deep Q network algorithm have shown great potentials for improving traffic light system performance. In this work, the cooperative traffic signal control is investigated, a double deep Q network agent is proposed to manage n-intersections in real time. the proposed approach uses an artificial intelligence method called double deep Q-network (DDQN) which use a single agent method in order to overcome the problem of dimensionality. This work proposes a cooperative traffic light controller that uses double deep Q network to describe and store the traffic states, also uses target network and experience replay to keep training stable during episodes time.

Simulation of different traffic scenarios using SUMO simulator demonstrates that CTSC-DDQN outperforms other static and actuated approaches by minimizing the average waiting time by up to 79% and the average queue length by up to 80%. Recently, traffic light systems have received much attention and research, although, they are still a developing field.

As part of Our future works, we would like to focus on improving the performance of the proposed approach by including multi-agent system in order to facilitate the communication between intersection. Also, it will be interesting to take account the optimization of the green time length.

Conflicts of Interest

The authors declare no conflict of interest.

Author Contributions

Contributions of author are elaborated as follows.

EL Bakkal salma: conceptualization, implementation and writing the paper.

Lakhouili Abdellah and EL Hassan Essoufi: Methods validation, Paper reviewing, formal analysis and supervision. All authors read and approved the final manuscript.

References:

[1] B. Yin, M. Dridi, and A. E. Moudni, "Traffic network micro-simulation model and control algorithm based on approximate dynamic programming", *IET Intelligent Transport Systems*, Vol. 10, No. 3, pp. 186-196, 2016.

[2] S. B. Cools, C. Gershenson, and B. D'Hooghe, "Self-organizing traffic lights: A realistic simulation", *Advances in Applied Self-Oganizing Systems*, pp. 45-55, 2013.

[3] P. Jing, H. Huang, and L. Chen, "An adaptive traffic signal control in a connected vehicle environment: A systematic review", *Information*, Vol. 8, No. 3, p. 101, 2017.

[4] A. G. Sims and K. W. Dobinson, "The Sydney coordinated adaptive traffic (scat) system philosophy and benefits", *IEEE Transactions on Vehicular Technology*, Vol. 29, No. 2, pp. 130-137, 1980.

[5] P. Hunt, D. Robertson, R. Bretherton, and M. C. Royle, "The scoot on-line traffic signal optimisation technique", *Traffic Engineering & Control*, Vol. 23, No. 4, 1982.

[6] S. Araghi, A. Khosravi, and D. Creighton, "A review on computational intelligence methods for controlling traffic signal timing", *Expert Systems with Applications*, Vol. 42, No. 3, pp. 1538-1550, 2015.

[7] A. J. Miller, "A queueing model for road traffic flow", *Journal of the Royal Statistical Society: Series B (Methodological)*, Vol. 23, No. 1, pp. 64-75, 1961.

[8] H. Hasselt, "Double q-learning", *Advances in Neural Information Processing Systems*, Vol. 23, 2010.

[9] W. Genders and S. Razavi, "Using a deep reinforcement learning agent for traffic signal control", *arXiv Preprint arXiv:1611.01142*, 2016.

[10] A. Vidali, L. Crociani, G. Vizzari, and S. Bandini, "A deep reinforcement learning approach to adaptive traffic lights management", *WOA*, pp. 42-50, 2019.

[11] X. Liang, X. Du, G. Wang, and Z. Han, "Deep reinforcement learning for traffic light control in vehicular networks", *arXiv Preprint arXiv:1803.11115*, 2018.

[12] J. Gu, Y. Fang, Z. Sheng, and P. Wen, "Double deep q-network with a dual-agent for traffic signal control", *Applied Sciences*, Vol. 10, No. 5, p. 1622, 2020.

[13] F. Rasheed, K. L. A. Yau, and Y. C. Low, "Deep reinforcement learning for traffic signal control under disturbances: A case study on Sunway city, Malaysia", *Future Generation Computer Systems*, Vol. 109, pp. 431-445, 2020.

[14] Z. Li, H. Yu, G. Zhang, S. Dong, and C. Z. Xu, "Network-wide traffic signal control optimization using a multi-agent deep reinforcement learning", *Transportation Research Part C: Emerging Technologies*, Vol. 125, p. 103059, 2021.

[15] R. S. Sutton and A. G. Barto, "Introduction to reinforcement learning", *MIT Press Cambridge*, 1998.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning", *arXiv Preprint arXiv:1312.5602*, 2013.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and G. Ostrovski, "Human-level control through deep reinforcement learning", *Nature*, Vol. 518, No. 7540, pp. 529-533, 2015.

[18] Y. Bengio, "Deep learning of representations: Looking forward", In: *Proc. of International Conf. on Statistical Language and Speech Processing*, pp. 1-37, 2013.

[19] L. Li, Y. L, and F. Y. Wang, "Traffic signal timing via deep reinforcement learning", *IEEE/CAA Journal of Automatica Sinica*, Vol. 3, No. 3, pp. 247-254, 2016.

[20] R. Riccardo and G. Massimiliano, "An empirical analysis of vehicle time headways on rural two-lane two-way roads", *Procedia-Social and Behavioral Sciences*, Vol. 54, pp. 865-874, 2012.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", *arXiv Preprint arXiv:1412.6980*, 2014.

[22] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "Sumo simulation of urban mobility: an overview", In: *Proc. of SIMUL 2011, The Third International Conf. on Advances in System Simulation. ThinkMind*, 2011.