E0-399 : Research in Computer Science Report

Reinforcement Learning Algorithms for Vehicular Traffic Control

Shreya Salmalge
M. Tech CSA

Problem Statement

The aim is to study reinforcement algorithms for maximizing traffic flow in road networks by:

- Adaptive control of traffic lights at junctions
- Rerouting vehicles in the road network

1 INTRODUCTION

Traffic congestion is the root cause of various social and economic problems like increased pollution, increased fuel or energy consumption, and increased travel times. With rising traffic in cities and limited road infrastructure, any attempt to optimize the system's traffic flow would necessitate the intelligent design of traffic signal timing at junctions. To tackle the problem of minimizing traffic congestion, we study two sets of reinforcement learning algorithms. First, we study algorithms that reduce congestion in the road network by adaptively controlling traffic lights at the junction. Second, we study algorithms that minimize traffic by rerouting.

2 ALGORITHMS FOR ADAPTIVE TRAFFIC LIGHT CONTROL

2.1 Reinforcement Learning With Function Approximation for Traffic Signal Control

In [1], the authors propose a Q-learning-based algorithm with function approximation to control traffic lights at junctions. The problem of increasing dimensionality of state space is tackled with feature-based state representation. This algorithm considers a setting of a road network with m junctions, m > 1. Each junction has multiple crossroads, with each road having j lanes. This setting is formulated as MDP.

- **State**
  State is a vector of queue lengths and the elapsed times. $t_i(n)$ is the elapsed time for the red signal on lane i at time n (time since the signal turned red on that lane). $q_i(n)$ is the queue length on lane i at time n. For a network with a total of N signaled lanes, the state at time n is -
  $$s_n = (q_1(n), ..., q_N(n), t_1(n), ..., t_N(n))^T$$
- **Action**
  Centralized controller receives the state information from the various lanes. Controller decides which traffic lights to switch green during a cycle. This decision is then relayed back to the individual junctions.
  Action a_n comprise the sign configuration (feasible combination of traffic lights to switch) in m junctions of the road network.
  $$a_n = (a_1(n), ..., a_m(n))^T$$
  Where $a_i(n)$ is the sign configuration at junction i in time slot n. Action set $A(s_n)$ is given by set of feasible sign configurations in state s_n.

- **Cost**
  The cost function here has two components - the sum of the queue lengths of the individual lanes and the sum of the elapsed times on all lanes. The idea here is to regulate the flow of traffic to minimize the queue lengths while, at the same time, ensuring fairness so that no lane suffers from being red for a long duration. Lanes on the main road are given higher priority over other lanes. Let I_p denote the set of indexes of lanes whose traffic should be given higher priority. The single stage cost is given as :

$$k(s_n, a_n) = \alpha_1 * \left(\sum_{i \in I_p} \alpha_2 * q_i(n) + \sum_{i \notin I_p} \beta_2 * q_i(n) \right) + \beta_1 * \left(\sum_{i \in I_p} \alpha_2 * t_i(n) + \sum_{i \notin I_p} \beta_2 * t_i(n) \right)$$

  Where $\alpha_i, \beta_i >= 0$ and $\alpha_i + \beta_i = 1$ for i=1,2. Also $\alpha_2 > \beta_2$. We consider the infinite-horizon discounted-cost framework.

2.1.1 Q-Learning with full state representation(QTLC-FS)

Q-learning finds the optimal policy without knowledge of the transition probabilities of the underlying MDP. We can start this algorithm by arbitrarily initializing values of all $Q_0(i, a)$ and follow the update rule :

$$Q_{n+1}(i,a) = Q_n(i,a) + a(n) \times \left(k(i,a) + \gamma \min_{b \in A(\eta_n(i,a))} Q_n\left(\eta_n(i,a), b\right) - Q_n(i,a) \right).$$

QTLC-FS requires a lookup table to store the Q-values for every possible (s, a)-tuple.

2.1.2 Q-Learning with Function Approximation(QTLC-FA)

To alleviate this problem of the curse of dimensionality, feature based representation of states is used and Q-value are approximated as follows:

$$Q(s,a) \approx \omega^T \sigma_{s,a} \tag{1}$$

where $\sigma_{s,a}$ is d-dimensional feature (column) vector that corresponds to the state–action tuple (s, a) and ω is tunable parameter whose dimension is the same as in $\sigma_{s,a}$. Let s_n, s_{n+1}, denote the state at instants n and n+1, respectively. n^{th} update of parameter ω is given as :

$$\omega_{n+1} = \omega_n + \alpha(n) \sigma_{s_n, a_n} \times \left(k(s_n, a_n) + \gamma \min_{v \in A(s_{n+1})} \omega_n^T \sigma_{s_{n+1}, v} - \omega_n^T \sigma_{s_n, a_n} \right)$$

Action a_n is chosen according to $a_n = argmin_{v \in A(s_n)} \omega^T \sigma_{s_n, a}$

The features are chosen based on the queue lengths and elapsed times of each signaled lane of the road network. We select features $\sigma(s_n, a_n)$ as:

$$\sigma_{s_n, a_n} = (\sigma_{q_1(n)}, ..., \sigma_{t_1(n)}, \sigma_{t_N(n)}, ..., \sigma_{q_1(n)}, \sigma_{a_1(n)}, ..., \sigma_{a_m(n)})^T \tag{2}$$

where

$$\sigma_{q_i(n)} = \begin{cases} 0, & if\ q_i(n) < L1 \\ 0.5 & if\ L1 \leq q_i(n) \leq L2 \\ 1 & if\ q_i(n) > L2 \end{cases}$$

$$\sigma_{t_i(n)} = \begin{cases} 0, & if\ t_i(n) < T1 \\ 1 & otherwise \end{cases}$$

2.2 Threshold Tuning Using Stochastic Optimization for Graded Signal Control

Idea here is to use suitably chosen threshold levels L1 and L2 to mark congestion as being low, medium or high range. Similarly, threshold T1 for elapsed time. In [2], authors use same definition of state, action and reward as defined in [1]. Methodology in [2] consists of A threshold-based

sign configuration policy obtained from a TLC algorithm for a fixed set of thresholds. Authors also propose another algorithm that operates on top of the TLC algorithm itself for tuning the thresholds.

The sign configuration policy that governs the state evolution is based on given queue-length thresholds L1 and L2 and elapsed-time threshold T1. Let $\theta = (L_1, L_2, T_1)^T$. where L1 and L2 lie between L_{min} and L_{max}, T1 lie between T_{min} and T_{max}. A stochastic iterative algorithm for finding the optimal thresholds in the case of a long-run average cost objective would require two nested loops as follows
1) The inner loop estimates the long-run average cost and also picks actions from the underlying TLC algorithm.
2) The outer loop updates θ along a negative descent direction using an estimate obtained using the outcome of the inner loop procedure. Using a multiple-time-scale stochastic approximation procedure, we run the inner and outer loops in tandem. The resulting scheme is shown to converge to the optimal solution for the long-run average cost objective.

- **Gradient of the objective function**
  The threshold-tuning algorithm estimates the gradient of the objective function using a one-sided SPSA-based estimate which incorporates a Hadamard-matrix-based deterministic construction for the perturbations.

$$\nabla_\theta J(\theta) \approx \left(\frac{J(\theta + \delta\triangle)}{\delta} \right) \triangle^{-1}$$

- **Operation of the threshold-tuning algorithm**
  System is simulated for a perturbed parameter value $(\theta + \delta\Delta)$. Average cost estimates that were obtained through simulation are used to update in the negative gradient descent direction. Updates are done as follows:

$$L_1(n+1) = \pi_1 \left(L_1(n) - a(n) \left(\frac{\tilde{Z}(nL)}{\delta\Delta_1(n)} \right) \right)$$

$$L_2(n+1) = \pi_1 \left(L_2(n) - a(n) \left(\frac{\tilde{Z}(nL)}{\delta\Delta_2(n)} \right) \right)$$

$$T_1(n+1) = \pi_2 \left(T_1(n) - a(n) \left(\frac{\tilde{Z}(nL)}{\delta\Delta_3(n)} \right) \right)$$

  Where $\pi_1(x) = min(max(L_{min}, x), L_{max})$ and $\pi_2(x) = min(max(T_{min}, x), T_{max})$. a(n) is step size.$L1$ is a fixed parameter that controls the rate of update of θ in relation to that of $\tilde{Z}$. $\tilde{Z}(nL)$ represents the cost function averaging term obtained by accumulating the single-stage cost over L cycles. Full threshold tuning algorithm is given in Algorithm 1.

2.2.1 Traffic Light Control Algorithms with threshold tuning

Three TLC algorithms are presented, each corresponding to a given class of graded-threshold-based sign configuration policies with given thresholds.

- **Q-Learning Traffic Light Control With State Aggregation (QTLC-SA-TT):** In this algorithm, states and actions are aggregated as described in equation 2. Update rules for this algorithm paired with tuning algorithm are as follows:

$$\tilde{Z}(m+1) = \tilde{Z}(m) + b(n)(k(\hat{s_m}(\theta), \tilde{a_m}) - \tilde{Z}(m)) \tag{3}$$

$$Q_{m+1}(\hat{s}(\theta), \hat{a}) = Q_m(\hat{s}(\theta), \hat{a}) + b(n) \left(k(\hat{s}, \hat{a}) + \gamma \min_{b \in A(j(\theta))} Q_m(j(\theta), b) - Q_m(\hat{s}(\theta), \hat{a}) \right)$$

- **Q-Learning With Function Approximation With a Novel Feature Selection Scheme (QTLC-FA-NFS-TT):** In this algorithm, Q-values are approximated as described in section

Algorithm 1 Threshold-tuning algorithm

Inputs: R, a large positive integer; θ_0 , initial parameter vector; $\delta > 0$; Δ
UpdateTheta(), the stochastic update rule
$Simulate(\theta) \rightarrow X$: The function that performs one time step of the road traffic simulation and output the single-stage cost value $k(\hat{s_n}, .)$
UpdateAverageCost(): The function that updates the average cost estimated $\tilde{Z}(.)$ and is specific to the TLC-algorithm.
UpdateTheta(): The function that updates the threshold parameter θ
Output: $\theta^* = \theta_R$
$\theta \leftarrow \theta_0, n \leftarrow 1$
repeat
    $\hat{X} \leftarrow Simulate(\theta + \delta\Delta)$
    UpdateAverageCost()
    if $n\%L = 0$ **then**
        UpdateTheta()
    end if
until n==R
return θ_R

2.1.2. in equation 1. Update rules for accumulated single stage cost $\tilde{Z}(m)$ is same as given in equation 3. Update rule for ω is given as follows :
For m = nL, ..., (n+1)L-1

$$\omega_{m+1} = \omega_m + b(n)\left(k(\tilde{s_m}, \hat{a}_m) + \gamma \min_{v \in s_{\tilde{m+1}}} \omega_m^T \sigma_{s_{\tilde{m+1}},v} - \omega_m^T \sigma_{\tilde{s_m},\hat{a}_m}\right)$$

Here, features $\sigma_{s_n,a_n} = (\sigma_1(n), ..., \sigma_k(n))$ where $\sigma_1(n)$ is chosen as per table 1.

State	Action	Feature
$q_i(n) < L_1$ and $t_i(n) < T_1$	RED GREEN	0 1
$q_i(n) < L_1$ and $t_i(n) \geq T_1$	RED GREEN	0.2 0.8
$L_1 \leq q_i(n) < L_2$ and $t_i(n) < T_1$	RED GREEN	0.4 0.6
$L_1 \leq q_i(n) < L_2$ and $t_i(n) \geq T_1$	RED GREEN	0.6 0.4
$q_i(n) \geq L_2$ and $t_i(n) < T_1$	RED GREEN	0.8 0.2
$q_i(n) \geq L_2$ and $t_i(n) \geq T_1$	RED GREEN	1 0

Figure 1: Feature Selection scheme for $\sigma_i(n)$ for lane i

- **PTLC-TT:** this is priority based TLC algorithm. The cost that is assigned to each lane is decided based on queue length and the elapsed time as given in table 2. Update rules for accumulated single stage cost $\tilde{Z}(m)$ is same as given in equation 3.

2.3 Robust Traffic Signal Timing Control using Multi-agent Twin Delayed Deep Deterministic Policy Gradients

Single agent RL methods do not scale efficiently when the action spaces and state spaces expand to higher dimensions. In [3], authors have developed a co-operative multi-agent deep reinforcement learning framework that undertakes and ameliorates traffic congestion. Target policy smoothing, delayed target policy updates and double critics are used to enhance learning stability.

Condition	Priority value
$q_i < L_1$ and $t_i < T_1$	1
$q_i < L_1$ and $t_i \geq T_1$	2
$L_1 \leq q_i(n) < L_2$ and $t_i(n) < T_1$	3
$L_1 \leq q_i(n) < L_2$ and $t_i \geq T_1$	4
$q_i \geq L_2$ and $t_i < T_1$	5
$q_i \geq L_2$ and $t_i \geq T_1$	6

Figure 2: Priority assignment scheme for $\sigma_i(n)$ for lane i

2.3.1 Problem Formulation

A road network $R(n, l)$ is defined where n denotes a collection of M different intersections on the road network given as $\{n_i\}_{i=1,2..,M}$ and l is the set of incoming lanes l_{jk}. Each intersection has a traffic light that executes phases which is controlled by an agent. Phase p from a set of possible phases P_{n_i} for an intersection n_i is executed in a round robin manner for a phase duration d. The phase duration for the green phasesd_g, is learnt by the method. The queue length $q_{l_{n_i}}$ is defined as the number of vehicles that are present in the incoming lanes for the intersection n_i and have a speed of less than 0.1m/s. The vehicle delay $q_{l_{n_i}}$ is defined as the delay faced by the vehicle located at the end of the queue in incoming lane l_{n_i}. This set up is formulated as MDP.

- **State**: each agent observes a local state o_t which is a subset of the global state s_t of the road network. At intersection n_i, local state is given as:

$$o_{t,n_i} := \{q_{t,l_{n_i}}, w_{t,l_{n_i}}\}$$

- **Action**: The decision to be made by each agent is the amount of green time to assign to any phase during a cycle. phase duration d_g is obtained from actor network output a as follows:

$$mid = \left(\frac{d_g^{max} - d_g^{min}}{2}\right) + d_g^{min}$$

$$interval = d_g^{max} - mid$$

$$d_g = mid + (a * interval)$$

- **Action**: Cumulative queue size and cumulative vehicle delay faced due to the set of incoming lanes l_{n_i} at intersection n_i Reward at time t is given as follows where is parameter that indicate importance of vehicle delay w.r.t. to queue lengths.:

$$w_{t,n_i} = \sum_{j=1,j\neq n_i}^{M} w_{t,l_{j,n_i}}, \qquad q_{t,n_i} = \sum_{j=1,j\neq n_i}^{M} q_{t,l_{j,n_i}}$$

$$r_t = -(q_{t,n_i} + c * w_{t,n_i})$$

2.3.2 Multi-Agent Twin Delayed Deep Deterministic Policy gradient method for Traffic control

The road network R has traffic lights as the agents present in the environment, at time t. Each agents see a local state $o_{t,i}$ and perform actions $a_{t,i}$. These actions cause the environment to change and the agents receive a reward $r_{t,i}$ and observe the next local state $o_{t+1,i}$. The rewards act as feedback signals about the actions $a_{t,i}$ taken by the agent i in achieving the global control objective.

Multiagent Twin Delayed Deep Deterministic Policy Gradient is a multiagent actor-critic based algorithm, which uses the feedback from critic (state- action value function) to improve the actor (policy). Overestimation bias is a common problem in reinforcement learning methods where the state- action value function is maximised to learn the optimal policy. This causes the learned policy to become overoptimistic and unstable. Multiagent Deep Deterministic Policy Gradient (MADDPG) suffers from this maximisation bias as it maximises the estimate of a target critic and uses it to learn

the optimal policy. MATD3 mitigates the bias by learning two critics and uses the minimum of their estimates to learn the optimal policy.

The critics are learnt by minimizing the cost function J_{Q_i} as below:

$$y_{Q_i} = r_i + \gamma * min_{j=1,2} \left\{ Q'_{i,j}(x, a_1, ..., a_N) \right\}$$
$$J_{Q_i} = E[[y_{Q_i} - Q_i(x, a_1, ..., a_N)]^2]$$

Each agent learns two centralized critics that use the actions of all the agents in the network and predict a state-action value function that describes how good the action of its agent was, given its knowledge about the actions of all the other agents. The actor uses the state-value function as feedback on its predictions. Actors are updated as

$$\nabla_{\theta_i} J(\mu_i) = E[\nabla_{\theta_i} \mu_i(a_i|o_i) \nabla_{a_i} Q_{1,i}(x, a_1, ..., a_N)]$$

Multi-agent Twin Delayed Deep Deterministic Policy Gradients algorithm is given in Algorithm 2.

Algorithm 2 Multi-agent Twin Delayed Deep Deterministic Policy Gradients algorithm

Inputs: off-policy RL algorithm A_l; instruction relabeling strategy S; language supervisor Ω; Environment E; number of relabeled future K
Initialize replay buffer B and $\pi_l(a|s, g)$
for $episode = 1$ **to** E **do**
    Initialize empty experience replay buffer D and network parameters
    Reset the environment, OU Noise N and obtain initial observation o_0
    for step $t = 1$ **to** $max_episode_length$ **do**
        Select action $a_i \sim \mu(o_i) + N_t$
        convert a_i to d_{g_i}
        execute action $(d_{g_1}, d_{g_2}, ..., d_{g_M})$ and observe $r_{t,i}, o_{t+1}$
        store $(o_t, a_1, ..., a_M, , \tau_1, .., \tau_M, o_{t+1}$ in D
        $o_t = o_{t+1}$
        for step agent $i = 1$ **to** M **do**
            Sample a mini-batch of size S
            sample $(o_t^b, a^b, r^b, o_{t+2}^b$ from D
            Set $y^b = r_i^b + \gamma min_{j=1,2} Q_{i,j}^{\mu'}(o_t^j, a_1, ..., a_M)$
            Minimize critic loss for both j= 1,2 $L(\theta_j) = \frac{1}{s} \sum_b (Q'_{i,j}(o_{t+1}^b, a_1, ..., a_M) - y^b)^2$
            if $t\%d = 0$ **then**
                Update policy μ with gradient
                $\nabla_{\theta_{\mu,i}} = \frac{1}{s} \sum_b \nabla_\theta \mu_i(o_{t,i}^b) * (\nabla_{a_i} Q_{i,1}(o_{t+}^b, a_1, ..., a_M))$
                Update target network parameters as
                $\theta_{Q_{1,i'}} = \tau \theta_{Q_{1,i}} + (1 - \tau) \theta_{Q_{1,i'}}$
                $\theta_{Q_{2,i'}} = \tau \theta_{Q_{2,i}} + (1 - \tau) \theta_{Q_{2,i'}}$
                $\theta_{\mu_{i'}} = \tau \theta_{\mu_i} + (1 - \tau) \theta_{\mu_{i'}}$
            end if
        end for
    end for
end for

3 ALGORITHMS FOR ROUTING

We study algorithm that deal with routing either in road network or in computer network. With modification, ideas from computer network routing algorithms can be applied to vehicular routing.

3.1 Predictive Q-Routing: A Memory-based Reinforcement Learning Approach to Adaptive Traffic Control

In [4], authors propose a memory-based Q-Learning algorithm called predictive Q-routing (PQ-routing) for adaptive traffic control. The adaptive traffic control problem is to devise routing policies

for controllers (i.e. routers) operating in a non-stationary environment to minimize the average packet delivery time. Packet Routing Policy decides which adjacent node should the current node send its packet to get it as quickly as possible to its eventual destination. Policy's performance is measured by the total time taken to deliver a packet.

Environment i.e network is non-stationary. Therefore the optimal policy varies with time as a result of changes in network traffic and topology. This policy is learned by agents by interacting with the environment. We assume that only local communication between controllers is allowed. The controllers must cooperate among themselves to achieve the common, global objective.

Idea of Predictive Q-learning is as follows. Under low network load, the optimal policy is simply the shortest path routing policy. When load level increases, instead of sending packets on shortest paths, packets can be send to less congested path. When network load drops again, congested paths are not used for a period of time, they will recover and become good candidates again. Thus shortest paths are frequently explored in controlled manner. This exploration is called probing.

Predictive Q-Learning algorithm is as follows:

TABLES:
$Q_x(d,y)$ - estimated delivery time from node x to node d via neighboring node y
$B_x(d,y)$ - best estimated delivery time from node x to node d via neighboring node y
$R_x(d,y)$ - recovery rate for path from node x to node d via neighboring node y
$U_x(d,y)$ - last update time for path from node x to node d via neighboring node y

TABLE UPDATES: (after a packet arrives at node y from node x)
ΔQ = (transmission delay + queueing time at y + $\min_z\{Q_y(d,z)\}$) $- Q_x(d,y)$
$Q_x(d,y) \leftarrow Q_x(d,y) + \alpha\,\Delta Q$
$B_x(d,y) \leftarrow \min(B_x(d,y), Q_x(d,y))$
if ($\Delta Q < 0$) then
  $\Delta R \leftarrow \Delta Q$ / (current time $- U_x(d,y)$)
  $R_x(d,y) \leftarrow R_x(d,y) + \beta\,\Delta R$
else if ($\Delta Q > 0$) then
  $R_x(d,y) \leftarrow \gamma\, R_x(d,y)$
end if
$U_x(d,y) \leftarrow$ current time

ROUTING POLICY: (packet is sent from node x to node y)
Δt = current time $- U_x(d,y)$
$Q'_x(d,y) = \max(Q_x(d,y) + \Delta t\, R_x(d,y), B_x(d,y))$
$y \leftarrow \arg\min_y\{Q'_x(d,y)\}$

Instead of selecting actions based solely on the current Q-values, the recovery rates are used to yield better estimates of the Q-values before the minimum selector is applied. Learning rate for Q-function α is kept 1. Parameter β=0.7, is used for learning recovery rate. Parameter γ= 0.9, is used for controlling decay of recovery rate(affects probing frequency).

3.2 Reinforcement Learning for Adaptive Routing

In [5], Authors present an application of gradient ascent algorithm for reinforcement learning to a complex domain of packet routing in network communication. Agents i.e Routers makes routing decision according to individual policy based on its local observation. Network acts as environment. Every agent's local observation consists of destination of packets and status of links. Agents decide where to send packet according to stochastic policy. Agents receive reward signal distributed through the network by acknowledgement packets. This reward depends on total delivery time. Parameters of policy are adjusted according to some measure of the global performance of the network. We work in a setting where nodes do not have any information regarding the topology of network or their position in it.

For a single agent and its interaction with partially observable MDP, agent's policy is stored with the help of look up table with θ_{oa} values. θ_{oa} indicates value for each observation-action(destination/link) pair. Policy is stochastic, that is - it returns a probability distribution over possible actions to be taken. Probability of taking an action a given observation o is given as :

$$\mu(a,o,\theta) = Pr(a(t) = a|o(t) = o,\theta)) = \frac{exp(\theta_{oa}/\Xi)}{\sum_{a'} exp(\theta_{oa'}/\Xi)}$$

H_t is set of of all the possible experience sequences of length t. experience h is given by $\langle o(1), a(1), r(1), ..., o(t), a(t), r(t), o(t+1)\rangle$. At time τ, $r(\tau, h)$ and $a(\tau, h)$ are τ^{th} reward and action respectively. h^{τ} is prefix of H_t given as $\langle o(1), a(1), r(1), ..., o(\tau), a(\tau), r(\tau), o(\tau+1)\rangle$.

Value of following policy μ with parameter θ is expected cumulative discounted reward is given as :

$$V(\theta) = \sum_{t=1}^{\infty} \gamma^t \sum_{h \in H_t} Pr(h|\theta) r(t, h)$$

Derivative for each weight θ_{oa}

$$\frac{\partial V(\theta)}{\partial \theta_{oa}} = \sum_{t=1}^{\infty} \gamma^t \sum_{h \in H_t} Pr(h|\theta) r(t, h) \times \sum_{\tau=1}^{t} \frac{\partial ln Pr(a(\tau, h)|h^{\tau-1}, \theta)}{\partial \theta_{oa}}$$

To run algorithm, we sample from distribution of histories by interacting with the environment. Then we calculate, during each trial an estimate of the gradient, accumulating the quantities (at time t):

$$\gamma^t r(t, h) \sum_{\tau=1}^{t} \frac{\partial ln \mu(a, o, \theta)}{\partial \theta_{oa}}$$

Under chosen policy, we get:

$$\frac{\partial ln \mu(a, o, \theta)}{\partial \theta_{o'a'}} = \begin{cases} 0 & if o' \neq o \\ -\frac{1}{\Xi} \mu(a', o, \theta) & if o' = o, a' \neq a \\ \frac{1}{\Xi}[1 - \mu(a, o, \theta)] & if o' = o, a' = a \end{cases}$$

Applying this algorithm in a network of connected controllers constitutes the algorithm of routing by distributed Gradient Ascent Policy Search (GAPS).

3.3 Multi-Agent Reinforcement Learning for Markov Routing Games

[6] aims to develop a paradigm that models the learning behavior of intelligent agents (including but not limited to autonomous vehicles, connected and automated vehicles, or human-driven vehicles with intelligent navigation systems where human drivers follow the navigation instructions completely) with a utility-optimizing goal and the system's equilibrating processes in a routing game among atomic selfish agents. At each junction, agents decide what next link to take to optimize a prescribed objective function.

3.3.1 Problem Formulation

There are N intelligent agents indexed by $i \in \{1, 2, .., N\}$. Each agent is moving from origin $o_i \in O$ to destination $d_i \in D$. Each agent aims to select its optimal next-go-to edge at every intermediate node by minimizing a travel cost functional over a predefined planning horizon [0,T]. System evolves according to all agents' actions and some transition probability at each time step. Process continues till either all agents reach the destination, or the planning time horizon is terminated. When one solves its own optimal control problem while others are doing so simultaneously, a routing game is formed. System evolution dynamic remains unknown to these agents. Agents have to learn this game and select their individual routing strategy.

- **Markov Routing Game** We denote the Markov game by Partially observable Markov decision process:

  $$(S, O_1, O_2, ..., O_1, A_1, A_2, ..., A_N, P, R_1, R_2, ..., R_N, N, \gamma)$$

  There are N controllable adaptive agents in environment. Environment state $s \in S$ is distribution of agents and traffic condition on each link. State s is not fully observable to agents. Agents have their private observation $o_i = (n, t)$ where n is node and t is time. Based on o_i, agents take action a_i which is choosing an outbound link from node n. Joint action of all agents triggers a state transition $s \to s'$ with probability $p(s'|s, a)$. At time t,

agent i receives reward $r_{i,t'}(s, a, s')$ which is some negative travel cost such as travel time or travel distance.

Agents $i \in \{1, 2, ..., N\}$ uses a policy π_i to choose actions after drawing observation o_i. Agent i aims to maximize its discounted expected cumulative reward ρ_i by deriving an optimal policy μ_i^* where

$$\rho_i = \sum_{t=0}^{T} \gamma^t r_{i,t}$$

Q-value function for agent i, Q_i, is dependent on the environment state $s \in S$ and the joint action $a \in A$ of all agents

$$Q_i = Q_i(s, a)$$

Optimal policy of agent i, μ_i^* is the i_{th} agent's best response to others' policies when others hold their policies constant.

- **Nash Equilibrium of Markov Routing Game** A set of policies $\mu_1, \mu_2, ..., \mu_N$ is a Nash equilibrium if each is a best response to the others. Nobody can improve her value function by unilaterally switching her policy, while all others hold their policies fixed. That is, $\forall i \in \{1, ..., N\}$, the value attained by agent i from any state s is

$$V_i(s) = max_{ai} E_{a^{-i} \sim A} E_{s' \sim P(.|s,a)}[r_i(s, a, s') + \gamma V_i(s')]$$

At a Nash equilibrium, each agent maximizes its value function given that all other agents remain fixed. Every Markov game has a Nash equilibrium in stationary policies [7]. Markov strategy is the policy that depends only on the current state. If all players other than i play Markov strategies, then player i has a best response that is a Markov strategy.Deriving a Nash equilibrium in general requires a perfect knowledge of the system evolution dynamics and the rewards of the game. However, multi-agent RL is a computational tool for Markov games that models individual agents' routing behavior with a partial or unknown information structure and a stochastic model-free environment.

- **RL in Markov Routing Game** For a real-world multi-agent problem, there are commonly more than tens of thousands of agents. We broadly categorize agents into C groups and enable Q function sharing within each group. We denote the shared Q function for agents in group $c \in \{1, 2, ..., C\}$ as

$$Q_c(o_i, o_{-i}, a_i, a_{-i})$$

But dimension of input to Q function increases as number of agents increase. In the task of route choice, observations and actions of agents who are currently far away from agent i have a very limited influence on agent i. Therefore, high-dimensional o_{-i} and a_{-i} could be approximated by some low-dimensional aggregate information of nearby agents.

$$Q_c(o_i, o_{-i}, a_i, a_{-i}) \approx Q_c(o_i, o_{-i}, \tilde{a}_i)$$

Definition 3.1 (Mean Action) *Mean action $\tilde{a}_i$ is defined as the traffic flow on the link that is chosen by agent i. Note that the traffic flow is calculated right after agent i enters the link.*

- **Mean field multi-agent deep Q-learning** A DQN parameterized by θ_c , denoted by $Q_c(o_i, o_{-i}, \tilde{a}_i|\theta_c)$, is used to approximate $Q_c(o_i, o_{-i}, \tilde{a}_i)$. Agents in group $c \in \{1, 2, ..., C\}$ with not only the environment but also other agents and collects experience tuples in the form of $(o_i, a_i, o_i', r_i, \tilde{a}_i)$. DQN updates its parameter θ_c by minimizing the following loss:

$$L(\theta_c) = E_{o_i, a_i, o_i', \tilde{a}_i}[(r_i + \gamma max_{a_i'} E_{\tilde{a}_i \sim \mu_{-i}^*}[Q_c(o_i, o_{-i}, \tilde{a}_i|\theta_c^-)] - Q_c(o_i, o_{-i}, \tilde{a}_i|\theta_c))^2]$$

where $Q_c(.|\theta_c^-)$ is a target network copied from $Q_c(.|\theta_c)$ every τ episodes to stabilize training. After updating Q function, optimal policy is :

$$\mu_i^*(o_i) = argmax_{a_i} E_{\tilde{a}_i \sim \mu_{-i}^*}[Q_c(o_i, o_{-i}, \tilde{a}_i|\theta_c)], \forall o_i \in O_i$$

The mean field multi-agent deep Q-learning (MF-MA-DQL) algorithm is summarized in Algorithm.

Algorithm 3 Mean field multi-agent deep Q-learning (MF-MA-DQL)

Inputs: exploration parameter $\varepsilon = \varepsilon_0, learningrate\eta = \eta 0, targetnetworkupdateperiod\tau$
Initialize one DQN $Q_c(o_i, o_{-i}, \tilde{a}_i|\theta_c)$, parameterized by θ_c, and one target network $Q_c(o_i, o_{-i}, \tilde{a}_i|\theta_c^-))$ for each group $c \in \{1, 2, ..., C\}$
Initialize N dictionaries to store the optimal policy for agents, i.e., $\mu_i^*, \forall i \in \{1, 2, ..., N\}$
Initialize one experience replay buffer B_c for each group $c \in \{1, 2, ..., C\}$
$episodee = 0$
repeat
    From the initial environmental states_0 , each agent i draws observation o_i
    t = 0
    repeat
        For each agent i, select action a_i according to the ϵ-greedy method
        Execute joint action $a = (a_1, a_2, ..., a_N)$ in the environment to trigger state transition $s \to s'$
        Each agent i draws new observation o_t', receives reward r_t, , and observes mean action $\tilde{a}_i$
        Store experience tuple $(o_i, a_i, o_i', r_i, \tilde{a}_i)$ into replay buffer B_c if agent i belongs to group c
        $i \leftarrow i + 1$
    until t = T
    for step $c = 1$ **to** C **do**
        for step $j = 1$ **to** J_c **do**
            Sample a mini-batch of size K_c from replay buffer B_c
            Update parameter θ_c of Q_c by minimizing the loss
            Update optimal policy μ_i^* of agent i
        end for
    end for
    $episode = episode + 1$
    Decrease the exploration parameter ϵ
    Decrease the learning rate η
    if $episode\%\tau = 0$ **then**
        for step $c = 1$ **to** C **do**
            Update parameter θ_c^- of the target network
        end for
    end if
until the algorithm converges
Return optimal policies $\mu_i^*, \forall i \in \{1, 2, ..., N\}$

4 Experiments

Experiments are carried out to adaptively control traffic lights with Deep Q-Networks(DQN). We apply variations of DQN in traffic light control algorithm. We considers a setting of a road network with m junctions, m > 1. Each junction has multiple crossroads, with each road having j lanes. This setting is formulated as MDP. For these experiments, state, action and reward are defined as follows:

- **State**
  We apply state aggregation to obtain state vector σ_{s_n} from queue lengths and the elapsed times as following.

$$\sigma_{s_n} = (\sigma_{q_1(n)}, ..., \sigma_{q_N(n)}, \sigma_{t_1(n)}, ..., \sigma_{t_N(n)})^T$$

  where

$$\sigma_{q_i(n)} = \begin{cases} 0, & if\ q_i(n) < L1 \\ 0.5 & if\ L1 \le q_i(n) \le L2 \\ 1 & if\ q_i(n) > L2 \end{cases}$$

$$\sigma_{t_i(n)} = \begin{cases} 0, & if\ t_i(n) < T1 \\ 1 & otherwise \end{cases}$$

  here, $q_i(n)$ and $t_i(n)$ are queue length and elapsed time in lane i at time n. L1,L2,T1 are either hyperparameters or tuned by algorithm itself.

- **Action**

  Action a_n comprise the sign configuration (feasible combination of traffic lights to switch) in m junctions of the road network.

  $$a_n = (a_1(n), ..., a_m(n))^T$$

  Where $a_i(n)$ is the sign configuration at junction i in time slot n. Action set $A(s_n)$ is given by set of feasible sign configurations in state s_n.

  We consider every intersection, 4 actions i.e phases are possible.

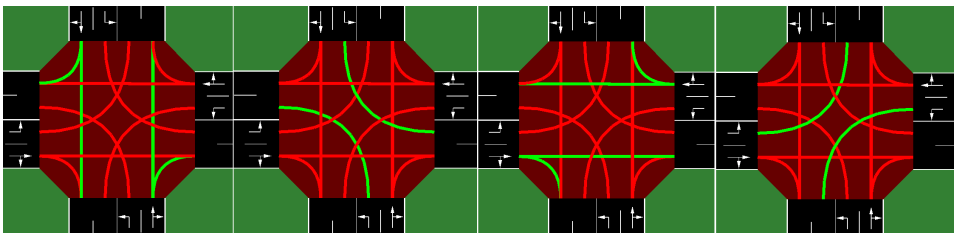

Figure 3: Possible phases at every intersection

- **Cost**

  The cost function here has two components - the sum of the queue lengths of the individual lanes and the sum of the elapsed times on all lanes.

  $$k(s_n, a_n) = \alpha_1 * \sum_i q_i(n) + \beta_1 * \sum_i t_i(n)$$

  Where $\alpha_i = 0.5, \beta_i = 0.5$.

All experiments are carried out in SUMO. Using TRACI api we can create a road network, define routes of vehicles and their arrival time, simulate time steps in SUMO to collect experience.

4.1 Traffic light control with Deep Q-Network

Here, we use DQN Q_ω to approximate Q-function where ω are network weights. Algorithm used to train the model is given as following:

Algorithm 4 Traffic light control with Deep Q-Network

Initialize replay buffer B and DQN Q_ω
for $episode = 1$ **to** $max\_episodes$ **do**
    Initialize empty experience replay buffer B
    Reset the environment
    for step $t = 1$ **to** $max\_episode\_length$ **do**
        Select action a_t according to ϵ-greedy strategy
        execute action a_t and observe r_t, s_{t+1}
        store $(s_t, a_t, r_t, s_{t+1}$ in B
        $s_t = s_{t+1}$
    end for
    for step agent $i = 1$ **to** max_epochs **do**
        Sample a mini-batch of size S
        Set $y_i = r_t + \gamma min_{a'} Q_\omega(s_{t+1}, a')$ for $i = (s_t, a_t, r_t, s_{t+1}$ in batch
        update ω by minimizing loss $L(\omega_j) = \frac{1}{s} \sum_i (Q_\omega(s_t, a_t) - y_i)^2$
    end for
end for

Resultant average reward, average queue lengths and average waiting times of vehicles are presented in figure 4 and 5.

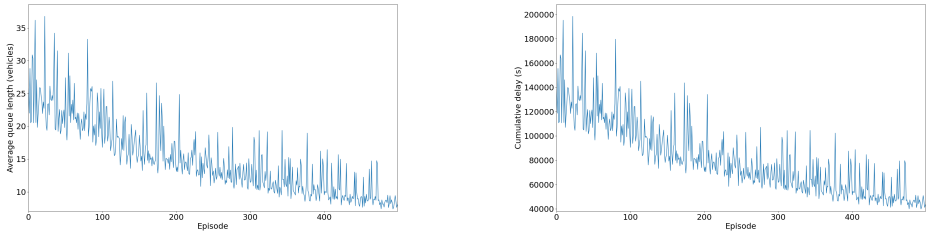

Figure 4: Average Queue lengths and cumulative delay obtained while training on Deep Q-Network for 500 episodes

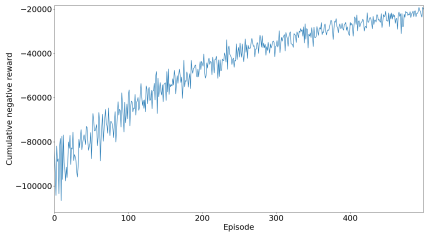

Figure 5: Cumulative reward obtained while training on Deep Q-Network for 500 episodes

4.2 Traffic light control with Deep Q-Network along with threshold tuning

As described above, we use DQN Q_ω to approximate Q-function. In addition, we have parameters $\theta = [L_1, L_2, T_1]$ which are used to aggregate the states. These parameters are tuned according to Threshold tuning algorithm mentioned in section2.2. Full algorithm used to train the model is given as following:

Algorithm 5 Traffic light control with Deep Q-Network along with threshold tuning

Initialize replay buffer B and DQN Q_ω
for $episode = 1$ **to** $max\_episodes$ **do**
    Initialize empty experience replay buffer $B, n = 0, Z = 0, a_0, b_0$
    Reset the environment
    for step $t = 1$ **to** $max\_episode\_length$ **do**
        Select action a_t according to ϵ-greedy strategy
        execute action a_t and observe r_t, s_{t+1}
        store (s_t, a_t, r_t, s_{t+1}) in B
        $s_t = s_{t+1}$
    end for
    for step agent $i = 1$ **to** max_epochs **do**
        Sample a mini-batch of size S
        Set $y_i = r_t + \gamma min_{a'} Q_\omega(s_{t+1}, a')$ for $i = (s_t, a_t, r_t, s_{t+1})$ in batch
        update ω by minimizing loss $L(\omega j) = \frac{1}{s}\sum_i (Q_\omega(s_t, a_t) - y_i)^2$
        $batch\_cost = -\sum_i r_t$
        $a_n = a_0/n$
        $b_n = b_0/n^\alpha$
        $Z = Z + b_n * (\frac{batch\_cost}{batchSize} - Z)$
        updateTheta() ▷ according to equations in section 2.2
        n = n + 1
    end for
end for

Resultant average reward, average queue lengths and average waiting times of vehicles are presented in figure 6 and 7.

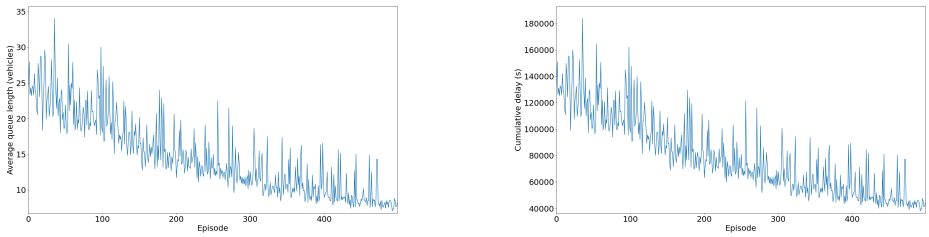

Figure 6: Average Queue lengths and cumulative delay obtained while training on Deep Q-Network with threshold tuning for 500 episodes

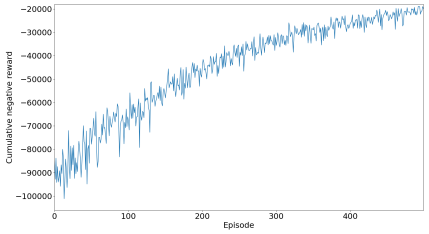

Figure 7: Cumulative reward obtained while training on Deep Q-Network with threshold tuning for 500 episodes

4.3 Traffic light control with Double Deep Q-Network

Here, we use DQN Q_ω to approximate Q-function where ω are network weights. We maintain a copy of above Q-Network in target DQN $Q_{\omega'}$ to calculate target values i.e y_i. This modification is done to improve stability while training DQN. Target network is periodically updated after fixed number of iterations d. Algorithm used to train the model is given as following:

Algorithm 6 Traffic light control with Double Deep Q-Network

Initialize replay buffer B and DQN Q_ω, target network $Q_{\omega'}$, n=0, d
for $episode = 1$ **to** $max\_episodes$ **do**
    Initialize empty experience replay buffer B
    Reset the environment
    for step $t = 1$ **to** $max\_episode\_length$ **do**
        Select action a_t according to ϵ-greedy strategy
        execute action a_t and observe r_t, s_{t+1}
        store (s_t, a_t, r_t, s_{t+1}) in B
        $s_t = s_{t+1}$
    end for
    for step agent $i = 1$ **to** max_epochs **do**
        Sample a mini-batch of size S
        Set $y_i = r_t + \gamma min_{a'} Q_{\omega'}(s_{t+1}, a')$ for $i = (s_t, a_t, r_t, s_{t+1})$ in batch
        update ω by minimizing loss $L(\omega_j) = \frac{1}{s}\sum_i (Q_\omega(s_t, a_t) - y_i)^2$
        if $n\%d = 0$ **then**
            Update target network parameters as $\omega' = \omega$
        end if
        n = n + 1
    end for
end for

Resultant average reward, average queue lengths and average waiting times of vehicles are presented in figure 8 and 9.

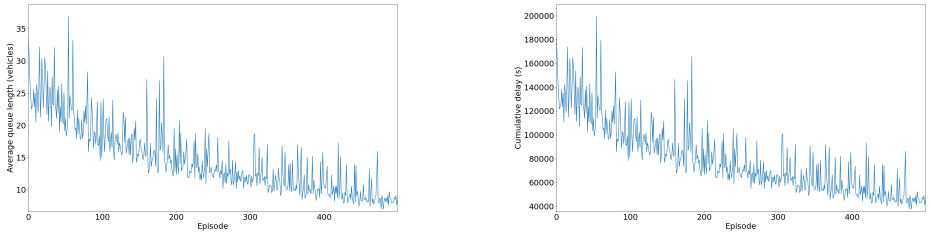

Figure 8: Average Queue lengths and cumulative delay obtained while training on Double Deep Q-Network for 500 episodes

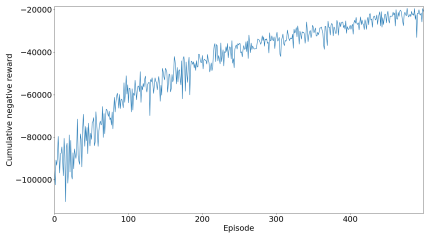

Figure 9: Cumulative reward obtained while training on Double Deep Q-Network for 500 episodes

4.4 Traffic light control with SOR Deep Q-Network

We use DQN Q_θ to approximate Q-function where θ are network weights. We also maintain a copy of above Q-Network in target DQN $Q_{\theta'}$ to calculate target values as done in Double DQN. Target network is periodically updated after fixed number of iterations d. This method proposes a simple modification in the target Q-values to speed up the DQN algorithm. Here we calculate target values as follows : for $i = (s_t, a_t, r_t, s_{t+1}$ in batch

$$y_i = w\left(r_t + \gamma \underset{b \in A}{max} Q_{\theta'}(s_{t+1}, b) + (1 - w) * \underset{c \in A}{max} Q_{\theta'}(s_t, c)\right)$$

Algorithm used to train the model is given as following:

Resultant average reward, average queue lengths and average waiting times of vehicles are presented in figure 10 and 11.

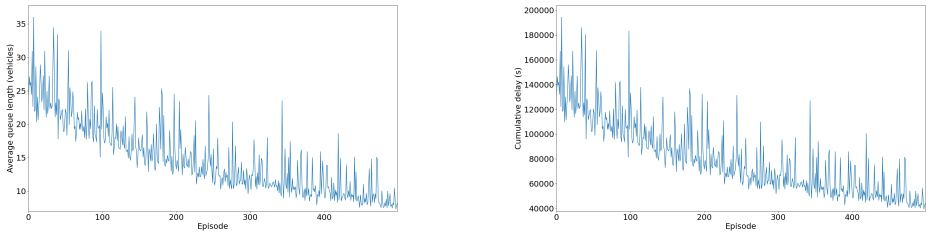

Figure 10: Average Queue lengths and cumulative delay obtained while training on SOR-Deep Q-Network for 500 episodes

Algorithm 7 Traffic light control with SOR Deep Q-Network

Initialize replay buffer B and DQN Q_θ, target network $Q_{\theta'}$, n=0, d
for $episode = 1$ **to** $max\_episodes$ **do**
    Initialize empty experience replay buffer B
    Reset the environment
    for step $t = 1$ **to** $max\_episode\_length$ **do**
        Select action a_t according to ϵ-greedy strategy
        execute action a_t and observe r_t, s_{t+1}
        store (s_t, a_t, r_t, s_{t+1}) in B
        $s_t = s_{t+1}$
    end for
    for step agent $i = 1$ **to** max_epochs **do**
        Sample a mini-batch of size S
        Set $y_i = w\left(r_t + \gamma \underset{b \in A}{max} Q_{\theta'}(s_{t+1}, b) + (1-w) * \underset{c \in A}{max} Q_{\theta'}(s_t, c)\right)$ for $i = (s_t, a_t, r_t, s_{t+1})$ in batch
        update θ by minimizing loss $L(\theta_j) = \frac{1}{s}\sum_i (Q_\theta(s_t, a_t) - y_i)^2$
        if $n\%d = 0$ **then**
            Update target network parameters as $\theta' = \theta$
        end if
        n = n + 1
    end for
end for

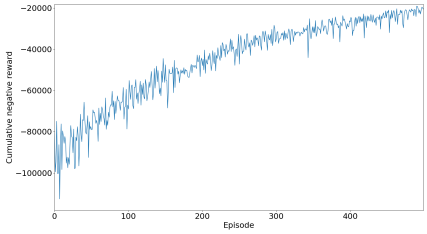

Figure 11: Cumulative reward obtained while training on SOR-Deep Q-Network for 500 episodes

5 Conclusion

Reinforcement learning presents interesting paradigms for traffic light control and routing problems. Algorithms presented in this report either deal with traffic light control or routing but not both simultaneously. Even though these algorithms work well, there is scope for improvement. A good traffic management system should take into account the interdependence of signal control and routing. Developing a combined algorithm that routes vehicles on congested lanes and accordingly also changes traffic signals at junctions can be considered as the future direction.

References

[1] P. LA and S. Bhatnagar, "Reinforcement Learning With Function Approximation for Traffic Signal Control," in IEEE Transactions on Intelligent Transportation Systems, vol. 12, no. 2, pp. 412-421, June 2011, doi: 10.1109/TITS.2010.2091408.

[2] P. L. A. and S. Bhatnagar, "Threshold Tuning Using Stochastic Optimization for Graded Signal Control," in IEEE Transactions on Vehicular Technology, vol. 61, no. 9, pp. 3865-3880, Nov. 2012, doi: 10.1109/TVT.2012.2209904.

[3] Shanmugasundaram, P. and Bhatnagar, S. (2022). Robust Traffic Signal Timing Control using Multiagent Twin Delayed Deep Deterministic Policy Gradients. In Proceedings of the 14th International Conference on Agents and Artificial Intelligence - Volume 2

[4] Choi, Samuel P. M. and Dit-Yan Yeung. “Predictive Q-Routing: A Memory-based Reinforcement Learning Approach to Adaptive Traffic Control.” NIPS (1995).

[5] L. Peshkin and V. Savova, "Reinforcement learning for adaptive routing," Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No.02CH37290), 2002, pp. 1825-1830 vol.2, doi: 10.1109/IJCNN.2002.1007796.

[6] Zhenyu Shou, Xu Chen, Yongjie Fu, Xuan Di, Multi-agent reinforcement learning for Markov routing games: A new modeling paradigm for dynamic traffic assignment, Transportation Research Part C: Emerging Technologies, Volume 137, 2022, 103560, ISSN 0968-090X,

[7] Filar, J., Vrieze, K., 1997. Applications and special classes of stochastic games. In: Competitive Markov Decision Processes. Springer, pp. 301–341.

[8] I. John and S. Bhatnagar, "Deep Reinforcement Learning with Successive Over-Relaxation and its Application in Autoscaling Cloud Resources," 2020 International Joint Conference on Neural Networks (IJCNN), 2020, pp. 1-6, doi: 10.1109/IJCNN48605.2020.9206598.