Robust Deep Reinforcement Learning for Task Scheduling in MEC-Enabled IoT Networks

Vikky Masih, Arghyadip Roy
Mehta Family School of Data Science and Artificial Intelligence, Indian Institute of Technology Guwahati, India
{m.vikky, arghyadip}@iitg.ac.in

***Abstract*—Robust Reinforcement Learning (RL) based task scheduling approaches can address the inherent tradeoff between energy consumption and deadline violation in a Multi-access Edge Computing (MEC) based Internet of Things (IoT) network, while maintaining robustness against changes in the task arrival rate. However, tabular robust RL algorithms suffer from high computational and storage complexity, and therefore are not scalable to a system with a large number of IoT nodes. To this end, in this paper, we propose a robust deep RL based task scheduling algorithm to solve the underlying Robust-Return Constrained Markov Decision Process (R2CMDP) problem. The proposed algorithm introduces a tunable amount of robustness in the solution of the RL framework. Complexity analysis and ns-3 simulation results are presented to demonstrate the efficacy of our algorithm.**

***Index Terms*—MEC, IoT, Robust RL, Deep RL.**

I. Introduction

Various new use cases are being enabled by the next-generation wireless and cellular Internet of Things (IoT) standardization efforts from organizations such as European Telecommunications Standards Institute (ETSI) and Third Generation Partnership Project (3GPP) [1]. With the introduction of Multi-access Edge Computing (MEC) technology, various computationally expensive applications have become feasible for implementation in systems with resource-constrained IoT nodes (devices). The MEC technology brings managed computing services closer to the IoT nodes, thereby reducing processing delay and internet backhaul congestion, compared to cloud computing [2].

Consider a scenario where multiple IoT nodes receive tasks following a stochastic arrival process. Whenever a new task arrives in the system, the centralized controller at the MEC server needs to decide whether to process the task locally on the parent IoT node or at the MEC server. The trade-off between power consumption and deadline violation in task scheduling has been studied in the literature, e.g., in [3], the underlying Markov Decision Process (MDP) problem is solved using Reinforcement Learning (RL) based methods to find the optimal task scheduling policy. Solving the scheduling problem using Dynamic Programming (DP) requires prior knowledge of the state transition probabilities of the underlying MDP, and therefore, unlike RL, the DP methods suffer from the *curse of modeling* [4].

Another practical challenge in obtaining an optimal task scheduling policy is the possibility of model mis-specification, which may occur due to perturbations in the underlying MDP model parameters, such as changes in task arrival rate [5]. A traditional RL based task scheduling policy will exhibit higher sensitivity to such model mis-specifications and hence is not robust [6]. For example, a traditional RL based task scheduling policy learned for a particular task arrival rate will perform poorly when the task arrival rate changes. The only works addressing this issue using robust formulations are [6] and [7]. Both [6] and [7] consider the transition probabilities of the underlying MDP to lie within an uncertainty set. Authors of [7] develop robust DP based algorithms that suffer from the curse of modeling. The authors of [6] propose tabular robust RL algorithms that are free from such issues. The algorithms proposed in [6] aim to solve the underlying Robust Return Constrained Markov Decision Process (R2CMDP) problem. However, the storage and computational complexities of these algorithms are high.

Works such as [8]–[10] consider Deep RL (DRL) techniques that are scalable to a large number of IoT nodes, but there is no consideration of robustness. This motivates us to propose a DRL based robust task scheduling algorithm that can provide better scalability when the number of IoT nodes (and hence, the number of states) is large. In DRL, value functions of states are approximated by the usage of neural networks [4], enabling us to work in a parameter space whose dimensions are much lower than the cardinality of the state space. In this work, we integrate elements of DRL within the framework of robust RL based task scheduling [6] that can be more efficient than tabular robust RL in terms of storage and computational complexities. Our contributions are as follows:

- A novel robust DRL algorithm based on R2CMDP formulation is proposed for task scheduling in an MEC-enabled IoT network.
- Complexity analysis of the proposed algorithm is presented.
- The performance of the proposed algorithm is evaluated via extensive simulations in ns-3.

To the best of our knowledge, this is the first work that proposes a scalable task scheduling algorithm in MEC-enabled IoT systems with a large number of IoT nodes, while also providing a variable amount of robustness to model mis-specifications (such as varying task arrival rate).

II. System Model

Our system consists of multiple IoT nodes connected to a wireless Base Station (BS) (See Fig. 1). An MEC server

is connected to the BS using a lossless link. Newly arriving tasks at the IoT nodes can be processed locally at the parent device or can be offloaded to the MEC server for execution. It is assumed that these tasks arrive at the IoT nodes following a Poisson process with rate parameter λ. Each task has an exponentially distributed mean service time of $1/\mu$. The system model is similar to [6], [11], and therefore λ is not fixed and can vary within an uncertainty set as described in later sections. Note that the model can easily be extended to a system with multiple BSs by modifying the state space.

Fig. 1. An MEC-IoT System.

A. State & Action Space

A controlled random process in the continuous time domain is used to model the system over a finite state-space ($\mathcal{S}$, say). We assume that the wireless channel condition between the BS and the IoT nodes can be classified into two categories, termed good and bad channels. Let q_L be the number of tasks being processed locally at the parent IoT node, $q_G(q_B)$ be the number of tasks being processed at the MEC server with good (bad) channel conditions. The state is then defined as the 4-tuple (q_L, q_G, q_B, z), where $z \in \{Z_1, Z_2, Z_3, Z_4, Z_5\}$ is the event that triggers task scheduling. $Z_1(Z_2)$ represents task arrival at the IoT node with good (bad) channel conditions w.r.t. the BS. Z_3 represent completion of a task at one of the IoT nodes. Similarly, $Z_4(Z_5)$ represents completion of a task at the MEC server that experienced good (bad) channel conditions.

The action space is defined as $\mathcal{A} = \{A_0, A_1, A_2\}$. Action A_0 represents doing nothing. Action A_1 (A_2) represents processing the task at the parent IoT node (MEC server). Whenever an event is triggered, the centralized scheduler at the BS needs to choose an action $a \in \mathcal{A}$ for task scheduling. For new task arrivals, action A_1 or A_2 is feasible only if $q_L < N_1$ and $(q_G + q_B) < N_2$, where N_1 and N_2 are large positive integers. Action A_0 is chosen in the case of task completions. When the system reaches its capacity, an arriving task is dropped.

B. Transition Probabilities

Let $p_{ss'}(a)$ be the transition probability that the system transitions from state $s = (q_L, q_G, q_B, z)$ to $s' = (q'_L, q'_G, q'_B, z')$ under action a. The rate of departure (say, $v(\cdot)$) from the state s is given by $v(s) = \lambda + \mu(q_L + q_G + q_B)$. Let p_g be the probability that a new task arriving at an IoT node experiences a good channel condition. Then the transition probability [12] is

$$p_{ss'}(a) = [\lambda p_g \mathbb{1}_{\{z'=Z_1\}} + \lambda(1-p_g)\mathbb{1}_{\{z'=Z_2\}} + q'_L\mu\mathbb{1}_{\{z'=Z_3\}} + q'_G\mu\mathbb{1}_{\{z'=Z_4\}} + q'_B\mu\mathbb{1}_{\{z'=Z_5\}}]/v(s').$$

The number of actively processing tasks under a given action a and state s are $q'_L = q_L - \mathbb{1}_{\{z=Z_3\}} + \mathbb{1}_{\{a=A_1\}}$, $q'_G = q_G - \mathbb{1}_{\{z=Z_4\}} + \mathbb{1}_{\{a=A_2 \wedge z=Z_1\}}$ and $q'_B = q_B - \mathbb{1}_{\{z=Z_5\}} + \mathbb{1}_{\{a=A_2 \wedge z=Z_2\}}$ (see [6] for further details).

C. Power Consumption and Deadline Violation Cost

Finite power consumption and deadline violation costs are incurred for every action taken for task scheduling. The power requirement for task computation is given by $P_{c,x} = w_x f_x^3$ where $x \in \{L, M\}$ represents the processing device (Local node / MEC Server), f_x is the clock frequency of the Central Processing Unit (CPU), and w_x is the processor architecture coefficient [6]. For offloading a task (action A_2), the task needs to be transmitted via a wireless channel to the server for processing. Let the average channel gains for good and bad channel conditions be $\mathbb{E}[|h_g|^2]$ and $\mathbb{E}[|h_b|^2]$, respectively, and let $d = \mathbb{E}[|h_g|^2]/\mathbb{E}[|h_b|^2]$ be known to the controller. The transmission power of the IoT node having bad channel conditions ($P_{o,b}$, say) is increased by a factor of d w.r.t good channel device power ($P_{o,g}$, say) to maintain the average received signal power [13]. Hence, the power consumption cost rate (say, $c_p(s,a)$) is

$$c_p(s,a) = q'_L P_{c,L} + (q'_B + q'_G)P_{c,M} + (q'_G + dq'_B)P_{o,g}. \quad (1)$$

For processing a task locally at the parent IoT node, the total delay ($T_{c,L}$, say) is due to the computation time only and is given by $T_{c,L} = BC/f_L$, where B is the payload size of the task in bits processed per time slot, and C is the number of CPU clock cycles required for processing a bit. The service time of the entire task is exponentially distributed with a mean of $1/\mu$. For MEC processing via offloading, the total delay ($T_{c,M}(s)$, say) is due to transmission delay as well as computation time and is given by $T_{c,M}(s) = B/R_M + BC(q_G + q_B + 1)/f_M$, where R_M is the wireless channel capacity. Here, $T_{c,M}(s)$ is stochastic (due to R_M being random) and depends on the system state s. The first and second terms in $T_{c,M}(s)$ represent the task transmission time and the computation time (where the CPU resources are distributed uniformly among the active tasks), respectively. The wireless channel capacity [11] is

$$R_M = \begin{cases} W\log_2\left(1 + \frac{P_{o,g}|h_g|^2}{N_0}\right), & \text{if good channel,} \\ W\log_2\left(1 + \frac{P_{o,b}|h_b|^2}{N_0}\right), & \text{if bad channel,} \end{cases}$$

where W is the channel bandwidth, and N_0 is the noise power. It is assumed that the IoT nodes have one to one mapping with non-overlapping channels of bandwidth W [11]. Finally, the deadline violation cost rate in state s under action a ($c_d(s,a)$, say) is:

$$c_d(s,a) = \mathbb{1}_{\{[a=A_1 \wedge T_{c,L}>D] \vee [a=A_2 \wedge T_{c,M}(s)>D]\}}, \quad (2)$$

where D is the deadline associated with a task. When system capacity is reached (i.e., $q_L = N_1$ and $(q_G + q_B) = N_2$), we drop a new task.

III. Problem Formulation & Solution Technique

The problem formulation is similar to [6], where the authors aim to determine a sequential decision-making policy (or rule) that is robust to model mis-specifications [5]. The task scheduling problem is formulated as an R2CMDP where the objective is to minimize the worst-case expected total discounted cost due to power consumption with a constraint on the expected total discounted deadline violation cost. In this case, a *randomized stationary* optimal policy is known to exist [14].

All states are reachable from any state, with a non-zero probability, under any *stationary policy*. Therefore, the underlying Markov chains are irreducible, and the stationary distributions are unique. We assume that the transition probabilities of the underlying MDP vary within an uncertainty set $\mathcal{P}$ with a nominal transition probability matrix P_0 (say) at its center. The next state transition probabilities under action a in state s are $[p_{ss'}(a)]_{1\times|\mathcal{S}|} \in \mathcal{P}(s,a)$, where $\mathcal{P}(s,a)$ is the uncertainty set for state-action pair (s,a). Since a stationary optimal policy exists for our problem, we consider a rectangular uncertainty set (See [5, Lemma 3.3]). This implies that *nature* can select the transition function independently for each state-action combination (s,a). Let Π be the set of all stationary policies, $\gamma \in [0,1)$ be the discount factor, and $D_{\max}$ be the upper bound on the expected total discounted deadline violations. Then our infinite-horizon discounted continuous time R2CMDP objective is given by

$$\min_{\pi\in\Pi} \max_{P'\in\mathcal{P}} \mathop{\mathbb{E}}_{P',\pi}\left[\int_{t=0}^{\infty}\gamma^t c_p(s_t,a_t)dt\right],$$
$$\text{subject to } \mathop{\mathbb{E}}_{P_0,\pi}\left[\int_{t=0}^{\infty}\gamma^t c_d(s_t,a_t)dt\right] \le D_{\max},$$

where the expectation operator is over policy π and the transition probability matrix P. The optimal stationary randomized policy $\pi : \mathcal{S} \to \mathcal{A}$ can be obtained by solving the R2CMDP problem. The obtained policy will exhibit robustness in terms of returns while satisfying the constraint.

Uniformization technique [12] is used to transform the continuous-time R2CMDP into a discrete-time R2CMDP. Let $\tau(s,a)$ be the expected time the system stays in state s under action a in the continuous-time domain. Similarly, in the discrete-time domain, let $\hat{p}_{ss'}(a)$, $\hat{c}_d(s,a)$, and $\hat{c}_p(s,a)$ be the equivalent transition probabilities, cost due to deadline violations, and cost due to power consumption, respectively. Using uniformization, $\hat{p}_{ss'}(a) = p_{ss'}(a)\mathbb{1}_{\{s'\neq s\}}\eta/\tau(s,a) + (1-\eta/\tau(s,a))\mathbb{1}_{\{s'=s\}}$, where $0 < \eta \ll \min_{s,a}\tau(s,a)$. Similarly, we have $\hat{c}_p(s,a) = c_p(s,a)$, and $\hat{c}_d(s,a) = c_d(s,a)$ [12].

IV. Robust Deep RL Algorithm

In [6], a tabular RL based robust task scheduling algorithm (called TSA-RQ) is proposed. However, it is not scalable to large state-action spaces as the storage and computational complexities become very high. Towards this, in this section, we propose a novel task scheduling algorithm that integrates elements of deep learning within the framework of robust RL based task scheduling [6]. We refer to it as the Task Scheduling Algorithm based on R2C Deep Q-learning (TSA-RDQ) (See Algorithm 1). The objective is to learn the optimal policy for an MEC-IoT network to minimize the worst-case expected total discounted cost due to power consumption, subject to a constraint on the expected total discounted deadline violations. The proposed task allocation algorithm is developed using a conglomeration of ideas from deep Q-learning [15] and two-timescale Stochastic Approximation (SA) [16]. Note that the consideration of the second timescale is to accommodate the constraints associated with the problem.

A. Construction of Uncertainty Set

The proposed algorithm, similar to TSA-RQ, relies on the structure of the uncertainty set. The R-contamination model [17] is used to construct the uncertainty set for a state-action pair as $\mathcal{P}_s^a = \mathcal{P}(s,a) = \{(1-R)p_s^a + Rq \mid q \in \Delta_{|\mathcal{S}|}\}$, where $p_s^a = P_0(s,a) = [p_{ss'}(a)]_{1\times|\mathcal{S}|}$ is the centroid of the uncertainty set for the pair (s,a), $R \in [0,1]$ is the design parameter of the uncertainty set, and q is a random probability simplex in $\mathbb{R}_+^{|\mathcal{S}|-1}$. Now, let the uncertainty set for the transition probability matrix be $\mathcal{P} = \bigotimes_{(s,a)\in\mathcal{S}\times\mathcal{A}}\mathcal{P}_s^a$, where $\bigotimes$ is the direct product. This implies (s,a)-rectangularity on $\mathcal{P}$ [5]. The physical interpretation of the model is that R measures the size of the uncertainty set (corresponding to the set in which the task arrival rate can belong) for which $p_s^a = P_0(s,a)$ acts as the centroid (corresponding to the nominal task arrival rate).

B. Two-timescale Robust Deep Q-learning

As described in Algorithm 1, after taking an action a_n at n^{th} iteration, the system transitions from state s_n to state s_{n+1}, and the costs incurred due to power consumption and deadline violations are obtained as $\hat{c}_p(s_n,a_n)$ and $\hat{c}_d(s_n,a_n)$, respectively. Following [17], this observation is used to obtain an unbiased estimate of $\mathcal{P}_s^a$ at iteration n. The single-sample based maximum likelihood estimate of p_s^a at n^{th} iteration is $\hat{p}_{s_n}^{a_n} = \mathbf{1}_{\{s_{n+1}\}}(s')\ \forall\ s' \in \mathcal{S}$. Similarly, the single-sample based estimate of $\mathcal{P}_s^a$ at n^{th} iteration is $\hat{\mathcal{P}}_{s_n}^{a_n} \triangleq \{(1-R)\hat{p}_{s_n}^{a_n} + Rq | q \in \Delta_{|\mathcal{S}|}\}$. Define the Lagrangian function as

$$\mathcal{L}(s,a,\beta) = \hat{c}_p(s,a) + \beta(\hat{c}_d(s,a) - \bar{D}_{\max}), \tag{3}$$

where $\bar{D}_{\max} = D_{\max}(1-\gamma)$. The Bellman operator $\boldsymbol{T}$ on the quality function $Q(s,a)$ (say) for the R2CMDP is: $\boldsymbol{T}Q(s,a) = \mathbb{E}[\mathcal{L}(s,a,\beta)] + \gamma\max_{P_s^a\in\mathcal{P}_s^a}\mathbb{E}[V(s')]$, where $V(s) = \min_{a\in\mathcal{A}}Q(s,a)$ is the value function of state $s \in \mathcal{S}$.

The $Q(s,a)$ is approximated using a Deep Neural Network (DNN) $Q_\theta(s,a) = Q(s,a;\theta)$ with parameter set θ such that $|\theta| \ll |\mathcal{S}\times\mathcal{A}|$. In our system formulation, $|\mathcal{A}|$ is small (precisely, $\mathcal{A} = \{A_0, A_1, A_2\}$) while $|\mathcal{S}|$ is finite, albeit it can be very large for a large IoT network (with large N_1 and N_2). The DNN takes state s as input and provides estimates of Q-function values $Q_\theta(s,a),\ \forall\ a \in \mathcal{A}$ as output. A target network $Q_{\theta'}(s,a)$ with the same underlying model is used to reduce the effect of moving Temporal Difference (TD) target.

The R2C Q-learning TD target is defined as $Q'_n(s_n, a_n) = \mathcal{L}(s_n, a_n, \beta_n) + \gamma R V_w + \gamma(1-R)\min_a Q_{\theta'_n}(s_{n+1}, a)$, where $V_w = \max_s \min_a Q_{\theta'_n}(s, a)$ is the value function of the worst-possible state at n^{th} iteration based on the target network and is determined using the Adaptive Moment Estimation (ADAM) optimizer [18]. Similarly, $\min_a Q_{\theta'_n}(s_{n+1}, a)$ is the value function of the next state, and is directly computed using $\min(\cdot)$ operator on the output of the target network. Note that at $R = 0$, the $Q'_n(s_n, a_n)$ reduces to the traditional constrained Deep Q-Network (DQN) TD target. When $0 < R < 1$, then the target has an additional term to account for the worst case scenario, leading to R2C Q-learning. We update θ along a faster timescale [16] $g(n)$ using the Batch Gradient Descent (BGD) method. To this end, a finite length replay memory buffer (Say, $\mathcal{M}$) is used to record the last N_m state transitions as $\mathcal{M}_{n+1} = \{(s_k, a_k, \mathcal{L}_k, s_{k+1}) | n - N_m + 1 \leq k \leq n\}$. A mini-batch of B number of transitions $\{(S_i, A_i, \mathcal{L}_i, S'_i)\}_{i=1}^{B} \sim \mathcal{M}_{n+1}$ is uniformly sampled from the buffer to calculate the Mean Squared Error (MSE) loss (line 6 of Algorithm 1)

$$L_{\theta_n} = \frac{1}{B}\sum_{i=1}^{B}(Q_{\theta_n}(S_i, A_i) - y_i)^2, \tag{4}$$

where $y_i = \mathcal{L}_i + \gamma((1-R)\min_{a'} Q_{\theta'_n}(S'_i, a') + RV_w)$ represents the target.

Now, similar to the framework of SA [16], we use two positive step size sequences, $g(n)$ and $h(n)$, that satisfy $h(n) = \mathbf{o}(g(n))$ along with other standard learning rate conditions. Note that these signify the learning rates in two different time scales. In other words, the learning rate of one timescale changes slowly compared to the learning rate of the other timescale. We then update the parameters of the policy Q-network using gradient descent (line 7 of Algorithm 1)

$$\theta_{n+1} \leftarrow \theta_n - g(n)\nabla_\theta L(\theta_n). \tag{5}$$

The following Polyak update rule [4] is used to update the parameters of the target Q-network (line 8 of Algorithm 1)

$$\theta'_{n+1} = (1-\tau)\theta'_n + \tau\theta_{n+1}, \tag{6}$$

where $0 < \tau \ll 1$ is the learning rate for the updates. We update β along a slower timescale [16] $h(n)$ using the following gradient ascent scheme (line 9 of Algorithm 1):

$$\beta_{n+1} = \Gamma[\beta_n + h(n)\sum_{k=0}^{n}\gamma^k(\hat{c}_d(s_k, a_k) - \bar{D}_{\max})], \tag{7}$$

where $\Gamma[.]$ is a projection operation given by $\Gamma[x] = \max\{0, \min\{x, \beta_{\max}\}\}$, and $\beta_{\max} > 0$ is a large real number. Note that the updates in the faster timescale perceive the iterative updates in the slower timescale as quasi-static. Meanwhile, the slower timescale updates view the faster timescale updates as equilibrated [16]. This two-timescale algorithm introduces a *leader-follower* approach without any interference with each other's convergence behavior.

C. Complexity Analysis

When the Q-network is a feed forward neural network, then the per iteration computational complexity of TSA-RDQ is $\mathbf{O}(|\mathcal{A}| + |\theta_{L,\max}|^2)$, where $|\theta_{L,\max}|$ is the number of parameters in the widest layer. The storage complexity is $\mathbf{O}(|\theta|)$. Note that $|\theta_{L,\max}| \ll |\theta| \ll |\mathcal{S}|$. We compare the complexity of our algorithm with traditional RL (QTSA [8]) and robust tabular RL (TSA-RQ), and present it in Table I. Tabular RL based algorithms, QTSA and TSA-RQ have a high storage complexity of $\mathbf{O}(|\mathcal{S}||\mathcal{A}|)$. TSA-RQ has the highest computational complexity of $\mathbf{O}(|\mathcal{S}||\mathcal{A}|)$ due to inner minimax optimization needed for robustness.

Algorithm 1: TSA-RDQ.

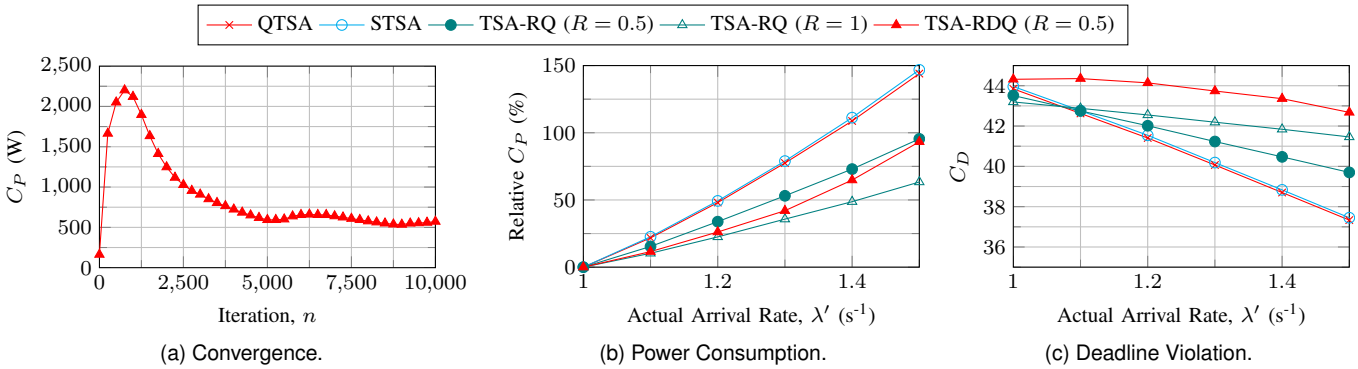
Input: γ, R, $D_{\max}$, $g(n)$, $h(n)$, N_m, B, τ and $\Gamma[.]$.
Initialization: Randomize $\beta_0, \theta_0, \theta'_0$, and $\mathcal{M}_0 \leftarrow \{\emptyset\}$.
1 **for** $n = 0, 1, 2, \ldots$ **do**
2 Take a_n based on ϵ_n-greedy policy w.r.t. $Q_{\theta_n}(s_n, a)$ and observe s_{n+1}.
3 Obtain $c_{p,n}$ and $c_{d,n}$ using (1) and (2).
4 Calculate $\mathcal{L}_n$ using (3) and update $\mathcal{M}_{n+1}$.
5 **if** $|\mathcal{M}_{n+1}| \geq B$ **then**
6 Compute MSE loss using (4).
7 Update policy Q-network using (5).
8 Update target Q-network using (6).
9 Update Lagrange multiplier β_{n+1} using (7).
10 **end**
11 $s_n \leftarrow s_{n+1}$.
12 **end**

TABLE I
FEATURES OF VARIOUS ALGORITHMS.

	QTSA	TSA-RQ	TSA-RDQ (ours)
Comp. Complexity	$\mathbf{O}(\lvert\mathcal{A}\rvert)$	$\mathbf{O}(\lvert\mathcal{S}\rvert\lvert\mathcal{A}\rvert)$	$\mathbf{O}(\lvert\mathcal{A}\rvert+\lvert\theta_{L,\max}\rvert^2)$
Storage Complexity	$\mathbf{O}(\lvert\mathcal{S}\rvert\lvert\mathcal{A}\rvert)$	$\mathbf{O}(\lvert\mathcal{S}\rvert\lvert\mathcal{A}\rvert)$	$\mathbf{O}(\lvert\theta\rvert)$
Robustness	✖	✔	✔

V. SIMULATION RESULTS

In this section, we perform system level simulations on the discrete event simulator (ns-3) to evaluate the robustness performance of the proposed algorithm while comparing it with other task scheduling algorithms in the literature [3], [6] and [8]. To simulate the distributional shift, we first learn the optimal policy with a nominal task arrival rate $\lambda = 1$ s^{-1}, and then evaluate the performance of the policy obtained on the arrival rate (say λ') such that $\lambda' \in [1 \text{ s}^{-1}, 1.5 \text{ s}^{-1}]$. The DQN is a multi-layer perceptron with two hidden layers of 10 neurons each and sigmoid activation functions. The values of different simulation parameters [11] are: $N_0 = -132.24$ dBm, $W = 180$ kHz, $f_M = 3$ GHz, $f_L = 10$ MHz, $w_L = 10^{-21}$ W/Hz^3, $w_M = 10^{-26}$ W/Hz^3, $B = 512$ bits and $C = 10^4$ CPU cycles/bit. Moreover, we choose $R = 0.5$,



(a) Convergence. (b) Power Consumption. (c) Deadline Violation.

Fig. 2. Performance comparisons of various algorithms ($\mu = 1\ s^{-1}$).

$N_1 = N_2 = 50$, $p_g = 0.5$, $D = 20$ ms, $\mathbb{E}[|h_g|^2] = 0.19$, $P_{o,g} = 0.5$ W, $d = 2$, $\gamma = 0.99$, and $D_{\max} = 45$. The learning rates are taken as $g(n) = \lceil n/500 \rceil^{-0.6}$ and $h(n) = \lceil n/10 \rceil^{-0.8}$, respectively. Note that D_n is approximated as $D_n \approx \frac{1}{1-\gamma}\mathbb{E}_\pi[c_d]$, which holds when $\gamma \to 1$ [12]. As shown in Fig. 2a, $C_P (= \sum_{k=0}^{n} \gamma^k \hat{c}_p(s_k, a_k))$ for TSA-RDQ converges after approximately 5000 iterations. The plots for TSA-RDQ are averaged over 30 independent runs.

Now, we compare the performance of the proposed algorithm (TSA-RDQ) with the state-of-the-art learning based task scheduling algorithms, namely, QTSA [8], STSA [3] and TSA-RQ [6]. We measure the performance of the algorithms in terms of relative C_P (percentage increase in C_P as compared to C_P at $\lambda = 1\ \text{s}^{-1}$), which is a measure of the robustness of power consumption and C_D $(= \sum_{k=0}^{n} \gamma^k \hat{c}_d(s_k, a_k))$. Fig. 2b and Fig. 2c demonstrate that relative performance metrics of QTSA and STSA show much higher sensitivity to changes in λ' than the proposed TSA-RDQ with $R = 0.5$. Plots with other values of R show a similar trend as in Fig. 2. The relative C_P at $\lambda' = 1.5s^{-1}$ for QTSA, STSA, TSA-RQ with $R = 0.5$ and the proposed algorithm TSA-RDQ with $R = 0.5$ are 144%, 147%, 95% and 93%, respectively. We observe a similar trend at other values of λ'. Thus, the sensitivity in terms of power consumption under varying λ' is much better for TSA-RDQ and TSA-RQ than the state-of-the-art QTSA and STSA. Hence, the performance of TSA-RDQ is slightly better than tabular TSA-RQ with a significant drop in storage and computational complexity. Note that the C_D for all these algorithms satisfies the constraint.

VI. Conclusion

This work targets the issue of scalability of R2CMDP based tabular RL algorithms for MEC enabled IoT systems with large state spaces. To address this, we have proposed a robust DRL algorithm for the task scheduling problem to minimize the worst-case expected total discounted power consumption subject to a constraint on the expected total discounted deadline violations. It also provides varying degrees of robustness based on the value of R. ns-3 simulation results indicate that TSA-RDQ is more robust compared to other task scheduling algorithms, while being scalable to a large number of IoT nodes. Future work can further investigate the theoretical convergence and robustness performance of the proposed method.

References

[1] P. Porambage *et al.*, "Survey on Multi-Access Edge Computing for Internet of Things Realization," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2961–2991, 2018.

[2] Y. Liu *et al.*, "Toward Edge Intelligence: Multiaccess Edge Computing for 5G and Internet of Things," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 6722–6747, 2020.

[3] T. Alfakih *et al.*, "Task Offloading and Resource Allocation for Mobile Edge Computing by Deep Reinforcement Learning Based on SARSA," *IEEE Access*, vol. 8, pp. 54 074–54 084, 2020.

[4] R. Sutton and A. Barto, *Reinforcement Learning: an Introduction*. MIT Press, 2018.

[5] G. N. Iyengar, "Robust Dynamic Programming," *Mathematics of Operations Research*, vol. 30, no. 2, pp. 257–280, 2005.

[6] V. Masih and A. Roy, "Robust Learning Algorithms for Task Scheduling in MEC-Enabled IoT Networks," *IEEE Transactions on Green Communications and Networking*, vol. 10, pp. 497–510, 2026.

[7] X. Zhang *et al.*, "MDP-Based Task Offloading for Vehicular Edge Computing Under Certain and Uncertain Transition Probabilities," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 3, pp. 3296–3309, 2020.

[8] J. Li *et al.*, "Deep reinforcement learning based computation offloading and resource allocation for MEC," in *IEEE WCNC*, 2018, pp. 1–6.

[9] X. Xiong *et al.*, "Resource allocation based on deep reinforcement learning in IoT edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1133–1146, 2020.

[10] J. Xu *et al.*, "Deep reinforcement learning for computation and communication resource allocation in multiaccess MEC assisted railway IoT networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 12, pp. 23 797–23 808, 2022.

[11] A. Roy and N. Biswas, "Robust & Low-Complexity Task Scheduling Algorithms for a Mobile Edge Computing System," *IEEE Transactions on Green Communications and Networking*, vol. 9, no. 3, pp. 1340–1353, 2025.

[12] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[13] 3GPP TSG Radio Access Network, "NR; Physical layer procedures for control," 3rd Generation Partnership Project (3GPP), Technical Specification 3GPP TS 38.213 V18.2.0, 2024.

[14] E. Altman, *Constrained Markov Decision Processes: Stochastic Modeling*. Routledge, 2021.

[15] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[16] V. S. Borkar, *Stochastic approximation: a dynamical systems viewpoint*. Cambridge University Press, 2008.

[17] Y. Wang and S. Zou, "Online Robust Reinforcement Learning with Model Uncertainty," in *Advances in Neural Information Processing Systems*, vol. 34, 2021, p. 7193–7206.

[18] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR 2015, Conference Track Proceedings*, 2015.