2023 62nd IEEE Conference on Decision and Control (CDC)
December 13-15, 2023. Marina Bay Sands, Singapore

Efficient Off-Policy Algorithms for Structured Markov Decision Processes

Sourav Ganguly[1], Raghuram Bharadwaj Diddigi[2] and Prabuchandran K J[1]

Abstract—

Reinforcement Learning (RL) algorithms help in training an autonomous agent to learn the optimal actions in an unknown environment. RL, in conjunction with neural networks, known as deep RL, has achieved remarkable success in many real-life practical applications. However, most of these algorithms are complex to train and require a lot of data to learn optimal decisions. Hence, it is imperative to develop RL algorithms that are simple and data efficient. In this work, we propose off-policy RL algorithms that can exploit special structures present in the optimal policy of the underlying Markov Decision process. The off-policy learning enables us to make use of the available data efficiently. To this end, we first propose an off-policy algorithm that estimates the value function of only those policies with the optimal policy structure to determine the best policy. We then propose two novel off-policy algorithms utilizing Upper Confidence Bound (UCB) with less time and space complexity than our first algorithm. Through extensive experimental evaluations on RL benchmark tasks, we illustrate the efficacy of the proposed algorithms.

I. Introduction

The sequential decision-making problems are one of the most commonly encountered problems in Artificial Intelligence, e.g., self-driving cars ([1], [2], [3], [4]), managing stock portfolio ([5], [6], [7]), playing card games ([8], [9], [10]) etc. Reinforcement Learning [11], a paradigm of machine learning is the most widely used technique to solve sequential decision-making problems. The solution based on RL has been successfully implemented in domains such as robotics ([12], [13], [14], [15]), games ([16], [17], [18], [19]), and recommendation systems ([20], [21], [22]).

The first step in solving a sequential decision-making problem involves formulating it in the mathematical framework of the Markov Decision Process (MDP). The next step then involves learning the best action or decision to be taken in any given state (i.e., the information available at each instant). This mapping from state space to the action space is called the policy function or simply policy, and the goal here is to learn an optimal policy. If the complete model information of the problem is known apriori, one could solve MDPs efficiently using dynamic programming techniques [23]. However, the model information is not known beforehand in many practical problems such as queuing problems [24], inventory management problems [25] etc. The objective in such problems is to learn the optimal policy based on data-driven or simulation-based techniques [26]. The model-free data-driven methods for learning the optimal actions or policy are known as Reinforcement Learning (RL) algorithms.

The number of states and actions determines the size of an MDP. Suppose there are m states and n actions possible in each state for an MDP, the total number of possible policies is n^m, from which one needs to determine the optimal policy. It is easy to see that if the number of states increase linearly, the number of policies increase exponentially. This makes training a standard RL algorithm and learning the optimal policy challenging. However, some practical applications, for example, machine Replacement problem [27], River-swim problem [28], inventory management problem [29], slow-server problem [26], etc. modeled as MDPs have a special structure in the optimal policy which when exploited can reduce the policy search space significantly [26]. Thus, instead of searching for the best policy among all the policies possible in the search space, exploring only those policies that are members of the reduced policy space (policies with structure) guarantees a significant reduction in the search complexity (from exponential to polynomial time complexity) [30].

Another important problem in determining the optimal policy by an RL algorithm is related to the nature of data used for training. As indicated earlier, RL is a data-driven method where the best policy is learned iteratively over time from the available data samples. There are two different ways of obtaining the data samples for training. The first is the on-policy training wherein the data can be obtained by acting according to the policy obtained at each iteration. Note that the data distribution changes according to the policy at every iteration. The alternate way is the off-policy training wherein either historic data is available or the data at every training iteration can be obtained based on a known policy (known as the behavior policy). In many realistic and practical applications, executing different policies at each iteration is not just undesirable but can also be catastrophic [31]. The suitable paradigm in such settings is the off-policy training.

Unlike on-policy training, off-policy RL training is challenging and requires correction to the state and state-action distribution in the update rules arising from discrepancies between the target and behavior policies at each training time step. To correctly handle this, most of the state-of-

[1] Sourav Ganguly is a master's student and Dr. Prabuchandran K.J. is an assistant professor in the Department of Computer Science and Engineering, Indian Institute of Technology, Dharwad, India. E-mail: {211011002, prabukj}@iitdh.ac.in. Dr. Prabuchandran K.J. is supported by the Science and Engineering Board (SERB), Department of Science and Technology, Government of India under the startup research grant SRG/2021/000048.

[2] Dr. Raghuram Bharadwaj is an assistant professor in the International Institute of Information Technology, Bangalore, India. E-mail: raghuram.bharadwaj@iiitb.ac.in

the-art off-policy algorithms utilize the idea of importance sampling [32]. However, a significant limitation of the importance sampling estimator is high variance problem which greatly affects learning. To mitigate the high variance problem, importance sampling estimators based on the stationary distribution of the policies rather than on trajectories have been proposed [33]. In [34], a complete off-policy actor-critic algorithm utilizing the value function estimation procedure given in [33] has been proposed. However, the correction proposed in [34] in the actor-critic algorithm is not exact, due to which it may not converge. The exact correction is utilized in [35], and an off-policy natural actor-critic algorithm has been proposed with convergence guarantees.

This work proposes three efficient and simple off-policy algorithms for structured MDPs. Unlike the deep RL techniques [36] that do not have robust convergence guarantees and require large amounts of training data, our proposed algorithms converge to optimal policy through intelligent utilization of off-policy data. Our first proposed algorithm computes the stationary distribution ratio between the behavior policy and all the structured policies and subsequently uses the ratios to estimate the average reward of these policies. Though efficient, this method requires simultaneous average value estimation of all structured policies at each training instant, thereby incurring high time and space complexities. In our second algorithm, the Upper Confidence Bound (UCB) [37] technique is employed to reduce the time complexity whereby only one of the policies that is likely to be the optimal policy is selected at each instant for the update. Further, another UCB-based algorithm is also proposed to tackle the space complexity where the stationary distribution of all structured policies can be computed efficiently. We now summarize our contributions as follows:

- We propose three efficient algorithms to find the optimal policy in a structured MDP.
- Our algorithms converge to the optimal policy with limited data samples and do not suffer from high variance problems.
- Through extensive experiments on benchmark RL tasks, we demonstrate that our algorithms perform better than the Q-learning (one of the most widely used off-policy algorithms).

II. Problem Statement and Background

A. Structured Markov Decision Process (MDP)

Markov Decision Processes (MDPs) consists of four components - a set of states called the state space ($\mathcal{S}$), a set of actions (that defines all the possible actions that can be taken in a state) called the action space ($\mathcal{A}$), a transition probability function $\mathcal{P}(s'|s,a)$ that gives the probability of transitioning to the state s' when action a is taken in state s, and a single-stage reward function $R(s,a)$ that provides numerical feedback for taking action a in state s.

We define a stationary policy function $\mu : S \rightarrow \Delta(A)$ as a mapping from state space to distribution over actions. In the average reward MDP settings [38], the objective is to learn an optimal policy μ^* that maximizes the objective $\rho(\mu)$ given by,

$$\rho(\mu) = \lim_{T\to\infty} \frac{1}{T}\mathbb{E}\left[\sum_{t=0}^{T-1} R(s_t,a_t)|s_0 \sim d_0, a_t \sim \mu(s_t)\right], \tag{1}$$

$$= \sum_{s,a} d^{\mu}(s)\mu(s,a)R(s,a), \tag{2}$$

where d_0 is an initial distribution over states, the expectation $\mathbb{E}[\cdot]$ is taken over the set of states and actions sampled at time steps t. Under mild technical conditions [39], (1) can be compactly written as (2), where d^{μ} is the stationary distribution of the states by following policy μ, i.e.,

$$d^{\mu}(s) = \lim_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T-1} Pr\{s_t = s|s_0 \sim d_0,\ \mu\},\ \forall s \in S. \tag{3}$$

In what follows, we will restrict our discussion to the space of deterministic policies, that are mappings from state space to action space (instead of distribution over actions). It has been shown in the literature that a finite MDP has a stationary optimal deterministic policy [23]. If there are m states in the state space and n actions in the action space, then the total number of stationary policies is n^m. In certain MDPs, the optimal policy is known to possess a special structure. To understand this, let us consider an example of an MDP with three states a, b, c and two actions $1, 2$. In this example, there are a total of 8 stationary policies. However, if we assume that the optimal policy is a threshold policy [26], then it can only be one of the four threshold policies given as: $\mu_1 = \{a:1, b:1, c:1\},\ \mu_2 = \{a:1, b:1, c:2\},\ \mu_3 = \{a:1, b:2, c:2\},\ \mu_4 = \{a:2, b:2, c:2\}$. For example, the structure here is such that once the action 2 is decided in state b, it is not optimal to go back to action 1 in subsequent states. It means a policy such as $\mu = \{a:1, b:2, c:1\}$ need not be considered in the search for the optimal policy. Hence, if such threshold structure can be exploited in the search for optimal policy, the solution space can reduce from exponential to polynomial in states.

B. Regret

In "online" RL settings when the policy need to be applied in real-time at each instant, it is critical that the algorithm learns to choose the optimal policy significantly more number of times over a sub-optimal policy. This idea can formalized using the notion of "Regret" [26] and is defined as follows:

$$\mathcal{R}(T) = \rho(\mu^*)(T-1) - \sum_{t=0}^{T-1} \mathbb{E}\left[R(s,\mu_t(s)\right], \tag{4}$$

where μ_t represents the policy chosen at time t. The expected regret until time t is a measure indicating the average number of times the optimal policy gets to be chosen by the algorithm in the time $t = \{0,\ldots,T-1\}$. In our work, we consider "regret" as the measure of performance to analyze the proposed algorithms.

C. Off-Policy learning

RL algorithms are model-free in the sense that they do not have access to the model (transition probability) of the environment and instead rely on the data to minimize the cumulative regret [40]. There has been two approaches using which the agent can obtain the data for its training. In the first approach, the agent learns by directly interacting with the environment and tries to improve the policy. In other words, the agent utilizes the current policy to explore the environment, gather experience, and update its policy parameters. This approach is called on-policy learning. However, as discussed earlier, on-policy learning is not a suitable setting in many safety critical applications. In the second approach, the agent interacts with the environment by executing a behavior policy and learns about the optimal policy or any other policy without executing it. Specifically, the agent uses the behavior policy to interact with the environment and gather experience but updates the target policy based on the experience collected. Unlike on-policy learning, off-policy learning allows the agent to explore alternative policies, making it more flexible and potentially more efficient in finding the optimal policy.

In the off-policy setting, there is a shift in the state and action distributions between target and behavior policies. Thus, one must account for this discrepancy in the policy update rules for successfully learning the optimal policy. In the literature, several methods have been attempted to accomplish this, such as Weighted Importance Sampling (WIS) [41], Doubly Robust (DR) [42], Self normalized importance sampling (SNIS) [43], Retrace estimator [44], Stochastic Gradient Importance Sampling [45] etc.

Importance Sampling (IS): The IS method [46], [47] is a technique for obtaining long-run rewards of a given target policy while following the behavior policy. Let us denote the behavior and target policies by π and μ, respectively. We denote an infinite length trajectory $\tau = (s_i, a_i, r_i, s_{i+1})_{i=0}^{\infty}$ when actions are sampled according to policy π and $\mathcal{P}_\pi$ the probability of obtaining such a trajectory, i.e., $a_i \sim \pi(\cdot|s_i),\ 0 \leq i < \infty$. Moreover, we denote the long-run average cost associated with a trajectory τ as $R(\tau) = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} r_t$ Then, from (1), we have

$$\rho(\mu) = \mathop{\mathbb{E}}_{\tau\sim P_\mu}[R(\tau)] \tag{5}$$

$$= \mathop{\mathbb{E}}_{\tau\sim P_\pi}[\omega(\tau)R(\tau)], \tag{6}$$

where, $\omega(\tau) = \lim_{T\to\infty} \prod_{t=0}^{T} \frac{\mu(s_t)}{\pi(s_t)}$ is called importance sampling weights [11]. It is defined as the ratio of the probability of taking action a in a state s following the target policy μ to the probability of taking the same action a in the same state s following the behavior policy π. Note the important difference in the computation of expectations in (5) and (6). In (5), the trajectories are obtained according to the target policy μ whereas in (6) the trajectories are obtained according to the behavior policy π.

By application of the law of large numbers to (6), one can estimate ρ_μ based on the data samples obtained from π. In practice, however, only finite length trajectories are sampled, and hence the finite average of this trajectory is used as a substitute for $R(\tau)$ in (6). However, it has been noted in the literature that the variance of importance sampling weights $w(\tau)$ are very high [11]. Hence, the estimates obtained using (6) are far from optimal. This has led researchers to consider alternate viewpoints and attempt to obtain efficient off-policy solutions. The following subsection briefly discusses an efficient off-policy value function estimation procedure proposed in [33] that we use in our work.

D. Efficient Off-policy estimation

Recall that our objective is to estimate the long-run average of a given target policy μ given the samples from the behavior policy π. In the previous subsection, we have seen many off-policy estimation procedures suffer from high variance. To mitigate this problem, in [33], a solution was proposed based on (2) (instead of (1)). Here, the long-run average reward of the target policy μ can be written as:

$$\rho(\mu) = \sum_{s,a} d^\mu(s)\mu(s,a)R(s,a) \tag{7}$$

$$= \sum_{s,a} w(s)\beta_{\mu/\pi}(a|s)d^\pi(s)\pi(s,a)R(s,a) \tag{8}$$

$$= E_{(s,a)\sim d^\pi}[w(s)\beta_{\mu/\pi}(a|s)R(s,a)], \tag{9}$$

where $w(s) = \frac{d^\mu(s)}{d^\pi(s)}$ is the ratio of stationary distributions and $\beta_{\mu/\pi}(a|s) = \frac{\mu(s,a)}{\pi(s,a)}$ is the ratio of action distributions (for a deterministic policy μ, we say $\mu(s,a) = 1$ for the action chosen in state s). While computing $\beta_{\mu/\pi}(a|s)$ is straightforward, the $w(s)$ computation is non-trivial. In [33], an efficient technique has been proposed to estimate the ratio of stationary distributions from data samples. Once the ratios $w(s),\ \forall s$ have been estimated, (9) can be invoked to estimate the long-run average reward from the off-policy samples. We refer to this estimation procedure as "Off-policy estimation through ratio of stationary distributions". We now briefly discuss the procedure to estimate the ratio of stationary distribution between target and behavior policies.

As stationary distribution of a Markov chain under mild technical conditions satisfies $d_\mu(s') = \sum_{s\in S} \mathcal{P}(s' \mid s, \mu(s))\ d_\mu(s),\ \forall s'$, $w(s)$ the ratio of stationary distribution can be shown to satisfy a similar recursion (see Theorem 1 and eqn. 9 in [33]). We can now parameterize this $w(s)$ using a neural network and minimize a min-max loss function (see eqn. 10 in [33]) given by:

$$\min_w \{D(w) := \max_{f\in\mathcal{F}} L(w/z_w, f)^2\}, \tag{10}$$

where

$$L(w,f) = \mathbb{E}_{(s,a,s')\sim\rho_\pi}\left[\Delta(w;s,a,s')f(s')\right]. \tag{11}$$

In (11), $\Delta(w;s,a,s') = w(s)\beta_{\mu/\pi}(a|s) - w(s')$, $z_w = \mathbb{E}[w(s)]$, and $\mathcal{F}$ is a set of all functions. By restricting the

set of all functions $\mathcal{F}$ to the space of all functions in a reproducing kernel Hilbert space (RKHS), the maximum loss value within RKHS can be computed (see (10) in [33]) as

$$\max_{f\in\mathcal{F}} L(w,f)^2 = \mathbb{E}\left[\Delta(w;s,a,s^{'})\Delta(w;\bar{s},\bar{a},\bar{s}^{'})k(s^{'},\bar{s}^{'})\right], \tag{12}$$

where $(s,a,s^{'})$ and $(\bar{s},\bar{a},\bar{s}^{'})$ are two independent samples from $\mathcal{P}_\pi$ and k is the positive definite kernel of the RKHS.

In this way, the long-run average reward for a given target policy μ can be estimated using data samples from a behavior policy π. However, the main objective of the RL agent is to learn the optimal policy referred to as the control problem. In the next section, we develop three algorithms for learning the optimal policies that incurs low regret.

III. Proposed Algorithms

In this section, three efficient off-policy algorithms for structured MDPs are proposed. All the algorithms aim to be sample efficient in that the algorithms converge to the optimal policy with fewer data samples. In the following algorithms, as the MDP under assumption has structured optimal policy, let K denote the number of policies with the threshold structure (usually K is polynomial in the number of states instead of being exponential due to the assumption), nS and nA denote the total number of states and all possible actions, respectively. Let T denote the number of time horizons we want to run our algorithm. We assume that the data is collected in batches and provided to the algorithm for learning the optimal policy. Let b denote the batch size (we set it to 50 in our experiments) and d^π denote the stationary state distribution of the behaviour policy π (can be easily computed from the behaviour policy data). Finally, G denotes the total number of gradient update steps we perform per iteration to compute the stationary distribution ratio.

Structured Off-policy Control (SOC) is the first algorithm that is proposed in this paper. In this algorithm, ratios of the stationary distribution of all the structured policies are updated at each iteration using the algorithm discussed in the Section II-D. Utilizing these ratios, estimates of the average reward of each policy are estimated using (9). Let $w_t^{\mu_k}$ denote the estimate of the ratio of stationary distribution between target policy μ_k and the behavior policy π at time t, corresponding to iteratively solving the optimization problem in (12). Then, the average reward of the policy $\mu_k,\ 1\le k\le K$, denoted by $\rho_t(\mu_k)$ can be estimated as follows:

$$\rho_t(\mu_k) = \sum_s d_t^{(\mu_k)}(s) R(s,\mu_k(s)), \tag{13}$$

where

$$d_t^{(\mu_k)}(s) = \frac{w_t^{\mu_k}(s)d^\pi(s)}{\sum_s w_t^{\mu_k}(s)d^\pi(s)},\ \forall s\in\mathcal{S}. \tag{14}$$

The best policy at time t is obtained as:

$$\mu_t = \arg\max_k \rho_t(\mu_k)$$

If we have full parameterization of $w_t^{\mu_k}(s)$ (tabular form for $w_t^{\mu_k}$) and the kernel chosen is a zero one-kernel ($k(s,s')=1$ if $s=s'$ and 0 otherwise), it can be seen that as $t\to\infty$, the estimates of ratios of the stationary distribution of all the target policies converge to the actual ratios (up to a constant factor), i.e.,

$$w_t^{\mu_k} \to w^{\mu_k} \text{as } t\to\infty,\ \forall k\in\{1,\dots,K\}, \tag{15}$$

where w^{μ_k} is the true ratio of stationary distributions (up to a constant factor) between policies μ_k and π.

From (13) and (15), under the full parameterization and zero-one kernel assumptions, we can infer that:

$$\rho_t(\mu_k) \to \rho(\mu_k), \text{as } t\to\infty,\ \forall k\in\{1,\dots,K\}, \tag{16}$$

The complete description of this algorithm is presented in Algorithm 1. Note that the training is performed in off-policy data batches, i.e., (state, action, next_state) samples collected from π, as it is seen to be more stable than the traditional single-sample training.

Algorithm 1 Structured Off-policy Control (SOC) algorithm

Input: Behaviour policy π, Target Policies $\{\mu_k\}_{k=1}^K$, nS, nA, T, b, d^π
Output: L : a list of policies selected at each training instance $l=1,2,\dots,\lfloor T/b\rfloor$
Initialize: K neural networks, each corresponding to the ratio of stationary distributions w.r.t to policies $\{\mu_k\}_{k=0}^K$, $\mathcal{D}$ = empty list
for $t=0,1\dots T-1$ **do**
  $a=\pi(.|s)$, $s^{'}\sim\mathcal{P}(.|s,a)$
  Append $(s,a,s^{'})$ to $\mathcal{D}$
  if $(t+1)$ modulus $b=0$ **then**
    for $k=1,\dots K$ **do**
      Update the neural network k utilising the data $\mathcal{D}$ and estimate $w_t^{\mu_k}$ (see Section II-D))
      $d_t^{(\mu_k)}(s) = \frac{w_t^{\mu_k}(s)*d^\pi(s)}{\sum_s w_t^{\mu_k}(s)*d^\pi(s)} \forall s\in\mathcal{S}$
    end for
    $\mu_t = \arg\max_i \left(\sum_s d_t^{(\mu_i)}(s) * R(s,\mu_i(s))\right)$
    Append μ_t to L
  end if
  $s=s^{'}$
end for
Output: Return L

The "SOC" algorithm is effective and converges to the optimal solution. However, note that at each training step, the neural networks corresponding to the ratios of the stationary distribution of all K policies are updated and thus incurs higher computational time.

We now propose a new algorithm, Structured Off-policy Control using UCB (SOCU), that is computationally efficient. The idea here is to choose a policy at each time instant and update the neural network corresponding to the selected policy alone (instead of updating all neural networks). The natural question now is regarding the selection of the policy to be updated at each instant. It might appear appealing

to update the current best policy at each time instant (i.e., greedy exploitation). However, this strategy might lead to sub-optimal policy evolution [11]. At the extreme end lies the strategy of randomly selecting a policy at each instant (i.e., random exploration). While the latter strategy works well theoretically, converging to the optimal policy would take a long time. Hence there is a need to balance exploitation and exploration intelligently. This suggests utilizing the Upper Confidence Bound (UCB) algorithm [37].

UCB is a well-known algorithm widely used in Multi-arm Bandit problems to balance exploration and exploitation in the agent's decision-making process. Let $r_k(t)$ denote the estimate of the long-run average of policy μ_k at time t. It is calculated as follows:

$$r_k(0) = 0, v_k(t) = \left(\sum_s d_t^{(\mu_k)}(s) * R(s, \mu_k(s)) \right) \tag{17}$$

$$r_k(t) = (1-\alpha) r_k(t-1) + \alpha * v_k(t) \ \forall k = \{1, \ldots, K\}, \tag{18}$$

where α is the threshold for the moving average. Let $n_k(t)$ denote the number of times the policy μ_k has been selected until time t. Then, the policy selected at time t is as follows:

$$\mu_t = \arg\max_{k \in K} \left(r_k(t) + \beta \sqrt{\frac{2 * \log(t)}{n_k(t)}} \right), \tag{19}$$

where β is a threshold that balances exploration and exploitation. Say there are two policies μ_i and μ_j that have a very similar long-run average estimates $r_i(t)$ and $r_j(t)$, respectively. Then, (19) dictates that the policy that has been explored less, i.e., $\arg\min\{n_i(t), n_j(t)\}$ will be selected at time t. On the other hand, if the policies i and j have been explored a similar number of times until time t, then the policy that has the highest estimate, i.e., $\arg\max\{r_i(t), r_j(t)\}$ will be selected. In this manner, the UCB policy selection scheme in (19) intelligently balances exploration and exploitation, which we leverage in our algorithm to learn the optimal policy with fewer computations efficiently. The complete description of this algorithm is presented in Algorithm 2.

The Algorithm 2 successfully reduces the computational time of the solution. However, it still suffers from high space complexity. The neural networks corresponding to all structured policies $\mu_k,\ k \in \{1, \ldots, K\}$ must be stored in Algorithm 2. When the number of states is huge (resulting in more possible policies), a significant storage space is required to store all the networks. Our next algorithm, Structured Off-policy Control using UCB - Variant 2 (SOCU-V2) is aimed at reducing the storage space by storing and updating a single neural network at each time instant rather than K neural networks.

The complete description of this algorithm is presented in Algorithm 3. In this algorithm, the neural network weights are updated corresponding to the policy selected at time t, i.e., μ_t. As $t \to \infty$, the UCB algorithm selects the optimal policy more often under the full parametrization of $w_t^{\mu_k}$ and zero-one kernel assumptions. Hence, the network optimally learns the ratio of stationary distribution between

Algorithm 2 Structured Off-policy Control using UCB (SOCU) algorithm

1: **Input:** Behaviour policy π, list of target policies $\{\mu_k\}_{k=1}^K$, $nS, nA, T, b, d^\pi, \alpha, \beta$
2: **Output:** L : a list of policies selected at each training instance $l = 1, 2, \ldots, \lfloor T/b \rfloor$
3: **Initialize:** K neural networks, each corresponding to the ratio of stationary distributions w.r.t to policies $\{\mu_k\}_{k=0}^K$, $n_k(0) = 1,\ 1 \le k \le K,\ \mathcal{D} =$ empty list
4: **for** $t = 0, 1 \ldots T-1$ **do**
5: $a = \pi(.|s)$, $s^{'} \sim \mathcal{P}(.|s, a)$
6: Append $(s, a, s^{'})$ to $\mathcal{D}$
7: **if** $(t+1)$ modulus $b = 0$ **then**
8: $\mu_t = \arg\max_{k \in K} \left(r_k(t) + \beta \sqrt{\frac{2*\log(t)}{n_k(t)}} \right)$,
9: Update the neural network corresponding to the policy μ_t, and update $w_t^{\mu_t}$ (see Section II-D)
10: $d_t^{(\mu_k)}(s) = \frac{w_t^{\mu_k}(s) * d^\pi(s)}{\sum_s w_t^{\mu_k}(s) * d^\pi(s)},\ \forall s \in \mathcal{S}, 1 \le k \le K$
11: $r_k(t+1) = (1-\alpha) r_k(t) + \alpha * \left(\sum_s d_t^{(\mu_k)}(s) * R(s, \mu_k(s)) \right),\ 1 \le k \le K$
12: $n_k(t+1) = n_k(t) + \mathbb{1}\{\mu_k == \mu_t\},\ 1 \le k \le K$
13: Append μ_t to L
14: **end if**
15: $s = s^{'}$
16: **end for**
17: **Output:** Return L

the behavior policy and the optimal policy. As a result, the long-run average corresponding to the optimal policy is computed using this ratio.

Remark 1: Please note that our proposed algorithms could also be easily applied to the general MDPs (in addition to structured MDPs). However, in general, the number of policies can be very high and scale exponentially with an increase in the number of states and actions that can affect the practical performance in real time. On the other hand, the number of policies in structured MDPs is typically small and it is polynomial in the number of states. This makes our algorithms more suited to structured MDP problems. To utilize our algorithm for general MDPs, we could choose a candidate set of policies and aim to find the best policy among the candidate policies utilizing our algorithms.

Algorithm	Time Complexity	Space Complexity
SOC	$O(T \times K \times G \times b^2)$	O(K)
SOCU	$O(T \times G \times b^2)$	O(K)
SOCU-V2	$O(T \times G \times b^2)$	O(1)

TABLE I: Time and space complexity comparison between the three proposed algorithms. It can be observed that the "SOC" algorithm has higher time complexity compared to the other two proposed algorithms. Moreover, the "SOCU-V2" has least time and space complexities

Algorithm 3 Structured Off-policy Control using UCB - Variant 2 (SOCU-V2) algorithm

Input: Behaviour policy π, list of target policies $\{\mu_k\}_{k=1}^{K}$, $nS, nA, T, b, d^{\pi}, \alpha, \beta$
Output:L : a list of policies selected at each training instance $l = 1, 2, \ldots, \lfloor T/b \rfloor$
Initialize: One neural network, $n_k(0) = 1,\ 1 \leq k \leq K$, $\mathcal{D}$ = empty list
for $t = 0, 1 \ldots T-1$ **do**
  $a = \pi(.|s)$, $s^{'} \sim \mathcal{P}(.|s, a)$
  Append $(s, a, s^{'})$ to $\mathcal{D}$
  if $(t+1)$ modulus $b = 0$ **then**
    $\mu_t = \arg\max_{k \in K} \left(r_k(t) + \beta \sqrt{\frac{2 * \log(t)}{n_k(t)}} \right)$,
    Update the neural network utilising the data $\mathcal{D}$, and update $w_t^{\mu_t}$ (see Section II-D)
    $d_t^{(\mu_k)}(s) = \frac{w_t^{\mu_k}(s) * d^{\pi}(s)}{\sum_s w_t^{\mu_k}(s) * d^{\pi}(s)},\ \forall s \in \mathcal{S}, 1 \leq k \leq K$
    $r_k(t+1) = (1-\alpha) r_k(t) + \alpha * \left(\sum_s d_k^{(\mu_k)}(s) * R(s, \mu_k(s)) \right),\ 1 \leq k \leq K$
    $n_k(t+1) = n_k(t) + \mathbb{1}\{\mu_k == \mu_t\},\ 1 \leq k \leq K$
    Append μ_t to L
  end if
  $s = s^{'}$
end for
Output: Return L

IV. Experiments and Results

In this section, we discuss the results of our proposed algorithms on two benchmark RL tasks. We compare our algorithms with two variants of Q-learning. First is the standard Q-learning algorithm, where the action selection at each training step is through $\epsilon-$greedy policy. For a more accurate comparison, we implement another variant of Q-learning, which we refer to as "Q-learning_bp", where the actions for Q-value updates are sampled from the behavior policy instead of the preferred greedy policy based on the Q-values. Note that the "Q-learning_bp" algorithm is trained on the same data used to train our proposed algorithms, thus an accurate comparison. The performance metric we use to compare is the notion of regret (please refer to (4)). We compare our algorithms on two RL tasks whose optimal policy has structure, viz., (1) machine replacement problem [26] (2) river swim [48] problem. In the machine replacement task, the objective is to decide between performing maintenance or replacing a machine with a new one depending on the current state of the machine. In our experiments, we considered 3 configurations of machine replacement based on the number of states, i.e., $nS = 4, 10,$ and 20. In the river swim task, a swimmer's objective is to decide whether to swim left or right, depending on her current position, to reach the correct destination. In both tasks, we set the behaviour policy to be the one that gives equal probability to all actions in any state. Detailed descriptions of the two benchmark tasks along with the complete implementation codes are given here[1].

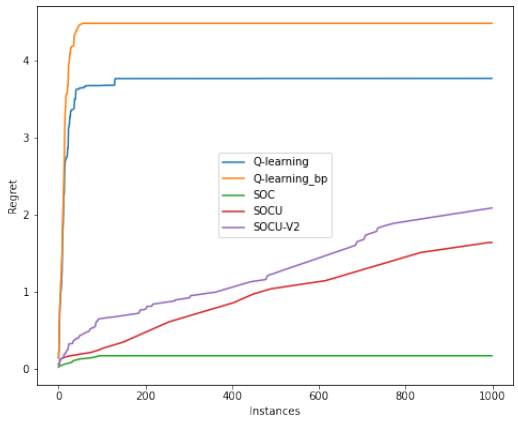

Fig. 1: Comparison between algorithms on four state machine replacement task averaged across five random runs

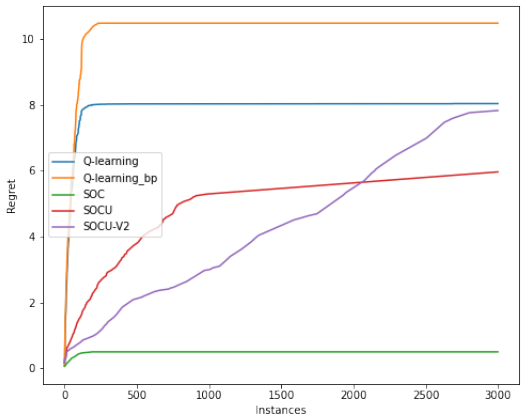

Fig. 2: Comparison between algorithms on ten state machine replacement task averaged across five random runs

A. Discussion

We plot the results obtained on the machine replacement and river swim problems in Fig. 1,2,3 and Fig. 4, respectively. In the figures, we plot the instance (50 training time steps) on the x-axis and the regret on the Y-axis. The following are the observations from the experiments.

1) The SOC algorithm has the least regret among all the proposed algorithms and Q-learning variants. This is because the SOC algorithm efficiently uses the off-policy data at every instant. In this algorithm, the neural networks corresponding to all the policies are simultaneously updated at every time instant. This

[1]https://github.com/Sourav1429/Off_policy_algorithms.git

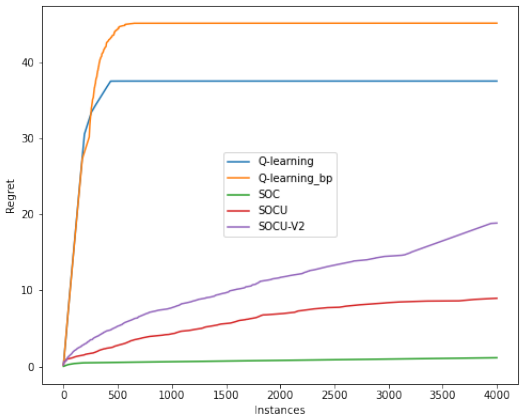

Fig. 3: Comparison between algorithms on twenty state machine replacement task averaged across five random runs

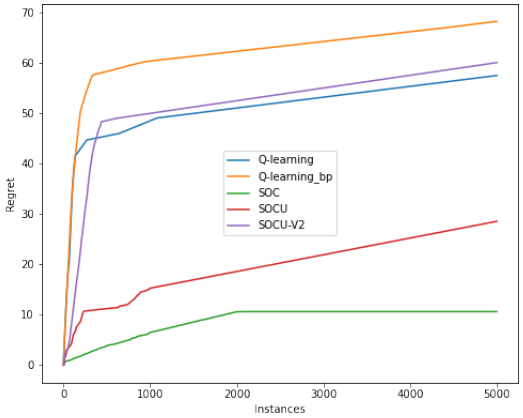

Fig. 4: Comparison between algorithms on river swim task averaged across five random runs

results in the faster convergence of the ratios of stationary distributions of all the policies. This enables the algorithm to learn the best policy in fewer time steps.

2) Although the SOC algorithm has the least regret, updating the weights of all the neural networks at each instance is a costly step, as shown in Table II. Hence, in the SOCU algorithm, one policy is intelligently selected using the UCB algorithm, and only the weights corresponding to the selected policy are updated at each instant. This results in a higher regret (compared to the SOC algorithm). However, the time complexity of the SOCU is significantly better compared to the SOC algorithm (please refer to Table II).
3) The SOC algorithm maintains K neural networks, one for each policy, requiring more storage space. In order to optimize for the space as well, the SOCU-V2 maintains and updates only a single neural network at each training instance. As a result of this trade-off, the regret of the SOCU-V2 is higher than the SOC and the SOCU algorithms.

The results show that our proposed algorithms have lower regret than the off-policy Q-learning algorithms on both benchmark tasks.

Algorithm	Environment	Time taken(in seconds)
SOC	MR (4 states)	58
SOC	MR (10 states)	150
SOC	MR (20 states)	300
SOC	RS	100
SOCU	MR (4 states)	16
SOCU	MR (10 states)	16
SOCU	MR (20 states)	17
SOCU	RS	16
SOCU-V2	MR (4 states)	15
SOCU-V2	MR (10 states)	16
SOCU-V2	MR (20 states)	16
SOCU-V2	RS	15

TABLE II: Execution time per gradient step in a run by all the three proposed algorithms. MR stands for machine replacement task and RS stands for river swim problem

V. Conclusions

In this work, we have proposed three efficient off-policy control algorithms for learning the best policy in structured MDPs. Our algorithms optimally utilize the off-policy data and efficiently select the target policy at each training step. We demonstrate the advantages of the proposed algorithms through experimental evaluations on two benchmark RL tasks. An interesting future direction would be to extend the proposed algorithms to continuous state space environments such as Atari games that would require leveraging the deep neural networks to generalize state-action distribution.

References

[1] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 6, pp. 4909–4926, 2021.

[2] Z. Cao, S. Xu, H. Peng, D. Yang, and R. Zidek, "Confidence-aware reinforcement learning for self-driving cars," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 7419–7430, 2022.

[3] R. Chopra and S. S. Roy, "End-to-end reinforcement learning for self-driving car," in *Advanced Computing and Intelligent Engineering: Proceedings of ICACIE 2018, Volume 1*. Springer, 2020, pp. 53–61.

[4] S. Kardell and M. Kuosku, "Autonomous vehicle control via deep reinforcement learning," *Computer Science*, vol. 48, 2017.

[5] U. Pigorsch and S. Schäfer, "High-dimensional stock portfolio trading with deep reinforcement learning," in *2022 IEEE Symposium on Computational Intelligence for Financial Engineering and Economics (CIFEr)*. IEEE, 2022, pp. 1–8.

[6] Y. Ye, H. Pei, B. Wang, P.-Y. Chen, Y. Zhu, J. Xiao, and B. Li, "Reinforcement-learning based portfolio management with augmented asset movement prediction states," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 01, 2020, pp. 1112–1119.

[7] A. Filos, "Reinforcement learning for portfolio management," 2019. [Online]. Available: https://arxiv.org/abs/1909.09571

[8] J. C. Backhus, H. Nonaka, T. Yoshikawa, and M. Sugimoto, "Application of reinforcement learning to the card game wizard," in *2013 IEEE 2nd Global Conference on Consumer Electronics (GCCE)*, 2013, pp. 329–333.

[9] D. Zha, K.-H. Lai, Y. Cao, S. Huang, R. Wei, J. Guo, and X. Hu, "Rlcard: A toolkit for reinforcement learning in card games," *arXiv preprint arXiv:1910.04376*, 2019.

[10] T. Liu, Z. Zheng, H. Li, K. Bian, and L. Song, "Playing card-based rts games with deep reinforcement learning," in *International Joint Conference on Artificial Intelligence*, 2019.

[11] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[12] J. Kober and J. Peters, *Reinforcement Learning in Robotics: A Survey*. Cham: Springer International Publishing, 2014, pp. 9–67. [Online]. Available: https://doi.org/10.1007/978-3-319-03194-1_2

[13] P. Kormushev, S. Calinon, and D. G. Caldwell, "Reinforcement learning in robotics: Applications and real-world challenges," *Robotics*, vol. 2, no. 3, pp. 122–148, 2013. [Online]. Available: https://www.mdpi.com/2218-6581/2/3/122

[14] T. Zhang and H. Mo, "Reinforcement learning for robot research: A comprehensive review and open issues," *International Journal of Advanced Robotic Systems*, vol. 18, no. 3, p. 17298814211007305, 2021. [Online]. Available: https://doi.org/10.1177/17298814211007305

[15] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, "How to train your robot with deep reinforcement learning: lessons we have learned," *The International Journal of Robotics Research*, vol. 40, no. 4-5, pp. 698–721, jan 2021. [Online]. Available: https://doi.org/10.1177%2F0278364920987859

[16] K. Shao, Z. Tang, Y. Zhu, N. Li, and D. Zhao, "A survey of deep reinforcement learning in video games," *arXiv preprint arXiv:1912.10944*, 2019.

[17] G. Lample and D. S. Chaplot, "Playing fps games with deep reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

[18] M. Lauriere, S. Perrin, S. Girgin, P. Muller, A. Jain, T. Cabannes, G. Piliouras, J. Perolat, R. Elie, O. Pietquin, and M. Geist, "Scalable deep reinforcement learning algorithms for mean field games," in *Proceedings of the 39th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., vol. 162. PMLR, 17–23 Jul 2022, pp. 12 078–12 095. [Online]. Available: https://proceedings.mlr.press/v162/lauriere22a.html

[19] M. Lanctot, E. Lockhart, J.-B. Lespiau, V. Zambaldi, S. Upadhyay, J. Pérolat, S. Srinivasan, F. Timbers, K. Tuyls, S. Omidshafiei, F. Timbers, K. Tuyls, S. Omidshafiei, and D. Hennes, "Openspiel: A framework for reinforcement learning in games," *arXiv preprint arXiv:1908.09453*, 2019.

[20] M. M. Afsar, T. Crump, and B. Far, "Reinforcement learning based recommender systems: A survey," 2021. [Online]. Available: https://arxiv.org/abs/2101.06286

[21] Z. Yan, "Reinforcement learning for recommendations and search," *eugeneyan.com*, Sep 2021. [Online]. Available: https://eugeneyan.com/writing/reinforcement-learning-for-recsys-and-search/

[22] X. Chen, S. Li, H. Li, S. Jiang, Y. Qi, and L. Song, "Generative adversarial user model for reinforcement learning based recommendation system," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 1052–1061. [Online]. Available: https://proceedings.mlr.press/v97/chen19f.html

[23] D. P. Bertsekas and J. N. Tsitsiklis, "Neuro-dynamic programming: an overview," in *Proceedings of 1995 34th IEEE conference on decision and control*, vol. 1. IEEE, 1995, pp. 560–564.

[24] W. Duckworth, A. Gear, A. Lockett, W. Duckworth, A. Gear, and A. Lockett, "Queuing problems," *A Guide to Operational Research*, pp. 32–39, 1977.

[25] S. W. Chan, R. Tasmin, A. N. Aziati, R. Z. Rasi, F. B. Ismail, and L. P. Yaw, "Factors influencing the effectiveness of inventory management in manufacturing smes," in *IOP Conference Series: Materials Science and Engineering*, vol. 226, no. 1. IOP Publishing, 2017, p. 012024.

[26] K. J. Prabuchandran, T. Bodas, and T. Tulabandhula, "Reinforcement learning algorithms for regret minimization in structured markov decision processes," 2016. [Online]. Available: https://arxiv.org/abs/1608.04929

[27] A. Forootani, M. G. Zarch, M. Tipaldi, and R. Iervolino, "A stochastic dynamic programming approach for the machine replacement problem," *Engineering Applications of Artificial Intelligence*, vol. 118, p. 105638, 2023.

[28] I. Osband, D. Russo, and B. Van Roy, "(more) efficient reinforcement learning via posterior sampling," *Advances in Neural Information Processing Systems*, vol. 26, 2013.

[29] S. Agrawal and R. Jia, "Learning in structured mdps with convex cost functions: Improved regret bounds for inventory management," in *Proceedings of the 2019 ACM Conference on Economics and Computation*, 2019, pp. 743–744.

[30] J. Ok, A. Proutiere, and D. Tranos, "Exploration in structured reinforcement learning," 2018. [Online]. Available: https://arxiv.org/abs/1806.00775

[31] S. Padakandla, P. K. J, S. Ganguly, and S. Bhatnagar, "Data efficient safe reinforcement learning," in *2022 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2022, pp. 1167–1172.

[32] J. Hanna, S. Niekum, and P. Stone, "Importance sampling policy evaluation with an estimated behavior policy," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 2605–2613. [Online]. Available: https://proceedings.mlr.press/v97/hanna19a.html

[33] Q. Liu, L. Li, Z. Tang, and D. Zhou, "Breaking the curse of horizon: Infinite-horizon off-policy estimation," 2018. [Online]. Available: https://arxiv.org/abs/1810.12429

[34] Y. Liu, A. Swaminathan, A. Agarwal, and E. Brunskill, "Off-policy policy gradient with state distribution correction," *arXiv preprint arXiv:1904.08473*, 2019.

[35] R. B. Diddigi, P. Jain, P. K J, and S. Bhatnagar, "Neural network compatible off-policy natural actor-critic algorithm," in *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2022, pp. 1–10.

[36] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[37] P. Auer, "Using confidence bounds for exploitation-exploration tradeoffs," *Journal of Machine Learning Research*, vol. 3, no. Nov, pp. 397–422, 2002.

[38] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Athena Scientific Belmont, MA, 1996, vol. 5.

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

[40] M. Jagadeesan, T. Zrnic, and C. Mendler-Dünner, "Regret minimization with performative feedback," 2022. [Online]. Available: https://arxiv.org/abs/2202.00628

[41] P. Bekaert, M. Sbert, and Y. D. Willems, "Weighted importance sampling techniques for monte carlo radiosity," in *Rendering Techniques 2000: Proceedings of the Eurographics Workshop in Brno, Czech Republic, June 26–28, 2000 11*. Springer, 2000, pp. 35–46.

[42] H. Kiyohara, Y. Saito, T. Matsuhiro, Y. Narita, N. Shimizu, and Y. Yamamoto, "Doubly robust off-policy evaluation for ranking policies under the cascade behavior model," in *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*. ACM, feb 2022. [Online]. Available: https://doi.org/10.1145%2F3488560.3498380

[43] Z. Yang, Y. Gao, A. Gerstenberger, J. Jiang, R. Schlüter, and H. Ney, "Self-normalized importance sampling for neural language modeling," 2021. [Online]. Available: https://arxiv.org/abs/2111.06310

[44] B. Daley and C. Amato, "Improving the efficiency of off-policy reinforcement learning by accounting for past decisions," 2021. [Online]. Available: https://arxiv.org/abs/2112.12281

[45] P. Zhao and T. Zhang, "Stochastic optimization with importance sampling," 2014. [Online]. Available: https://arxiv.org/abs/1401.2753

[46] R. Melchers, "Importance sampling in structural systems," *Structural safety*, vol. 6, no. 1, pp. 3–10, 1989.

[47] A. Tabandeh, G. Jia, and P. Gardoni, "A review and assessment of importance sampling methods for reliability analysis," *Structural Safety*, vol. 97, p. 102216, 2022.

[48] A. Ayoub, Z. Jia, C. Szepesvari, M. Wang, and L. Yang, "Model-based reinforcement learning with value-targeted regression," in *International Conference on Machine Learning*. PMLR, 2020, pp. 463–474.