Restless and Uncertain: Robust Policies for Restless Bandits via Deep Multi-Agent Reinforcement Learning

Abstract

We introduce robustness in *restless multi-armed bandits* (RMABs), a popular model for constrained resource allocation among independent stochastic processes (arms). Nearly all RMAB techniques assume stochastic dynamics are precisely known. However, in many real-world settings, dynamics are estimated with significant *uncertainty*, e.g., via historical data, which can lead to bad outcomes if ignored. To address this, we develop an algorithm to compute minimax regret–robust policies for RMABs. Our approach uses a double oracle framework (oracles for *agent* and *nature*), which is often used for single-process robust planning but requires significant new techniques to accommodate the combinatorial nature of RMABs. Specifically, we design a deep reinforcement learning (RL) algorithm, DDLPO, which tackles the combinatorial challenge by learning an auxiliary "λ-network" in tandem with policy networks per arm, greatly reducing sample complexity, with guarantees on convergence. DDLPO, of general interest, implements our reward-maximizing agent oracle. We then tackle the challenging regret-maximizing nature oracle, a non-stationary RL challenge, by formulating it as a multi-agent RL problem between a policy optimizer and adversarial nature. This formulation is of general interest—we solve it for RMABs by creating a multi-agent extension of DDLPO with a shared critic. We show our approaches work well in three experimental domains.

1 INTRODUCTION

Restless multi-armed bandits (RMABs), a model for constrained resource allocation among N independent stochastic processes (arms), are widely studied. Traditionally a *binary-action* problem, in which a planner decides whether or not to act on each of N arms, here we consider the *multi-action-type* generalization [Killian et al., 2021b, Glazebrook et al., 2011], for its enhanced ability to capture challenging real-world planning problems. Salient examples of RMABs include scheduling [Bagheri and Scaglione, 2015, Yang et al., 2018], machine replacement [Glazebrook et al., 2006, Ruiz-Hernández et al., 2020], aerial vehicle routing [Le Ny et al., 2008], anti-poaching patrol planning [Qian et al., 2016], and healthcare [Lee et al., 2019, Mate et al., 2020]. While these works have established important theoretical foundations, they share one key limitation: assuming stochastic dynamics are precisely known. However, having exact knowledge of dynamics is impossible in many real-world problems. E.g., in healthcare intervention planning, the probability that a patient will adhere to treatment after receiving an intervention is not perfectly known a priori; in anti-poaching patrol planning, the probability of finding a poacher's snare at some location is not known with certainty.

Accordingly, methods have been developed to learn RMAB policies *online*, assuming no a priori knowledge [Jung et al., 2019, Wang et al., 2020]. However, these methods require tens of thousands of samples to converge to good policies which is prohibitive for many real-world problems, e.g., in finite-length treatment settings such as tuberculosis [Mate et al., 2020] where there are only hundreds of rounds. Instead, real-world planners must make the most of noisy data at hand, estimating dynamics from historical data or consulting experts, inducing significant *uncertainty*. Although RMAB techniques can be used to plan with point estimates, ignoring uncertainty can lead to arbitrarily bad policies.

To address these shortcomings and push RMABs toward wider real-world applicability, we introduce *Robust RMABs*, a generalization of RMABs which allows stochastic dynamics to be specified as uncertainty intervals, rather than point estimates. This new problem is very computationally demanding, adding a combinatorial layer of complexity onto an already PSPACE-Hard problem [Papadimitriou and Tsitsiklis, 1994]. Addressing this complexity gives rise to a

Submitted to the 38th Conference on Uncertainty in Artificial Intelligence (UAI 2022). **To be used for reviewing only**.

rich set of challenges that necessitates the design of new techniques that not only help solve the robust objective we analyze, but also are of general interest to RMAB research.

Concretely, we plan under a *minimax regret* objective, using a double oracle (DO) framework [McMahan et al., 2003] that has seen success in problems involving a *single* Markov decision process (MDP) [Xu et al., 2021]. The DO approach casts the robust planning problem as a zero-sum game between an *agent* oracle and adversarial *nature* oracle. However, existing techniques fail for any non-trivially sized RMABs since the state and action spaces grow combinatorially in the number of arms N and resource constraint B, respectively. Specifically, given S-sized state spaces for each arm, in the full combinatorial problem, the state space is of size S^N, and the action space, and thus policy-network output, is of size $\binom{N}{B}$ (for binary-action RMAB—action space is larger for multi-action RMAB). At this size, we found that Xu et al. [2021], which tries to solve the full combinatorial problem as a single process, failed to learn good policies for RMABs as small as $N = 5$ arms, with $B = 3$ and $S = 2$. Moreover, under the minimax regret objective, the nature oracle represents a particularly difficult challenge, since it requires jointly searching the RMAB policy space and the continuous, uncertain space of transition probabilities. Previously, this has been posed as a non-stationary RL problem and solved heuristically with a single policy network [Xu et al., 2021]. We improve over this by reformulating the nature oracle as a multi-agent RL problem, and develop a multi-agent RL method for solving it for RMABs. In summary, our contributions are as follows:

1. We introduce the Robust RMAB problem with interval uncertainty over arm dynamics and develop techniques to solve a minimax regret objective via a DO approach.
2. To enable the DO approach, we introduce DDLPO, a novel deep RL algorithm for RMABs, of general interest. DDLPO tackles the combinatorial complexity of RMABs by learning an auxiliary "λ-network" in tandem with individual arm policy networks, which greatly reduces training sample complexity. The procedure implements the reward-maximizing agent oracle, has convergence guarantees, and solves RMABs with multiple action types [Killian et al., 2021b, Glazebrook et al., 2011], the first deep RL procedure to do so. DDLPO also easily extends to more general weakly-coupled MDPs [Adelman and Mersereau, 2008, Hawkins, 2003] and enables computing continuous-action policies, a previously unstudied RMAB direction.
3. We formulate the non-stationary regret-maximizing nature oracle as a multi-agent RL (MARL) problem, a framework of potential general interest in robust planning. We solve it for the combinatorially-hard RMAB setting by extending DDLPO to include a shared critic and a continuous-action policy network for nature's selection of the uncertain transition dynamics.

2 RELATED WORK

RMAB The reward-maximizing, binary-action RMAB problem was introduced by Whittle [1988]. His widely used Whittle index policy [Mate et al., 2020, Glazebrook et al., 2006, Bagheri and Scaglione, 2015] is asymptotically optimal under *indexability* [Weber and Weiss, 1990]. Glazebrook et al. [2011] and Hodge and Glazebrook [2015] extended the Whittle index to multi-action RMABs with special monotonic structure, while Killian et al. [2021b] gave a more general Lagrange-based method. Hawkins [2003] studied methods for weakly coupled Markov decision processes (WCMDP), which generalize multi-action RMABs to have multiple constraints, proposing Lagrangian solutions for small problems. Adelman and Mersereau [2008] and Gocgun and Ghate [2012] followed by providing better solutions to WCMDPs but sacrifice scalability. All these works assumed precise knowledge of stochastic dynamics. Some recent works have studied online RMABs with unknown dynamics but all have prohibitively large sample complexity [Gafni and Cohen, 2020, Jung and Tewari, 2019, Biswas et al., 2021, Killian et al., 2021a]. None consider robust planning under environment uncertainty, which we address.

RL for RMAB A few recent works learn Whittle indexes for indexable binary-action RMABs using (i) deep RL (DRL) [Nakhleh et al., 2021] and (ii) tabular Q-learning [Biswas et al., 2021, Fu et al., 2019, Avrachenkov and Borkar, 2020]. Killian et al. [2021a] take tabular Q-learning to the multi-action setting. In contrast, our DRL approach provides a more general solution to binary and multi-action RMAB domains, not requiring indexability or problem structure, and is far more scalable than tabular methods. We are also the first to handle continuous-action RMABs, key to the nature oracle. Also related is the space of combinatorial RL. However, most existing algorithms consider single-shot problems, e.g., traveling salesman [Kool et al., 2018, Khalil et al., 2017], which lack a notion of future state that is critical to solving any version of RMAB, and none accommodate the general cost/budget structure of multi-action RMAB [Song et al., 2019]; our methods address these limitations.

Robust planning Work on robust planning in RL mainly focuses on maximin reward via robust adversarial RL [Pinto et al., 2017] or multi-agent RL (MARL) [Lanctot et al., 2017, Li et al., 2019], but maximin reward leads to overly conservative policies [Nguyen et al., 2014]. The minimax regret criterion [Braziunas and Boutilier, 2007] avoids this pitfall, but this objective is challenging with very large or continuous strategy spaces. This can be addressed with the DO approach proposed by McMahan et al. [2003] which explores a small subset of strategies while still guaranteeing optimal convergence [Gilbert and Spanjaard, 2017]. Subsequently, DO has been extended to optimize MARL problems with multiple selfish agents [Lanctot et al., 2017]. Recently,

Xu et al. [2021] used DO to solve a single Markov Decision Process (MDP) minimax-regret planning problem and used RL to design the oracles. However, when applied to RMABs, the number of outputs in their policy network grows exponentially, as does the size of the state space being learned, both of which require prohibitively long training times beyond trivially sized RMABs. Accordingly, we found that their RL algorithms failed to scale past $N = 5$ arms and $S = 2$ states, whereas we show in section 5 that our algorithms solve problems that are orders of magnitude larger. Additionally, their approach is designed only for continuous state/action spaces, whereas our approach is capable of finding robust policies for any combination of discrete/continuous state/action spaces. We accomplish this via our novel formulation of the nature oracle as a MARL problem, which decomposes the causes of non-stationarity, i.e., agent and nature, and learn them with separate networks.

3 PRELIMINARIES

We consider the multi-action RMAB setting with N arms [Killian et al., 2021b, Glazebrook et al., 2011], which generalizes classical binary-action RMABs [Whittle, 1988].[1] Each arm $n \in [N]$ follows an MDP $(\mathcal{S}_n, \mathcal{A}_n, \mathcal{C}_n, T_n, R_n, \beta)$, where $\mathcal{S}_n$ is a set of finite, discrete states; $\mathcal{A}_n$ is a set of finite, discrete actions; $\mathcal{C}_n : \mathcal{A}_n \to \mathbb{R}$ defines action costs, where $\mathcal{C}_n[0] = 0$ encodes a no-cost "passive action" for all arms; $T_n : \mathcal{S}_n \times \mathcal{A}_n \times \mathcal{S}_n \to [0, 1]$ gives the probability of transitioning from one state to another given an action; $R_n : \mathcal{S}_n \to \mathbb{R}$ is a reward function; and $\beta \in [0, 1)$ is the discount factor. For ease of exposition, let $\mathcal{S}_n, \mathcal{A}_n, \mathcal{C}_n$, and R_n be the same for all $n \in [N]$, and thus drop the subscript n, though all methods apply to the general case. Let $\boldsymbol{s}$ be an N-length vector of states over all arms and let $\boldsymbol{A} \in \{0, 1\}^{N \times |\mathcal{A}|}$ be a decision matrix that one-hot-encodes the action taken on each arm. The planner computes policies π, which map states $\boldsymbol{s}$ to actions $\boldsymbol{A}$ with the constraint that, for each of H rounds, the sum cost of actions is less than a budget B.

We extend multi-action RMABs to the robust setting in which the exact transition probabilities are unknown. Instead, the transition dynamics T_n of each arm $n \in [N]$ are determined by a set of parameters $\omega_n \in \Omega_n$, each within a given interval uncertainty $\overline{\underline{\omega}}_n := [\underline{\omega}_n, \overline{\omega}_n]$. Let ω be a given parameter setting such that $\omega_n \in \overline{\underline{\omega}}_n$ for all $n \in [N]$. Let $G(\pi, \omega) = \mathbb{E}[\sum_{t=0}^{H} \beta^t \sum_{n \in [N]} R(\boldsymbol{s}_t^n) \mid \pi, \omega]$ be the planner's expected discounted reward under π and ω, where $\boldsymbol{s}_n^t$ is the state of arm n at time t. Then, *regret* is defined:

$$L(\pi, \omega) = G(\pi^\star_\omega, \omega) - G(\pi, \omega) \ , \tag{1}$$

where $\pi^\star_\omega$ is the optimal reward-maximizing policy under ω. In our robust setting, our objective is to compute a policy $\pi^\dagger$ that minimizes the maximum regret L possible for any realization of ω, i.e.:

$$\pi^\dagger = \min_\pi \max_\omega L(\pi, \omega) \ . \tag{2}$$

This problem is computationally expensive to solve since simply computing a policy π that maximizes the reward $G(\pi, \omega)$ is PSPACE-Hard [Papadimitriou and Tsitsiklis, 1994] even when the T_n are known, i.e., ω is given.

A more tractable approach for computing multi-action RMAB policies π is to utilize the Lagrangian relaxation [Hawkins, 2003, Killian et al., 2021b], reproduced below. For a given ω, the optimal policy $\pi^\star_\omega$ maximizes the constrained Bellman equation:

$$\begin{aligned} J(\boldsymbol{s}) &= \max_{\boldsymbol{A}^c} \left\{ \sum_{n=1}^{N} R(\boldsymbol{s}_n) + \beta \mathop{\mathbb{E}}_{\omega}[J(\boldsymbol{s}') \mid \boldsymbol{s}, \boldsymbol{A}^c] \right\} \\ &\text{where } \boldsymbol{A}^c \subseteq \boldsymbol{A} \\ &\text{s.t.} \sum_{n=1}^{N} \sum_{j=1}^{|\mathcal{A}|} \boldsymbol{A}_{nj} c_j \leq B \qquad \sum_{j=1}^{|\mathcal{A}|} \boldsymbol{A}_{nj} = 1 \ \ \forall n \in [N] \end{aligned} \tag{3}$$

where $\boldsymbol{A}_{nj} = 1$ if the jth action is taken on arm n (else 0) and $c_j \in \mathcal{C}$ is the jth action cost. We then take the Lagrangian relaxation of the budget constraint [Hawkins, 2003], giving:

$$J(\boldsymbol{s}, \lambda^\star) = \min_\lambda \left(\frac{\lambda B}{1 - \beta} + \sum_{n=1}^{N} \max_{j \in |\mathcal{A}|} \{ Q_n(\boldsymbol{s}_n, a_{nj}, \lambda) \} \right) \tag{4}$$

$$\text{where } \ Q_n(\boldsymbol{s}_n, a_{nj}, \lambda) = R(\boldsymbol{s}_n) - \lambda c_j + \beta \mathbb{E}_\omega \left[Q_n(\boldsymbol{s}'_n, a_{nj}, \lambda) \mid \pi^{La}_\omega(\lambda) \right] \ . \tag{5}$$

Here, a_{nj} is the jth action of arm n, Q is the state-action value function, and $\pi^{La}_\omega(\lambda)$ is the optimal policy for a given λ. The key insight is that this relaxation decouples the value functions of the arms, except for the shared λ, i.e., for a given value of λ, all Q_n could be solved via N individual value iterations. However, finding and setting $\lambda := \lambda^\star$ is critical to finding good policies for multi-action RMABs [Killian et al., 2021b, Glazebrook et al., 2011], where $\pi^{La}_\omega(\lambda^\star)$ is used to recover a policy that respects the original budget constraint by solving a knapsack with $Q_n(\boldsymbol{s}_n, a_{nj}, \lambda^\star)$ as values, $\mathcal{C}$ as weights, and the constraints of Eq. 3, then taking the actions according to the Q_n in the solved knapsack.

4 SOLVING ROBUST RMABs

We now build our approach for finding robust RMAB policies, visualized in Fig. 1(a). We use an iterative DO approach which achieves the minimax regret objective of Eq. 2 by casting the optimization problem as a zero-sum game between

[1] Our approaches also easily extend to weakly-coupled MDPs, which allow multiple budget constraints [Hawkins, 2003], as well as to continuous-action RMABs, previously unstudied.

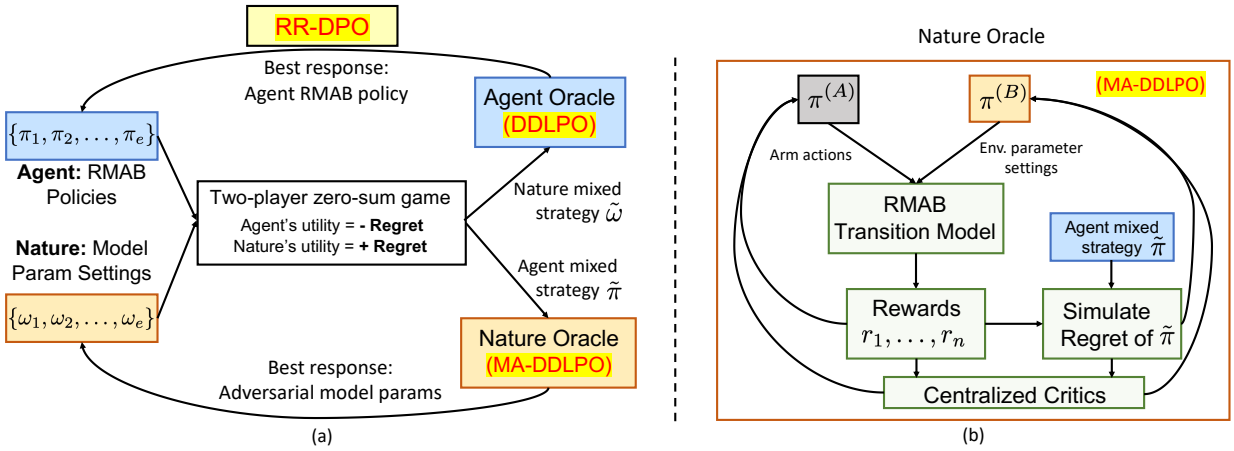

Figure 1: **(a)** Proposed framework for solving the Robust RMAB problem. The main loop follows a DO approach to iteratively compute a minimax regret optimal RMAB policy where each oracle is a novel DRL algorithm for RMABs. **(b)** The nature oracle: a novel multi-agent RL formulation of RMAB, that tackles non-stationarity with a centralized critic.

two players, an *agent* which learns policies π to minimize regret, and an adversarial *nature* which selects environment parameters ω to maximize regret of the agent. In this two-player game, the *pure strategy* space for the agent is the set of all feasible RMAB policies π that respect the budget constraint. The pure strategy space for nature is a continuous, closed set of parameters ω within the given uncertainty intervals. The algorithm maintains a pure strategy set for the agent and nature (Fig. 1(a) left boxes); each iteration, these strategy sets are used to compute a *mixed strategy*—i.e., a probability distribution over pure strategies—Nash equilibrium in a regret game (Fig. 1(a) center). Each oracle then learns a best response against the opponent's mixed strategy to add to its strategy set (Fig. 1(a) right boxes).

The agent oracle's goal is to find an RMAB policy π, or pure strategy, to minimize regret (Eq. 1) given a nature *mixed strategy* $\tilde{\omega}$. That is, the agent minimizes $L(\pi, \tilde{\omega})$ w.r.t. π, while $\tilde{\omega}$ is constant. Recall from Eq. 1 that $L(\pi, \tilde{\omega}) = G(\pi^\star_{\tilde{\omega}}, \tilde{\omega}) - G(\pi, \tilde{\omega})$. Since $\tilde{\omega}$ and $\pi^\star_{\tilde{\omega}}$ are constant, then the first term $G(\pi^\star_{\tilde{\omega}}, \tilde{\omega})$ is also constant. Thus minimizing $L(\pi, \tilde{\omega})$ is equivalent to maximizing the second term $G(\pi, \tilde{\omega})$, which is maximal at $\pi = \pi^\star_{\tilde{\omega}}$. In other words, the agent oracle must compute an optimal reward-maximizing policy w.r.t. $\tilde{\omega}$. Such a reward-maximizing objective aligns with existing RL techniques, but still requires that we address the challenge of learning in the combinatorial state and action spaces of the RMAB. To address this challenge, *we propose a new RL learning method which decomposes the RMAB into N per-arm learning problems and a complementary λ-network learning problem*, which together learn to spend limited budget where it will give the best return, detailed in the next section.

Conversely, the nature oracle seeks to find a parameter setting ω, or pure strategy, that maximizes the agent's regret given a mixed strategy $\tilde{\pi}$, i.e., maximize $L(\tilde{\pi}, \omega)$ with respect to ω, while $\tilde{\pi}$ is fixed. This objective is even more challenging because both $G(\pi^\star_\omega, \omega)$ and $G(\tilde{\pi}, \omega)$ are functions of ω. Most critically, computing $G(\pi^\star_\omega, \omega)$ requires obtaining an optimal policy $\pi^\star_\omega$ as ω changes in the optimization—this amounts to a planning problem in which an agent must learn an optimal policy while the environment changes, controlled by ω, making the nature oracle difficult to solve. Moreover, in the interval uncertainty setting we consider, ω is defined by a space of continuous values; thus nature's pure strategy space is infinite, making the problem even more complex, since it cannot be exhaustively searched.

To tackle this complexity we propose a novel method for implementing the regret-maximizing nature oracle by casting it as a MARL problem. The approach, visualized in Fig. 1(b), trains one auxiliary agent to solve for a policy $\pi^\star_\omega$ (π^A in Fig. 1(b)), needed to compute $G(\pi^\star_\omega, \omega)$ in the regret term, and simultaneously trains a second agent to learn worst-case parameters ω (π^B in Fig. 1(b)) that minimize $G(\tilde{\pi}, \omega)$—together, these will maximize the regret $L(\tilde{\pi}, \omega)$. The non-stationarity is mitigated in this MARL setup by centralized critic networks which allow each agent to include the other's actions in their learned state space. Ultimately, solving a MARL problem requires an RL algorithm to optimize the underlying policy; hence we first introduce our novel RL approach, DDLPO, to solve RMABs (Sec. 4.1) as a part of our agent oracle and then use the algorithm as the backbone of our nature oracle (Sec. 4.2).

4.1 AGENT ORACLE: DEEP RL FOR RMAB

Existing DRL approaches can be applied to the objective in Eq. 3, but, as detailed in Section 2, they fail to scale past trivially sized RMAB problems since the action and state spaces grow exponentially in N. To overcome this, we develop a novel DRL algorithm that instead solves the

Algorithm 1 DDLPO

Input: Initial state $\boldsymbol{s}_0$, nature mixed strategy $\tilde{\omega}$
Parameters: `n_epochs`, `n_subepochs`, `n_steps`

Randomly init. policy net π_{θ_n} for each arm $n \in [N]$
Randomly init. λ-network Λ
Initialize an empty `buffer`
for *epoch* $= 1, 2, \ldots,$ `n_epochs` **do**
  Sample $\lambda = \Lambda(\boldsymbol{s})$
  for *subepoch* $= 1, \ldots,$ `n_subepochs` **do**
    for timestep $t = 1, \ldots,$ `n_steps` **do**
      Sample action $a_n = \pi_{\theta_n}(s_n, \lambda) \quad \forall n \in [N]$
      Add trajectories $(\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}', \lambda)$ to `buffer`
    Update arm policy networks π_{θ_n} via PPO, using tuples in `buffer`
  Update Λ w/ sum discounted costs of final subepoch
return $\pi_{\theta_1}, \ldots, \pi_{\theta_N}$ and Λ

decoupled problem (Eq. 4). The key benefit of decoupling is to render policies and Q values of each arm independent, allowing us to learn N independent networks with *linearly sized state and action spaces, relieving the combinatorial burden of the learning problem*. However, this decoupling approach introduces a new technical challenge in solving the dual objective which maximizes over policies but minimizes over λ, as discussed in Sec. 3.

To solve this, we derive a dual gradient update procedure that iteratively optimizes each objective as follows: (1) holding λ constant, learn N independent policy networks via policy gradient, augmenting the state space to include λ as input, as in Eq. 4; (2) use sampled trajectories from those learned policies as an estimate to update λ towards its minimizing value via a novel gradient update rule. Another challenge is that $\lambda^\star$ of Eq. 4 depends on the current state of each arm—therefore, a key element of our approach is to learn this function $\lambda^\star(\boldsymbol{s})$ concurrently with our iterative optimization, using a neural network we call the λ-network that is parameterized by Λ. To train the λ-network, we use the following gradient update rule.

Proposition 1. *To learn the value λ that minimizes Eq. 4 given a state $\boldsymbol{s}$, the λ-network, parameterized by Λ, should be updated with the following gradient rule:*

$$\Lambda_t = \Lambda_{t-1} - \alpha \left(\frac{B}{1-\beta} + \sum_{n=1}^{N} D_n(s_n, \lambda_{t-1}(\boldsymbol{s})) \right) \quad (6)$$

where α is the learning rate and $D_n(s_n, \lambda)$ is the negative of the expected β-discounted sum of action costs for arm n starting at state s_n under the optimal policy for arm n for a given value of λ.

As D_n lacks a closed form, the key insight we make is that it can be estimated by sampling multiple rollouts of the policy networks of all arms during training. As long as arm policies are trained for adequate time on the given value of λ, the gradient estimate will be accurate, i.e., $D_n(s_n, \lambda_{t-1}(\boldsymbol{s})) \approx -\sum_{k=0}^{K-1} \beta^k c_n^k$ where K is the number of samples collected in an epoch and c_n^k is the action cost of arm n in round k. Moreover, this procedure will converge to the optimal parameters $\Lambda^\star$ if the arm policies are optimal.

Proposition 2. *Given arm policies corresponding to optimal Q-functions, Prop. 1 will lead Λ to converge to the optimal as the number of training epochs and $K \to \infty$.*

Proofs are given in the appendix. One interesting feature of this update rule is that to collect samples that reflect the proper gradient, the RMAB budget must not be imposed *at training time*—rather, the policy networks and λ-network must be allowed to learn to play the Lagrange policy of Eq. 4, which learns to spend the correct budget in expectation, via our iterative update procedure. Therefore, at training time, we sample actions randomly according to the actor network distributions, without imposing the budget constraint. However, *at test time, we always take actions in a way that respects the budget constraint* by following the knapsack procedure described at the end of section 3.

In theory, the policy networks could be trained via any DRL procedure that ensures the above characteristics for training the λ-network. In practice, we train with proximal policy optimization (PPO) [Schulman et al., 2017], a state-of-the-art policy gradient approach. Importantly, PPO is also flexible enough to handle both discrete and continuous actions which is necessary for the nature oracle.

Finally, to enable our iterative, dual-update procedure in practice, we need a mechanism to both (1) explore new arm policy actions after an update to Λ, then (2) exploit learned policy actions to develop good gradient estimates for Λ. We navigate this important trade-off by adding an entropy regularization term to the policy networks losses, controlled via a cyclical temperature parameter. We call our algorithm Deep Distributed Lagrange Policy Optimization (DDLPO), provide pseudocode in Algorithm 1, and include more implementation details in the appendix.

4.2 NATURE ORACLE: MULTI-AGENT RL

Armed with a DRL procedure for learning RMAB policies, we now develop the MARL procedure to implement the nature oracle. Recall the challenge of the nature oracle is to jointly optimize a policy $\pi_\omega^\star$ and model parameters ω. We propose to solve this optimization using MARL, designed to handle this form of non-stationarity [Lowe et al., 2017] via centralized critics. The procedure is visualized in Fig. 1(b).

To implement the nature oracle, we introduce two agents A and B, where A's goal is to optimize the RMAB policy $\pi_\omega^\star$ and B's goal is to find parameters ω that maximize regret

of the current agent mixed strategy $\tilde{\pi}$. We define a shared transition function $T : \mathcal{S} \times \mathcal{A}_A \times \mathcal{A}_B \to \mathcal{S}$. Here, $\mathcal{A}_A$ is the action space of the underlying multi-action RMAB. At a given state $\boldsymbol{s}$, the action space $\mathcal{A}_B$ defines for agent B actions ω which, in general, depend on $\boldsymbol{s}$. That is, at each step, agent B will select environment parameters ω, and thus transition probabilities that will influence the outcome of agent A's actions. We adopt the centralized critic idea from multi-agent PPO [Yu et al., 2021] to our RMAB setting to create MA-DDLPO. A notable strength of our MARL approach is that it allows the discrete-space policy of agent A and the continuous-space policy of agent B to be learned by separate networks, simplifying training compared to an alternative combined-network approach. Moreover, our choice to use PPO offers a convenient way to learn both types of policies as separate networks, while utilizing a single framework of update rules.

A critical step is then to define the rewards for agents A and B to match their objectives. Since agent A's objective is to find $\pi^\star_\omega$, it adopts the reward defined by the underlying RMAB, i.e., $R^{(A)}(\boldsymbol{s}) = \sum_{n=1}^{N} R_n(\boldsymbol{s})$. However, agent B's objective is to learn the regret-maximizing parameters ω. This objective is challenging because it requires computing and optimizing over the returns of the fixed input policy $\tilde{\pi}$ with respect to all possible ω, which is in general non-convex. In practice, to estimate the returns of $\tilde{\pi}_\omega$, we execute a series of roll-outs against agent B's current action. That is, given $\boldsymbol{s}$ at a given round, we sample an action from $\tilde{\pi}_\omega$ and the next state $\boldsymbol{s'}$, and define the *regret-based* reward of agent B, as $R^{(B)} = \sum_{n=1}^{N} R_n(s_n) - \frac{1}{Y}\sum_{y=1}^{Y} r_y^{\tilde{\pi},\omega}$, where $r_y^{\tilde{\pi},\omega}$ is the reward from each of Y one-step Monte Carlo simulations of the mixed strategy $\tilde{\pi}$ in ω.

To train the policies, agent A has the same policy network architecture as DDLPO, i.e., N discrete policy networks and one λ-network, and the agent B actor network is a single continuous-action policy network. Since agent A and B have separate reward functions, they have their own critic networks, but these critics are *centralized* in that they both take the actions of the other as input. Other than the centralized critic, agent A is trained the same way as DDLPO, and agent B is trained in a standard PPO fashion. In practice, to ensure good gradient estimates for agent A's λ-network in MA-DDLPO, we keep agent B's network—and thus the environment—constant between Λ updates, updating B's network with the same frequency as the λ-network updates. Pseudocode for MA-DDLPO and further details of its implementation are given in the appendix.

4.3 MINIMAX REGRET RMAB DOUBLE ORACLE

We now have all the pieces we need to run our robust algorithm Robust RMABs via Deep Policy Oracles (RR-DPO), visualized in Fig. 1(a), with pseudocode presented in Algorithm 2, adapted from the MIRROR framework [Xu et al., 2021]. We use DDLPO to instantiate the agent oracle, MA-DDLPO for the nature oracle, and run RR-DPO until the improvement in value for each player is within a tolerance ε or until a set number of iterations.

Algorithm 2 RR-DPO

Input: Environment simulator and parameter uncertainty intervals $\overline{\omega}_n$ for all $n \in [N]$
Parameters: Convergence threshold ε
Output: Agent mixed strategy $\tilde{\pi}$

$\Omega_0 = \{\omega_0\}$, with ω_0 selected at random
$\Pi_0 = \{\pi_{B_1}, \pi_{B_2}, \ldots\}$, where π_{B_i} are baseline and heuristic strategies
for epoch $e = 1, 2, \ldots$ **do**
  Solve for $(\tilde{\pi}_e, \tilde{\omega}_e)$, mixed Nash equilibrium of regret game with strategy sets Ω_{e-1} and Π_{e-1}
  $\pi_e = \text{DDLPO}(\tilde{\omega}_e)$
  $\omega_e = \text{MA-DDLPO}(\tilde{\pi}_e)$
  $\Omega_e = \Omega_{e-1} \cup \{\omega_e\}, \Pi_e = \Pi_{e-1} \cup \{\pi_e\}$
  if $L(\tilde{\pi}_e, \omega_e) - L(\tilde{\pi}_{e-1}, \tilde{\omega}_{e-1}) \leq \varepsilon$ and $L(\pi_e, \tilde{\omega}_e) - L(\tilde{\pi}_{e-1}, \tilde{\omega}_{e-1}) \leq \varepsilon$ **then**
    break
return $\tilde{\pi}_e$

We now establish conditions under which RR-DPO converges to the minimax regret optimal policy in finite iterations. For two-action RMAB, asymptotically, $\pi^{\text{La}}_\omega \to \pi^\star_\omega$. Thus, assuming each oracle returns true best responses, and under an analytical condition that is straightforward to achieve, i.e., finite pure strategy sets:[2]

Proposition 3. *RR-DPO converges in a finite number of steps to the minimax regret-optimal policy.*

In addition, we empirically verify that good policies are found for the multi-action case, and that RR-DPO converges using our continuous-strategy-space nature oracle. Further, we show that a policy that maximizes reward assuming a fixed parameter set can incur arbitrarily large regret when the parameters are changed (proofs in appendix).

Proposition 4. *In the Robust RMAB problem with interval uncertainty, the max regret of a reward-maximizing policy can be arbitrarily large compared to a minimax regret-optimal policy.*

5 EXPERIMENTAL EVALUATION

We first experimentally demonstrate the importance of robust planning in the presence of uncertainty using a hand-crafted synthetic domain (inspired by Prop. 4). We then evaluate our algorithm on two challenging real-world-inspired public health planning scenarios which demonstrate the capability of our robust RMAB framework.

[2] Straightforward to achieve for nature oracle via discretization.

We compare RR-DPO against five baselines. These baselines include three variations of the reward-maximizing approach from Hawkins [2003], which, given fixed model parameters ω, at each step computes a Lagrange policy, then chooses actions following the knapsack procedure described section 3. The three variations are pessimistic (**HP**), mean (**HM**), and optimistic (**HO**), which assume the model parameters are set at the lower bound, mean, and upper bound of the given intervals for each arm. We also implement **RLvMid**, which *learns* (rather than computes) a policy via DDLPO assuming *mean* parameters, and **Rand**, which acts randomly to fill the budget. All results are averaged over 50 random seeds and were executed on a cluster running CentOS with Intel(R) Xeon(R) CPU E5-2683 v4 @ 2.1 GHz with 8GB of RAM using Python 3.7.10. Our DDLPO implementation builds on OpenAI Spinning Up [Achiam, 2018] and RR-DPO builds on the MIRROR implementation [Xu et al., 2021], computing Nash equilibria using Nashpy 0.0.21 [Knight and Campbell, 2018]. Code is available in the supplement and hyperparameter settings are in the appendix.

5.1 EXPERIMENTAL DOMAINS

Synthetic demonstrates that reward-maximizing policies (RLvMid, HP, HM, HO) may incur large regret in the presence of uncertainty. There are three binary-action arm types $\{U, V, W\}$, each with $\mathcal{C} = \{0, 1\}$, $\mathcal{S} = \{0, 1\}$, $R(s) = s$, and the following transition matrix, with rows and columns corresponding to actions and next states, respectively:

$$T^n_{s=0} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}, \; T^n_{s=1} = \begin{bmatrix} 1.0 & 0.0 \\ 1 - p_n & p_n \end{bmatrix}$$

$$p_U \in [0.00, 1.00],\, p_V \in [0.05, 0.90],\, p_W \in [0.10, 0.95]$$

When an arm is at $s = 0$, each action has equal impact on the state transition. When the arms are at $s = 1$, selecting arms with high p_n is optimal. This implies that policies can be specified by the order in which arms would be acted on, when they are in state $s = 1$. Accordingly, $\pi_{HP} = [W, V, U]$, $\pi_{HM} = [W, U, V]$, and $\pi_{HO} = [U, W, V]$. However, observe that there exist values of p_n that can make each of the reward-maximizing policies incur large regret, e.g., for π_{HO} $p_U = 0.0, p_V = 0.9, p_W = 0.1$ would induce an optimal policy $[V, W, U]$, which is the reverse of π_{HO}.

ARMMAN is a real-world *maternal healthcare intervention problem* modeled as a binary-action RMAB [Biswas et al., 2021]. The goal is to select a subset of mothers each week to intervene on to encourage engagement with tailored automated maternal health messaging. The behavior of enrolled women is modeled by an MDP with three states: Self-motivated, Persuadable, and Lost Cause. We use the summary statistics given in their paper and assume uncertainty intervals of 0.5 centered around the transition parameters, resulting in 6 uncertain parameters per arm (details in appendix). Similar to the setup by Biswas et al. [2021], we assume 1:1:3 split of arms with high, medium, and low probability of increasing their engagement upon intervention. In our experiments, we scale the value of N in multiples of 5 to keep the same split of arm categories of 1:1:3.

SIS Epidemic Model is a discrete-state model in which arms represent distinct geographic regions and each member of an arm's population of size N_p is either (**S**)usceptible to or (**I**)nfected with an infectious disease. Such models have been the subject of increased interest following the COVID-19 pandemic [Hinch et al., 2021, Kerr et al., 2021], and will represent a large-state and multi-action experimental domain. In our model, the count of **S** members of the population is the state of each arm. Each arm's SIS model is defined by parameters κ, the average number of contacts per round, and r_{infect}, the probability of infection given contact with an **I** member. Details on computing discrete state transition probabilities from these parameters are derived from Yaesoubi and Cohen [2011] and given in the appendix. We introduce three intervention actions $\{a_0, a_1, a_2\}$ with costs $c = \{0, 1, 2\}$. Action a_0 represents no action, a_1 represents messaging about physical distancing (divides κ by a_1^{eff}), and a_2 represents distributing face masks (divides r_{infect} by a_2^{eff}). We impose the following uncertainty intervals: $\kappa \in [1, 10]$, $r_{infect} \in [0.5, 0.99]$, $a^{eff}_{\{1,2\}} \in [1, 10]$.

5.2 PERFORMANCE OF RR-DPO

First, we evaluate the performance of the algorithms in uncertain environments. We compute the regret of an agent's pure strategy π against a nature pure strategy ω as the difference in the average reward obtained by π against ω and the average reward of the best strategy in the experiment against ω. The average reward is the discounted sum of rewards over all arms for a horizon of length 10, over 25 simulations. In each setting, DO runs for 6 epochs, using 100 rollout steps and 100 training epochs for each oracle. After completion, each baseline strategy is evaluated by querying the nature oracle for the best response against that strategy, then computing max regret against all ω. The regret of RR-DPO is computed as the utility of the agent mixed strategy returned by the DO over the two-player regret game.

Fig. 2(a–f) shows RR-DPO has the lowest regret, beating the baselines in all domains. (a,b) shows results on the synthetic domain, demonstrating our approach can reduce regret by ~50% against the benchmarks, across various values of N and B. Moreover, as B increases, the regret incurred may increase, since higher budget implies better reward potential for the optimal policy; however, the regret for RR-DPO remains small even as B grows. Similarly, for the ARMMAN domain (c,d), a challenging domain adapted from a real-world dataset, our algorithm performs consistently better than the baselines, achieving regret that is around 50% lower than the best baselines. In the SIS domain (e–f),

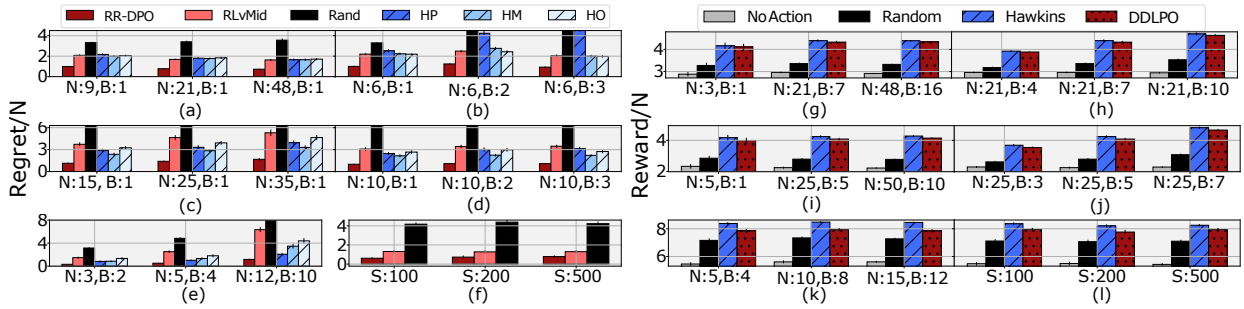

Figure 2: **(a–f)** Maximum policy regret of RR-DPO in robust setting for Synthetic (a,b), ARMMAN (c,d) and SIS (e,f) domains. Lower is better. Synthetic is scaled by 3 and ARMMAN by 5 to maintain the distributions of arm types specified in Section 5. (e) uses $S = 50$ and (f) uses $N = 5, B = 4$. RR-DPO beats all baselines by a large margin across various parameter settings. **(g–l)** Policy returns of DDLPO for reward-maximizing setting (agent oracle) for synthetic (g,h), ARMMAN (i,j), and SIS (k,l) domains. Higher is better. (k) uses $S = 50$ and (l) uses $N = 5, B = 4$. DDLPO is competitive across parameter settings.

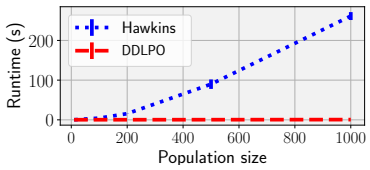

Figure 3: The poor scaling of query time of the Hawkins baseline compared to DDLPO, as discussed in Section 5, even for relatively small problem sizes ($N = 10, B = 2$).

another real-world planning setting with a larger state space and multiple actions, our results are robust across parameter settings. Importantly, this holds even as we increase the state space from $S = 100$ to 500 (f), in which running the Hawkins baseline becomes prohibitively expensive.

Finally, we run sensitivity analyses of the algorithms against H and the size of the uncertainty sets, given in the appendix. When H varies from 10 to 100, RR-DPO maintains very low regret, while competitor regret as much as doubles, increasing RR-DPO's relative improvement as high as ~60%. Similar results are obtained when varying the uncertainty intervals between 0.25, 0.5 and 1.0 times their widths from the experiments in Fig. 2, with RR-DPO always dominating.

5.3 PERFORMANCE OF DDLPO

We also evaluate the performance of DDLPO, our novel DRL approach to find reward-maximizing policies for multi-action RMABs, which implements our agent oracle. We compare against **No Action** and **Random** baselines as well as the computationally intensive solution by Hawkins which computes the Lagrange policy, but which requires exact model parameters and discrete states/actions. Hawkins upper bounds DDLPO for small discrete problems since it is exact whereas DDLPO learns the Lagrange policy from samples. Each experiment is a traditional reward-maximizing RMAB instantiated with a random sample of valid parameter settings for each seed.

Fig. 2(g–l) shows DDLPO achieves reward comparable to the Hawkins algorithm and significantly better than random, providing insight into the success of our RR-DPO approach which DDLPO enables, and showing promise for DDLPO as an algorithm of general interest. In the synthetic domain (g,h), DDLPO learns to act on the 33% of arms who belong to category W. The mean reward of DDLPO almost matches that of Hawkins algorithm as N scales with a commensurate budget (g). As we fix N and vary the budget (h), the optimal policy accumulates more reward, and DDLPO almost equals the optimal. We observe similar results on the ARMMAN domain (i,j), a challenging real-world health problem. On the SIS domain (k,l), the strong performance of DDLPO holds in a multi-action setting even as we increase the number of states from 50 to 500 (l).

Moreover, DDLPO beats Hawkins computationally: in Fig. 3, a single rollout (10 rounds) of Hawkins takes ~100 seconds when there are 500 states, scaling quadratically in general. This demonstrates that it would be prohibitive to run Hawkins in the loop of RR-DPO, since agent policies are evaluated thousands of times to compute the regret matrices. For just 25 simulations, computation would take ~42 minutes to evaluate one cell in the regret matrix, which is of size $|\Pi| \times |\Omega|$.

6 CONCLUSION

We address a key limitation blocking RMABs from many real-world settings: that arm dynamics are not known precisely. To plan safe, effective policies, robust approaches accounting for uncertainty are essential, which we give in RR-DPO, enabled by DDLPO, a novel deep-RL algorithm for RMABs of general interest. We hope our contributions bring us closer to deploying RMABs for real-world impact.

References

J. Achiam. Spinning Up in Deep Reinforcement Learning, 2018. URL https://github.com/openai/spinningup.

D. Adelman and A. J. Mersereau. Relaxations of weakly coupled stochastic dynamic programs. *Oper Res*, 56(3): 712–727, 2008.

K. Avrachenkov and V. S. Borkar. Whittle index based Q-learning for restless bandits with average reward. *arXiv preprint arXiv:2004.14427*, 2020.

S. Bagheri and A. Scaglione. The restless multi-armed bandit formulation of the cognitive compressive sensing problem. *IEEE Trans. Signal Process.*, 63(5):1183–1198, 2015.

A. Biswas, G. Aggarwal, P. Varakantham, and M. Tambe. Learn to intervene: An adaptive learning policy for restless bandits in application to preventive healthcare. In *IJCAI*, 2021.

D. Braziunas and C. Boutilier. Minimax regret based elicitation of generalized additive utilities. In *UAI*, 2007.

J. Fu, Y. Nazarathy, S. Moka, and P. G. Taylor. Towards q-learning the whittle index for restless bandits. In *ANZCC*, pages 249–254. IEEE, 2019.

T. Gafni and K. Cohen. Learning in restless multi-armed bandits via adaptive arm sequencing rules. *IEEE Trans. Automat. Contr.*, 2020.

H. Gilbert and O. Spanjaard. A double oracle approach to minmax regret optimization problems with interval data. *Eur. J. Oper. Res.*, 262(3):929–943, 2017.

K. D. Glazebrook, D. Ruiz-Hernandez, and C. Kirkbride. Some indexable families of restless bandit problems. *J. Appl. Probab*, 38(3):643–672, 2006.

K. D. Glazebrook, D. J. Hodge, and C. Kirkbride. General notions of indexability for queueing control and asset management. *Ann Appl Probab*, 21(3):876–907, 2011.

Y. Gocgun and A. Ghate. Lagrangian relaxation and constraint generation for allocation and advanced scheduling. *Comput Oper Res*, 39(10):2323–2336, 2012.

LLC Gurobi Optimization. Gurobi optimizer reference manual, 2021. URL http://www.gurobi.com.

J. T. Hawkins. *A Langrangian decomposition approach to weakly coupled dynamic optimization problems and its applications*. PhD thesis, Massachusetts Institute of Technology, 2003.

R. Hinch, W. J. M. Probert, A. Nurtay, M. Kendall, C. Wymant, M. Hall, et al. Openabm-covid19—an agent-based model for non-pharmaceutical interventions against covid-19 including contact tracing. *PLoS Comput. Biol.*, 17(7):e1009146, 2021.

D. J. Hodge and K. D. Glazebrook. On the asymptotic optimality of greedy index heuristics for multi-action restless bandits. *Adv Appl Probab*, 47(3):652–667, 2015.

Y. H. Jung and A. Tewari. Regret bounds for thompson sampling in episodic restless bandit problems. In *NeurIPS*, pages 9005–9014, 2019.

Y. H. Jung, M. Abeille, and A. Tewari. Thompson sampling in non-episodic restless bandits. *arXiv preprint arXiv:1910.05654*, 2019.

C. C. Kerr, R. M. Stuart, D. Mistry, R. G. Abeysuriya, K. Rosenfeld, G. R. Hart, et al. Covasim: an agent-based model of covid-19 dynamics and interventions. *PLoS Comput. Biol.*, 17(7):e1009149, 2021.

E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song. Learning combinatorial optimization algorithms over graphs. *NeurIPS*, 30, 2017.

J. A. Killian, A. Biswas, S. Shah, and M. Tambe. Q-learning Lagrange policies for multi-action restless bandits. In *KDD*, 2021a.

J. A. Killian, A. Perrault, and M. Tambe. Beyond "To act or not to act": Fast Lagrangian approaches to general multi-action restless bandits. In *AAMAS*, 2021b.

V. Knight and J. Campbell. Nashpy: A Python library for the computation of Nash equilibria. *Journal of Open Source Software*, 3(30):904, 2018. doi: 10.21105/joss.00904. URL https://doi.org/10.21105/joss.00904.

W. Kool, H. Van Hoof, and M. Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

M. Lanctot, V. Zambaldi, A. Gruslys, A. Lazaridou, K. Tuyls, J. Pérolat, et al. A unified game-theoretic approach to multiagent reinforcement learning. *NeurIPS*, 30:4190–4203, 2017.

J. Le Ny, M. Dahleh, and E. Feron. Multi-uav dynamic routing with partial observations using restless bandit allocation indices. In *American Control Conference*, pages 4220–4225. IEEE, 2008.

E. Lee, M. S. Lavieri, and M. Volk. Optimal screening for hepatocellular carcinoma: A restless bandit model. *Manuf. Serv. Oper. Manag.*, 21(1):198–212, 2019.

S. Li, Y. Wu, X. Cui, H. Dong, F. Fang, and S. Russell. Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient. In *AAAI*, volume 33, pages 4213–4220, 2019.

R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *arXiv preprint arXiv:1706.02275*, 2017.

A. Mate, J. A. Killian, H. Xu, A. Perrault, and M. Tambe. Collapsing bandits and their application to public health interventions. In *NeurIPS*, 2020.

H. B. McMahan, G. J. Gordon, and A. Blum. Planning in the presence of cost functions controlled by an adversary. In *ICML*, pages 536–543, 2003.

K. Nakhleh, S. Ganji, P.-C. Hsieh, I.-H. Hou, and S. Shakkottai. NeurWIN: Neural Whittle index network for restless bandits via deep RL. In *NeurIPS*, 2021.

T. H. Nguyen, A. Yadav, B. An, M. Tambe, and C. Boutilier. Regret-based optimization and preference elicitation for Stackelberg security games with uncertainty. In *AAAI*, pages 756–762, 2014.

C. H. Papadimitriou and J. N. Tsitsiklis. The complexity of optimal queueing network control. In *Structure in Complexity Theory*, pages 318–322. IEEE, 1994.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, et al. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, pages 8024–8035. 2019.

L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta. Robust adversarial reinforcement learning. In *ICML*, pages 2817–2826. PMLR, 2017.

Y. Qian, C. Zhang, B. Krishnamachari, and M. Tambe. Restless poachers: Handling exploration-exploitation tradeoffs in security domains. In *AAMAS*, pages 123–131, 2016.

D. Ruiz-Hernández, J. M. Pinar-Pérez, and D. Delgado-Gómez. Multi-machine preventive maintenance scheduling with imperfect interventions: A restless bandit approach. *Comput Oper Res*, 119:104927, 2020.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

H. Song, H. Jang, H. H. Tran, S. Yoon, K. Son, D. Yun, et al. Solving continual combinatorial selection via deep reinforcement learning. *arXiv preprint arXiv:1909.03638*, 2019.

S. Wang, L. Huang, and J. Lui. Restless-ucb, an efficient and low-complexity algorithm for online restless bandits. *NeurIPS*, 33:11878–11889, 2020.

R. R. Weber and G. Weiss. On an index policy for restless bandits. *J Appl Probab*, 27(3):637–648, 1990.

P. Whittle. Restless bandits: Activity allocation in a changing world. *J Appl Probab*, pages 287–298, 1988.

L. Xu, A. Perrault, F. Fang, H. Chen, and M. Tambe. Robust reinforcement learning under minimax regret for green security. In *UAI*, 2021.

R. Yaesoubi and T. Cohen. Generalized Markov models of infectious disease spread: A novel framework for developing dynamic health policies. *Eur. J. Oper. Res.*, 215(3): 679–687, 2011.

F. Yang, A. Zhu, S. Zhao, Y. Yang, and X. Luo. Optimal task offloading in fog-enabled networks via index policies. In *GlobalSIP*, pages 688–692. IEEE, 2018.

C. Yu, A. Velu, E. Vinitsky, Y. Wang, A. Bayen, and Y. Wu. The surprising effectiveness of MAPPO in cooperative, multi-agent games. *arXiv preprint arXiv:2103.01955*, 2021.

A PROOFS

A.1 PROOF OF PROPOSITION 1

Proposition 1. *To learn the value λ that minimizes Eq. 4 given a state $\boldsymbol{s}$, the λ-network, parameterized by Λ, should be updated with the following gradient rule:*

$$\Lambda_t = \Lambda_{t-1} - \alpha \left(\frac{B}{1-\beta} + \sum_{n=1}^{N} D_n(s_n, \lambda_{t-1}(\boldsymbol{s})) \right) \quad (6)$$

where α is the learning rate and $D_n(s_n, \lambda)$ is the negative of the expected β-discounted sum of action costs for arm n starting at state s_n under the optimal policy for arm n for a given value of λ.

Proof. The gradient update rule is derived by taking the gradient of Eq. 4 with respect to λ, which has two main terms, $\lambda B/(1-\beta)$, and the sum over Q_n, the Q-functions with respect to λ. Looking more closely at Q_n, the only terms which are a function of λ are the costs of actions taken by the policy that Q_n implies, i.e., terms $-\lambda c_j$. Thus, the gradient of Q_n is the negative expected discounted sum of costs taken by the optimal policy at the given value of λ, i.e., $\frac{dQ_n}{d\lambda} = -\mathbb{E}[\sum_{t=0}^{H} \beta^t c_{n,t}]$, where $c_{n,t}$ is the cost of the action taken on arm n in round t. □

A.2 PROOF OF PROPOSITION 2

Proposition 2. *Given arm policies corresponding to optimal Q-functions, Prop. 1 will lead Λ to converge to the optimal as the number of training epochs and $K \to \infty$.*

Proof. Eq. 4 is convex in λ, which follows from definition of Q_n, i.e., the max over piece-wise linear functions of λ is also a convex function in λ. Thus the learning task of Λ is also convex. Therefore, all that is required for asymptotic convergence of Λ is that (1) the gradients we estimate via Prop. 1 are accurate, and that (2) all inputs, i.e., all states $\boldsymbol{s}$, are seen infinitely often in the limit. (1) is achieved by the assumption that optimal Q-functions are given, an analytic condition that is achieved in practice by allowing the arm-networks to train for a reasonable number of rounds under a given output of the λ-network, before updating Λ. Specifically, given optimal Q-functions and their corresponding optimal policies, the sampled sums of spent budget from those optimal policies represent an unbiased estimator of each D_n. Note, though that to be an *unbiased* estimator, this relies on not imposing the budget constraint *at training time*, a procedure we carry out in practice.[3] Thus (1) is achieved. (2) is achieved by following a training procedure that uniformly randomly samples start states $\boldsymbol{s}$ for each round of training until convergence. Thus the proposition is established. □

[3]It is critical to note that at test time, *we always impose the budget constraint* — i.e., all of our methods solve the original constrained RMAB problem — they only use the Lagrangian relaxation as a tool to find good policies to the original constrained problem.

A.3 PROOF OF PROPOSITION 3

Proposition 3. *RR-DPO converges in a finite number of steps to the minimax regret-optimal policy.*

Proof. A common strategy for establishing optimal convergence of the double oracle is to show that the pure strategy sets of both players can be exhausted. We can achieve this in our setting under the conditions (1) that each player has a finite strategy set, i.e., is possible to be exhausted and (2) that each oracle gives an optimal best response. Since the agent pure strategy set is already finite, we can achieve (1) by discretizing the nature oracle—in effect by rounding the outputs of the policy network. For (2), for analytical purposes, we make the common assumption that our oracles internally converge to their optimal values. However, since our oracles return approximate policies that respect the budget constraint by taking the knapsack procedure over the learned π_ω^{La} (knapsack procedure described at the end of section 3), to establish optimal oracles, we also need a case where the knapsack procedure over the optimal Lagrange policy π_ω^{La} goes to π_ω^*. This is known to be the case for binary-action RMABs [Weber and Weiss, 1990]. Thus, optimal is convergence in finite iterations is established under the above conditions of a discretized nature, and binary-action RMAB. □

A.4 PROOF OF PROPOSITION 4

Proposition 4. *In the Robust RMAB problem with interval uncertainty, the max regret of a reward-maximizing policy can be arbitrarily large compared to a minimax regret-optimal policy.*

Proof. Consider a binary-action RMAB problem with two arms A and B. Let the reward from each arm be R when the arm is in a *good* state and 0 in a *bad* state. Our problem is to plan the best action with a budget of 1 and horizon of 1. Supposing the initial state is *bad* for each arm, the transition probabilities for the transition matrix for each arm n is $\begin{bmatrix} 1 & 0 \\ 1-p_n & p_n \end{bmatrix}$ where the uncertain variable p_n is constrained to be within $p_A, p_B \in [0, 1]$. Each value in the matrix corresponds to the probability of an arm at state *bad* transitioning to *bad* (column 1) or *good* (column 2) if we take the *passive* (row 1) or *active* action (row 2).

To compute a reward-maximizing policy that does not consider robustness to uncertainty, we must optimize for one

instantiation of the uncertainty set, which requires making one of three assumptions.

- *Case 1:* If we assume $p_A = p_B$, then an optimal policy is to act with probability a_A on arm A and a_B on arm B as long as $a_A + a_B = 1$. W.l.o.g., suppose $a_A \geq a_B$; then nature would set $p_A = 0$ and $p_B = 1$, imposing regret at least $R/2$.
- *Case 2:* If $p_A > p_B$, then the optimal policy would be to always act on arm A with probability $a_A = 1$ and never act on B ($a_B = 0$). Nature would then set $p_A = 0$ and $p_B = 1$ to impose regret R.
- *Case 3:* If $p_A < p_B$, the case is symmetric to Case 2 and result in regret R. Clearly, max regret is minimized when our action is such that $a_A + a_B = 1$; in this setting, we learn this optimal policy only under Case 1. Following Case 2 or 3, the difference between our regret and the minimax regret is $R/2$, which grows arbitrarily higher as $R \to \infty$.

A slight modification to this problem renders Case 1 non-optimal. Let the reward be R when arm A is in a *good* state and $R-1$ for arm B, so the optimal policy learned under the assumption from Case 1 leads to $a_A = 1$ and $a_B = 0$. Then nature could respond with $p_A = 0$ and $p_B = 1$, yielding reward 0 and regret $R-1$, while the minimax regret–optimal policy achieves a minimum reward of $(R-1)/2$ (by playing $a_A = 0.5$ and $a_B = 0.5$ where nature responds with $p_A = 0$ and $p_B = 1$). Thus, the gap again can grow arbitrarily high as $R \to \infty$ provided that $R > 1$. We therefore have that in all cases, any reward-maximizing policy can achieve arbitrarily bad performance in terms of regret. □

B MA-DDLPO: NATURE ORACLE ALGORITHM

We provide the pseudocode for MA-DDLPO in Alg. 3, which is used to implement the nature oracle.

Algorithm 3 MA-DDLPO

Input: Agent mixed strategy $\tilde{\pi}$
Parameters: `n_epochs`, `n_subepochs`, `n_steps`, `n_sims`

Initialize agent A: arm policies $\pi_n^{(A)}$ $\forall n \in [N]$, λ-network Λ, arm critic networks $\phi_n^{(A)}$ $\forall n \in [N]$
Initialize agent B: nature parameter policy $\pi^{(B)}$, critic network $\phi^{(B)}$
Initialize empty `buffer`
for *epoch* $= 1, 2, \ldots,$ `n_epochs` **do**
  Sample $\boldsymbol{s}$ at random
  Sample $\lambda = \Lambda(\boldsymbol{s})$
  for *subepoch* $= 1, \ldots,$ `n_subepochs` **do**
    for $t = 1, \ldots,$ `n_steps` **do**
      Sample agent A action $a_n^{(A)} \sim \pi_n^{(A)}(s_n, \lambda)$ for each $n \in [N]$
      Sample agent B action $\omega^{(B)} \sim \pi^{(B)}$ ($\pi^{(B)}$ may take $\boldsymbol{s}$ as input)
      $r^{(A)}, \boldsymbol{s}' = \text{SIMULATE}(\boldsymbol{s}, \boldsymbol{a}^{(A)}, \omega^{(B)})$
      $\tilde{r} = \text{SIMULATE}(\boldsymbol{s}, \tilde{\pi}(\boldsymbol{s}), \omega^{(B)}, \texttt{n_sims})$ (mean of 1-step rollouts of $\tilde{\pi}$)
      $r^{(B)} = r^{(B)} - \tilde{r}$ (agent regret)
      Store $(\boldsymbol{s}, \boldsymbol{a}^{(A)}, \omega^{(B)}, r^{(A)}, r^{(B)}, \boldsymbol{s}')$ in `buffer`
      $\boldsymbol{s} = \boldsymbol{s}'$
    Update $\pi_n^{(A)}$, $\phi_n^{(A)}$ for all n using trajectories in `buffer`. $\pi_n^{(A)}$ gets $\omega^{(B)}$ as part of state
  Update Λ via discounted trajectories in `buffer`
  Update $\pi^{(B)}, \phi^{(B)}$ from trajectories in `buffer`. $\phi^{(B)}$ gets $a^{(A)}$ as part of state
return $\pi^{(B)}$

C EXPERIMENTAL DOMAIN DETAILS

C.1 ARMMAN

The MDPs in the ARMMAN domain Biswas et al. [2021] have three ordered states representing the level of engagement of the beneficiaries in the previous week. Rewards are better for lower states, i.e., $R(0) = 1, R(1) = 0.5, R(2) = 0$. At each step, the beneficiary may only change by one level, e.g., low-to-medium or high-to-medium but not low-to-high. They also assume that beneficiaries follow one of three typical patterns, A, B, and C, resulting in three MDPs with different transition probabilities. There are two patterns of effects present that differentiate the beneficiary types. (1) For each of the above types, the planner can only make a difference when the patient is in state 1. Type A responds very positively to interventions, but regresses to low reward states in absence. Type B has a similar but less amplified effect, and type C is likely to stay in state 1, but can be prevented from regressing to state 2 when an action is taken. (2) Further, types A and C have only a 10% chance of staying in the high reward state, while type B has a 90% chance of staying there.

We converted these patient types to robust versions where the transition probabilities are uncertain as follows:

$$T^i_{s=0} = \begin{bmatrix} p^i_{000} & 1 - p^i_{000} & 0.0 \\ p^i_{010} & 1 - p^i_{010} & 0.0 \end{bmatrix},$$

$$T^i_{s=1} = \begin{bmatrix} 0.0 & 1 - p^i_{102} & p^i_{102} \\ p^i_{110} & 1 - p^i_{110} & 0.0 \end{bmatrix},$$

$$T^i_{s=2} = \begin{bmatrix} 0.0 & 1 - p^i_{202} & p^i_{202} \\ 0.0 & 1 - p^i_{212} & p^i_{212} \end{bmatrix},$$

where i indexes the type (i.e., A, B or C). We then set each $p^i_{sas'}$ to be in a range of width 0.5 centered on the entries

from each of the A, B, C beneficiary types for $s \in \{1, 2\}$. To add additional heterogeneity to the experiments, for $s = 0$, we set the range to 1.0 so that any beneficiary type can be made to have some non-negligible chance of staying in the good state, rather than only type B beneficiaries. The full set of parameter ranges are given in the Table 1 below.

Param	L	U	L	U	L	U
Type A			Type B		Type C	
p^i_{000}	.00	1	.00	1	.00	1
p^i_{010}	.00	1	.00	1	.00	1
p^i_{102}	.50	1	.35	.85	.35	.85
p^i_{110}	.50	1	.15	.65	.00	.50
p^i_{202}	.35	.85	.35	.85	.35	.85
p^i_{212}	.35	.85	.35	.85	.35	.85

Table 1: Upper (U) and lower (L) parameter ranges for the robust ARMMAN environment.

In all experiments, 20% of arms were sampled from type A, 20% from type B and 60% for type C. To add additional heterogeneity, for each of the 50 random seeds we uniformly sample a sub-range contained within the ranges given in Table 1. In the agent oracle experiments, for each of the 50 random seeds, since these require fully instantiated transition matrices, we uniformly sample each parameter value for each arm according to its type such that the values are contained in the ranges given in Table 1.

C.2 SIS EPIDEMIC MODEL

In this domain, each arm follows its own compartmental SIS epidemic model. Each arm's SIS model tracks whether each of N_p members of a population is in a susceptible (S) or infectious (I) state. This can be tracked with N_p states, since it can be computed how many people are in state I if only the number of people in state S and the population size N_p is known.

To define a discrete SIS model, we instantiate the model given in Yaesoubi and Cohen [2011] section 4.1 with a Δt of 1. We also augment the model to include action effects and rewards. Specifically, $R(N_S) = N_S/N_p$, where N_S is the number of susceptible (non-infected) people. Further, there are three actions $\{a_0, a_1, a_2\}$ with costs $c = \{0, 1, 2\}$. Action a_0 represents no action, a_1 divides the contacts per day κ (λ in Yaesoubi and Cohen [2011]) by a_1^{eff}, and a_2 divides the infectiousness r_{infect} ($r(t)$ in Yaesoubi and Cohen [2011]) by a_2^{eff}. That is, taking action a_1 will *reduce* the average number of contacts per day in a given arm, and taking action a_2 will reduce the probability of infection given contact in a given arm, thus reducing the expected number of people that will become infected in the next round. However, to make this a robust problem, the relative effect sizes of each action for each arm will not be known to the planner, nor will the κ or r_{infect}. We impose the following uncertainty intervals for all arms: $\kappa \in [1, 10]$, $r_{infect} \in [0.5, 0.99]$, $a_1^{eff} \in [1, 10]$, and $a_2^{eff} \in [1, 10]$.

In the robust double oracle experiments, to add additional heterogeneity, for each of the 50 random seeds we uniformly sample a sub-range contained within the ranges given above for each arm. In the agent oracle experiments, for each of the 50 random seeds, since these require fully instantiated transition matrices, we uniformly sample each parameter value for each arm such that the values are contained in the ranges given above.

D HYPERPARAMETER SETTINGS AND IMPLEMENTATION DETAILS

Neural networks: All neural networks in experiments are implemented using PyTorch 1.3.1 Paszke et al. [2019] with 2 fully connected layers each with 16 units and tanh activation functions, and a final layer of appropriate size for the relevant output dimension with an identity activation function. The output of discrete actor networks (i.e., the policy network from the agent oracle, and the policy network of agent A in the nature oracle) pass through a categorical distribution from which actions are randomly sampled at training time, without a budget imposed. It is critical not to impose the budget at training time, so that the budget spent by the optimal policy under a given λ will result in a meaningful gradient for updating the λ-network. The output of continuous actor networks (i.e., agent B in the nature oracle which selects environment parameter settings) instead are passed as the means of Gaussian distributions – with the log standard deviations learned as individual parameters separate from the network – from which continuous actions are sampled at training time. At test time, actions are sampled from both types of networks deterministically. For categorical distributions, we greedily select the highest probability actions. For Gaussian distributions, we act according to the means. All discount factors were set to 0.9. The remaining hyperparameters that were constant for all experiments for the agent and nature oracles are indicated in Table 2. For **Robust Double Oracle** experiments, all agent and nature oracles were run for 100 training epochs. For **Agent Oracle** experiments, DDLPO was run for 100 training epochs for the synthetic and ARMMAN domains and 200 epochs for the SIS domain.

λ-network: Critical to training the λ-network is cyclical control of the temperature parameter that weights the entropy term in the actor loss functions. Recall that the λ-network is only updated every `n_subepochs`. In general, after each update to the λ-network, we want to encourage exploration so that actor networks explore the new part of the state space defined by updated predictions of λ. How-

ever, after `n_subepochs` rounds, we will use the cost of the sampled actor policies as a gradient for updating the λ-network, and that gradient will only be accurate if the actor policy has converged to the optimal policies for the given λ predictions. Therefore, we also want to have little or no exploration in the round before we update the λ-network. In general, we would also like the entropy of the policy network to reduce over time so that the actor networks and λ-networks eventually both converge.

To accomplish both of these tasks, the weight (temperature) of the entropy regularization term in the loss function of the actor network will decay/reset according to two processes. The first process will linearly decay the temperature from some positive, but time-decaying starting value (see next process) τ_t immediately after each λ-network update, down to 0 after `n_subepochs`. The second process will linearly decay the temperature from a maximum τ_0 (*start entropy coeff* in Table 2) down to $\tau_{\min}$ (*end entropy coeff* in Table 2) by the end of training.

We found that it also helps to train the actor network with no entropy and with the λ-network frozen for some number of rounds before training is stopped (*lambda freeze epochs* in Table 2).

Double Oracle: In all experiments in the main text, we initialize the agent strategy list with HO, HM, and HP, and the nature strategy list with pessimistic, mean, and optimistic nature strategies, then run RR-DPO for 6 iterations. This produces a set of 8 agent strategies, 8 nature strategies, a table where each entry represents the regret of each agent pure strategy (row) against each nature pure strategy (column), and an optimal mixed strategy over each set that represents a Nash equilibrium of the minimax regret game given in the table. The regret table is computed by first computing the returns of each agent/nature pure strategy combination, then subtracting the max value of each column from all entries in that column (i.e., the best agent strategy for a given nature strategy gets 0 regret). The regret of RR-DPO is reported as the expected utility corresponding to the Nash equilibrium of the regret game given by the table, once that regret table is normalized to account for the returns of baselines (see next paragraph).

After this main loop completes, we then compute the regret of the baselines by evaluating each baseline policy against each pure strategy in the nature strategy list. Then, we also run the nature oracle against each baseline policy to find a nature strategy that should maximize the regret of that baseline. The regret for each baseline is reported as the max regret against this new nature strategy, as well as all pure nature strategies from the main RR-DPO loop.

Hawkins Baselines: The Hawkins policies are implemented with gurobipy 9.1.2, a Python wrapper for Gurobi (9.0.3) Gurobi Optimization [2021] following the LP given in Hawkins [2003] equation 2.5 to compute λ and $Q(s, a, \lambda)$ for each arm and the integer program in equation 2.12 to select actions.

RLvMid Baseline: We found that RLvMid found effective policies for the nature strategy it was trained against (as evidenced in Figure 2)(a-f), but that that learned policy could be brittle against other nature strategies. This is likely because different nature strategies produce different distributions of states, meaning RLvMid would fit policies well to states seen when planning against the mean nature strategy, but underfit its policies for states seen more often in different distributions. However, the lone RLvMid baseline policy can somewhat correct for this effect by training an ensemble of policies against slight perturbations of the mean nature strategy that adjust the parameter values output by nature by a small ϵ. In all experiments we train 3 RLvMid policies against 3 random perturbations of the mean nature strategy, then report the regret of RLvMid as the minimum of the max regrets returned by any of the 3.

parameter	**value**
agent	
clip ratio	2.0e+00
lambda freeze epochs	2.0e+01
start entropy coeff	5.0e-01
end entropy coeff	0.0e+00
actor learning rate	2.0e-03
critic learning rate	2.0e-03
lambda learning rate	2.0e-03
trains per epoch	2.0e+01
n_subepochs	4.0e+00
nature	
clip ratio	2.0e+00
lambda freeze epochs	2.0e+01
start entropy coeff	5.0e-01
end entropy coeff	0.0e+00
actorA learning rate	1.0e-03
criticA learning rate	1.0e-03
actorB learning rate	5.0e-03
criticB learning rate	5.0e-03
lambda learning rate	2.0e-03
trains per epoch	2.0e+01
n_subepochs	4.0e+00
n_sims	2.5e+01

Table 2: Hyperparameter settings for agent and nature oracles for all experiments.

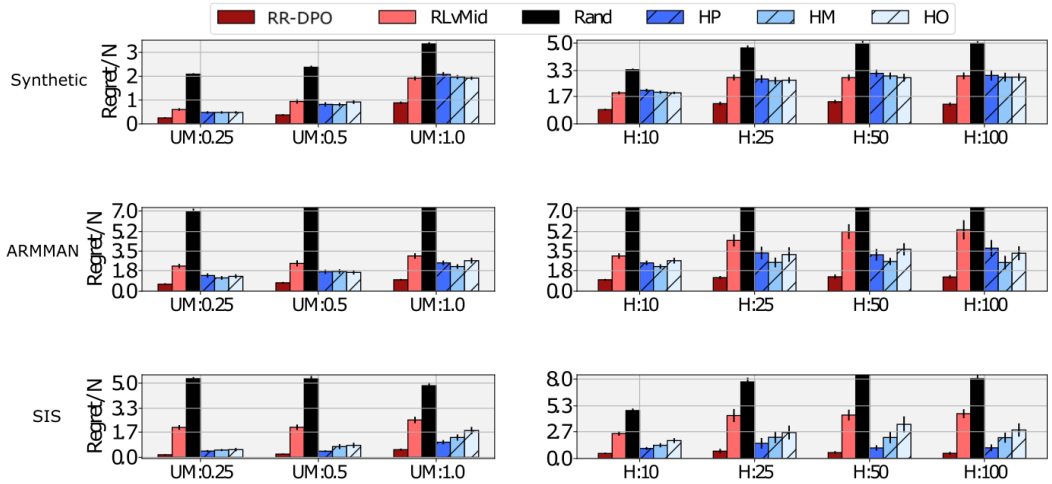

Figure 4: (Left column) varies the uncertainty intervals to be 0.25, 0.5 and 1.0 times their widths (UM = uncertainty multiplier). The gap between our robust RR-DPO method and non-robust methods becomes larger as the uncertainty interval increases, and our robust algorithm RR-DPO always provides the lowest regret policies. (Right column) varies the horizon H in 10, 25, 50, 100. As expected, the gap between RR-DPO and the baselines either stays the same, or increases as H is increased, further demonstrating the robustness of our algorithm to various parameters.