Restless and Uncertain: Robust Policies for Restless Bandits via Deep Multi-Agent Reinforcement Learning (Supplementary material)

Jackson A. Killian[1] **Lily Xu**[1] **Arpita Biswas**[1,2] **Milind Tambe**[1,2]

[1]Computer Science, Harvard University, Cambridge, MA, USA
[2]Center for Research on Computation and Society, Harvard University, Cambridge, MA, USA

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:*

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

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.[1] 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. □

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, i.e., in our case, the arm-networks and λ-network converge optimally. However, since our networks learn the Lagrange-relaxed

[1]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.

version of the problem, some additional tools are needed. Speficially, we must identify conditions in which DDLPO-Act gives policies which approach π^*_ω. This can be achieved in the binary-action setting with $\alpha =$ 'Whittle', which uses a binary search procedure to identify a value of λ such that exactly B arms have $Q_n(a = 1, \lambda) > Q_n(a = 0, \lambda)$, then acting on those arms. This procedure is equivalent to the Whittle index policy, which is asymptotically optimal for binary-action RMABs [Weber and Weiss, 1990]. □

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 DDLPO-ACT SUBROUTINES

Here we provide the integer program which implements `QKnapsack`, one of the action-selection procedures used in Alg. 2 to take actions at test time. `QKnapsack` takes λ and $Q_n(s, a, \lambda)$ from the learned λ-network and arm networks, respectively, and returns the combination of actions that maximizes the sum of Q-values over all arms, subject to the costs of each action $\mathcal{C}$ and the budget constraint B.

$$\max_X \sum_{n=1}^{N} \sum_{j=1}^{|\mathcal{A}|} x_{nj} Q_n(s_n, a_{nj}, \lambda) \tag{2}$$

$$\text{s.t.} \sum_{i=n}^{N} \sum_{j=1}^{|\mathcal{A}|} x_{nj} c_j \leq B \tag{3}$$

$$\sum_{j=1}^{|\mathcal{A}|} x_{nj} = 1 \quad \forall n \in 1...N \tag{4}$$

$$x_{nj} \in \{0, 1\} \tag{5}$$

In Alg. A1, we give the procedure `BinaSearch` which implements a binary search over the learned $Q(\lambda)$-values to find a charge λ for which exactly B arms prefer to act rather than not act. This mimics the Whittle index policy in binary-action settings.

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

Algorithm A1 BinaSearch (for the Whittle Index Policy)

Input: State $\boldsymbol{s}$, arm critic networks $\phi_1, \ldots, \phi_N$, budget B, tolerance ϵ.

```
1: $q_{nj} = \phi_n(s_n, a_{nj}, \lambda = 0) \quad \forall n \in [N], \forall j \in [|\mathcal{A}|]$
2: $lb = 0$
3: $ub = \max_{n \in [N], j \in [|\mathcal{A}|]} \{q_{nj}\}$
4: while $ub - lb > \epsilon$ do
5:     $\lambda = \frac{ub+lb}{2}$
6:     $q_{nj} = \phi_n(s_n, a_{nj}, \lambda) \quad \forall n \in [N], \forall j \in [|\mathcal{A}|]$
7:     if fewer than $B$ arms have $q_{n,j=1} > q_{n,j=0}$ then
8:         $ub = \lambda$          // Charging too much, decrease
9:     else if more than $B$ arms have $q_{n,j=1} > q_{n,j=0}$ then
10:        $lb = \lambda$          // Can charge more, increase
11:    else if exactly $B$ arms have $q_{n,j=1} > q_{n,j=0}$ then
12:        break
13: $\boldsymbol{a} = \boldsymbol{0}$
14: $a_n = 1$ where $q_{n,j=1} > q_{n,j=0}$
15: if $ub - lb \le \epsilon$ then
16:     break ties randomly s.t. $||\boldsymbol{a}||_1 = B$
17: return $\boldsymbol{a}$
```

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 A1 below.

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 A1. In the agent oracle experiments, for each of the

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 A1: Upper (U) and lower (L) parameter ranges for the robust ARMMAN environment.

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 A1.

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: $\kappa \in [1, 10]$, $r_{\mathit{infect}} \in [0.5, 0.99]$, $a_1^{\mathit{eff}} \in [1, 10]$, and $a_2^{\mathit{eff}} \in [1, 10]$.

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 A2. 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. However, 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 A2) down to $\tau_{\min}$ (*end entropy coeff* in Table A2) 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 A2).

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 A2: Hyperparameter settings for agent and nature oracles for all experiments.

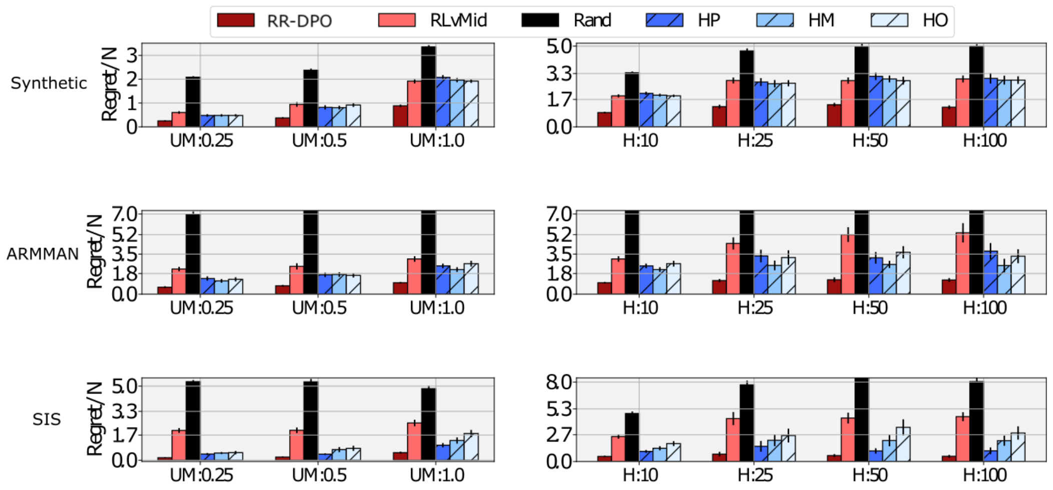

Figure A1: (Left column) varies the uncertainty intervals to be 0.25, 0.5 and 1.0 times their widths (UM = uncertainty multiplier). The gap between our robust RR-DPO method and non-robust methods becomes larger as the uncertainty interval increases, and our robust algorithm RR-DPO always provides the 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.