Reinforcement learning with combinatorial actions for coupled restless bandits

Lily Xu[1], **Bryan Wilder**[2], **Elias B. Khalil**[3], **Milind Tambe**[1]

[1]Harvard University
[2]Carnegie Mellon University
[3]University of Toronto
lily_xu@g.harvard.edu, bwilder@andrew.cmu.edu, khalil@mie.utoronto.ca, milind_tambe@harvard.edu

Abstract

Restless bandits are a useful model for many real-world problems, but existing approaches have been restricted to simplified action settings that allow for tractable solutions. Here, we consider CORMAB, a broader class of problems with *combinatorial actions* that cannot be decoupled across the arms of the restless bandit, requiring direct solving over the joint action space. Leveraging recent advances in embedding trained machine learning models into optimization problems, we propose SEQUOIA, an algorithm for training a reinforcement learning agent to find a good policy that considers long-term reward by embedding a Q-network directly into a mixed-integer program to select one of exponentially many combinatorial actions in each timestep. SEQUOIA builds on the widely used deep Q-network framework but also benefits from a set of enhancements that accelerate training by exploiting the structure of CORMAB. We empirically validate this approach on four novel restless bandit problems with combinatorial constraints—specifically multiple interventions, traveling salesperson, bipartite matching, and capacity constraints—and show that SEQUOIA significantly outperforms a number of myopic policies.

1 Introduction

Reinforcement learning (RL) has made tremendous progress in recent years to be able to solve a wide range of practical problems (Treloar et al. 2020; Marot et al. 2021; Silvestro et al. 2022; Degrave et al. 2022). While successful at dealing with large or infinite state spaces, RL becomes more challenging when the action space is combinatorial. This limitation is pertinent to many real-world sequential decision-making problems, where resource constraints frequently lead to combinatorial action spaces (Dulac-Arnold et al. 2020). Consider for example a sequential resource allocation problem in which public health workers are dispatched to visit patients. The workers have a limited daily budget to maximize patient wellbeing. Even when the number of workers and patients is small, these requirements give rise to an exponentially large combinatorial action space to optimize over.

A suitable framework for modeling such problems is that of *restless multi-armed bandits* (Niño-Mora 2023), which have been applied to settings from clinical trial design (Villar, Bowden, and Wason 2015) to patient interventions (Mate et al. 2022). An "arm" of the restless bandit corresponds to a patient in the aforementioned public health example. At each timestep, an arm is in one of a finite number of states (e.g., representing the health of that individual), and we aim to select actions so that arms move into the more beneficial states. Historically, restless bandit solutions have relied on simplifications that exploit problem structure for tractability. For example, the common Whittle index policy (Whittle 1988) requires that each arm transitions independently and that each action impacts only a single arm (Fig. 1A). This formulation enables a threshold-based top-K policy which decouples the combinatorial action selection by learning policies for each arm independently.

However, real-world applications often involve complicated problem structures that impede decomposing the combinatorial actions (Fig. 1B–D). In public health settings, for example, actions may represent heterogeneous interventions

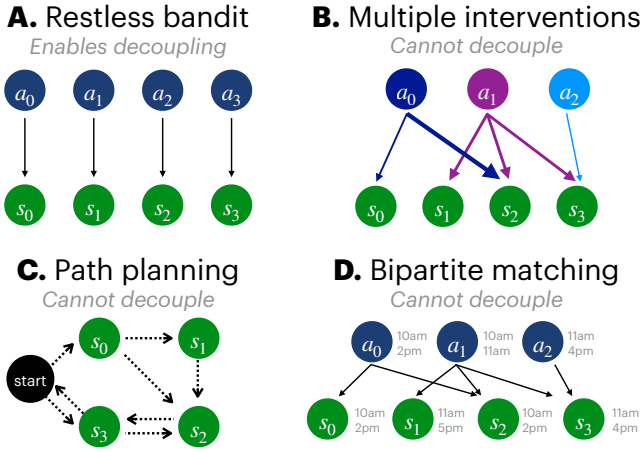

Figure 1: **(A)** Standard restless bandits, where each arm transitions depending on whether it is acted upon, can be solved by threshold-based policies that do not require combinatorial action selection. **(B–D)** However, restless bandit approaches have not been able to address more challenging settings where the combinatorial constraints on the actions cannot be decoupled. Our paper considers this class of strongly coupled restless bandits (CORMAB). We describe these new problem formulations for restless bandits in detail in Section 3.

that may impact not just a single arm but several simultaneously. Consider the problem of planning a path for a health worker to visit patients within a travel time constraint: each action (path) impacts multiple patients and cannot be decomposed further (e.g., into separate edges) while ensuring the overall path is feasible. Or, individuals may be acted on by multiple actions simultaneously (e.g., exposure to different messaging campaigns or nurse visits), with a potentially nonlinear cumulative effect. In these more complex settings, *we can no longer decompose the actions* 2 and must reason about the entire combinatorial structure simultaneously, which existing algorithms cannot accommodate. To that end, we consider in this paper the restless bandit problem with combinatorial actions (which we call COMBRMAB) and provide four problem formulations to highlight the broad applicably of this setting.

The existing literature at the intersection of RL and combinatorial optimization has not considered sequential problems where the action space per step is combinatorial. Rather, existing approaches solve *single-shot* combinatorial optimization problems with RL by decomposing them into iterative choices from a small action set (e.g., iteratively choosing the next node in a traveling salesperson problem) (Khalil et al. 2017; Barrett et al. 2020; Delarue, Anderson, and Tjandraatmadja 2020).

Here, we consider for the first time *sequential* combinatorial settings where the reward comes not from a single-shot action but is incurred after enacting a policy over many timesteps. We leverage recent advances in integrating deep learning with mathematical programming, in which a neural network with ReLU activations can be directly embedded into a mixed-integer program. Incorporated in an RL training procedure, this technique enables our approach to learn the long-term reward using a deep learning approximation of the value function, while tractably optimizing this function over the combinatorial space at each step with a mixed-integer program. We call our algorithm SEQUOIA, for SEQUential cOmbInatorial Actions.

Our paper contributes: (1) four new problem formulations for restless bandits, (2) a general-purpose solution approach that combines deep Q-learning with mathematical programming, and (3) empirical evaluations demonstrating the strength of this approach. Our work opens up a broad class of problem settings for restless bandits with more complex action constraints. Beyond restless bandits, our work provides a proof-of-concept for general RL planning over Markov decision processes with per-timestep combinatorial action spaces.

2 Problem description

We consider offline planning for restless bandits with combinatorial action constraints, where the transition dynamics and reward are known a priori, but the state and action space are too large to be solved directly. Specifically, we consider the setting where the arms are *strongly coupled* so we cannot no longer decouple the arms of the bandit into independent controlled Markov chains using Lagrangian relaxation. We begin by describing the standard restless bandit, then present a general formulation for constrained combinatorial-action RMABs, which we refer to as COMBRMAB.

2.1 Standard restless bandits

The standard restless bandit problem is modeled by a set of J Markov decision processes (MDPs). Each MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, R)$ represents one arm of the restless bandit. The state $s_j \in \mathcal{S}$ of an arm j transitions to next state s'_j depending on the selected action $a_j \in \mathcal{A}$. We use the vector $\boldsymbol{s} \in \mathcal{S}^\times$ to denote the joint state of all J arms, and similarly $\boldsymbol{a} \in \mathcal{A}^\times$ for the joint action space.

The state of the arms evolve according to transition functions $P_j : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, assumed to be known. Traditional restless bandits assume that each action acts on a single arm only, enabling an optimal policy to *decouple* the solution into separate calculations per arm.

Each state $s_j \in \mathcal{S}$ has an associated reward $r_j(s_j)$, and the total reward at each timestep t is the sum of the rewards of the arms: $R^{(t)}(\boldsymbol{s}) = \sum_{j=1}^{J} r_j(s_j)$. Our objective is to maximize the cumulative reward across H timesteps. The traditional restless bandit problem requires selecting a binary action $a_j \in \{0, 1\}$ for each arm, subject to a total budget constraint $\sum_{j=1}^{J} a_j \leq B$.

2.2 Restless bandits with strongly coupled actions

We now consider COMBRMABs, a broader class of problems for restless bandits where actions may be *strongly coupled* due to combinatorial constraints. We consider a set of N actions a_i that each impact the transition probability of some subset of the arms. At each timestep, we must pick some *combinatorial* action over the arms. We require that the action vector $\boldsymbol{a}$ satisfies $\boldsymbol{a} \in \mathcal{C} \subseteq \mathcal{A}^\times$ where the subspace $\mathcal{C}$ is defined by a set of constraints; we provide instantiations of $\mathcal{C}$ in Section 3. Now the transition probabilities may be coupled, with individual actions impacting multiple arms. We denote the joint transition probability as $P^\times : \mathcal{S}^J \times \mathcal{A}^N \times \mathcal{S}^J \to [0, 1]^J$.

From a given state $\boldsymbol{s}$, by Bellman's optimality principle we seek the combinatorial action $\boldsymbol{a}$ that maximizes the expected long-term reward, with discount factor $\gamma \in [0, 1)$:

$$\max_{\boldsymbol{a} \in \mathcal{C}} Q(\boldsymbol{s}, \boldsymbol{a}) \tag{1}$$

$$\text{s.t. } Q(\boldsymbol{s}, \boldsymbol{a}) = R(\boldsymbol{s}) + \gamma \sum_{\boldsymbol{s}' \in \mathcal{S}} P^\times(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') V(\boldsymbol{s}') \qquad \forall \boldsymbol{s} \in \mathcal{S}^\times, \boldsymbol{a} \in \mathcal{C}$$

$$V(\boldsymbol{s}) = \max_{\boldsymbol{a} \in \mathcal{C}} Q(\boldsymbol{s}, \boldsymbol{a}) \qquad \forall \boldsymbol{s} \in \mathcal{S}^\times \ .$$

This value function is difficult to implement as the $\max$ over actions $\boldsymbol{a}$ is taken over a combinatorial number of possible actions, and there are a combinatorial number of constraints (for each possible state $\boldsymbol{s}$). Enumerative methods are therefore intractable, so we will address COMBRMAB of this form using RL.

3 Four motivating problem settings

We introduce four instantiations of COMBRMAB problems which cannot be modeled or solved by standard restless ban-

dit approaches. These widely applicable resource allocation problems consider compounding effects, path planning, bipartite matching, and capacity constraints. For each problem, we provide an overview and a mixed-integer programming formulation; in each case, action selection requires various forms of constrained optimization. To the best of our knowledge, the following problem formulations are all novel for restless bandits.

3.1 Multiple interventions for public health

In public health, different informational campaigns may impact different (groups of) individuals. Interventions could include, for example, radio, ads at bus stations, church events, or fliers. We model this problem as a restless bandit where one action can impact a fixed subset of arms. We propose the first combinatorial setting in which (a) one action impacts multiple arms, and (b) multiple actions may be applied simultaneously to each arm with compounding effects.

Here, each action a_i corresponds to one messaging intervention, each arm j corresponds to a patient, and the state s_j of a patient is equal to 1 if they are actively engaged in a health program and 0 otherwise. For example, consider the graph in Fig. 1B. Then for arm $j = 2$, we have:

$$P_2(s_2 = 0, \boldsymbol{a}, s_2' = 1) = \phi(\omega_{2,s=0} + \omega_{0p_2} a_0 + \omega_{1p_2} a_1)$$
$$P_2(s_2 = 1, \boldsymbol{a}, s_2' = 1) = \phi(\omega_{2,s=1} + \omega_{0q_2} a_0 + \omega_{1q_2} a_1)\ ,$$

where $\phi : \mathbb{R} \to [0, 1]$ is a known link function with known parameters ω that represent the response of each patient to different interventions. For example, $\omega_{0,2,0}$ represents the effect of action a_0 on arm $j = 2$ when it is in the 0-state; the larger $\omega_{0,2,1}$, the more likely this arm will transition to the 1-state if $a_0 = 1$.

In the public health literature, these curves have often been described as S-shaped, such as in a widely used model of smoking cessation (Levy, Bauer, and Lee 2006) The S-shape implies that as individuals are impacted by more actions, they first experience increasing returns in their probability of transition to an improved state (the first part of the S) before plateauing as the effect saturates (the second part). While realistic, S-shaped curves are often NP-hard to approximately optimize because they violate the diminishing returns assumptions required for submodular optimization (Schoenebeck and Tao 2019).

Note that this combinatorial problem formulation generalizes the standard restless bandit: we can recover the original restless bandit setting by considering $N = J$ actions, where the jth action is connected to the jth arm, $\omega_{p_j} = 0$ for all j, and edge weights of 1.

3.2 Path-constrained CORMAB

Suppose that each arm of the restless bandit lies in a network $(\mathcal{V}, \mathcal{E})$. We have one arm at each node $j \in \mathcal{V}$, with edges $(j, k) \in \mathcal{E}$ connecting the nodes. Each (undirected) edge (j, k) has cost d_{jk} corresponding to the distance along the edge. The action is to pick a path along the edges in $\mathcal{E}$ with a constraint on its total length T; we act on each arm whose node we visit along the path. For example, nodes may correspond to physical homes where patients reside and we are trying to schedule an at-home patient intervention (Kergosien, Lenté, and Billaut 2009). The static variant of this problem is a traveling salesperson problem (TSP) with profits (Feillet, Dejax, and Gendreau 2005).

We encode this path-planning problem as flow constraints on a time-unrolled graph. We introduce flow variables $f_{j,k,t}$, which equals 1 if we take the edge from node j to k as the t-th segment of our path and 0 otherwise. Thus for every edge (j, k) in $\mathcal{E}$, we have decision variables $f_{j,k,t}$ and $f_{k,j,t}$ for all $t \in \{1, \ldots, T\}$. Node $\text{⌂} \in \mathcal{V}$ is the source, where we must begin and end the path. Finally, decision variable a_j indicates whether we act on arm j, i.e., we pass through node j.

$$\max_{\mathbf{f}, \boldsymbol{a}} \ Q(\boldsymbol{s}, \boldsymbol{a}) \tag{2a}$$
$$\text{s. t.} \sum_{k:(\text{⌂},k)\in\mathcal{E}} f_{\text{⌂},k,0} = 1 \tag{2b}$$
$$\sum_{k:(k,\text{⌂})\in\mathcal{E}} f_{k,\text{⌂},T} = 1 \tag{2c}$$
$$\sum_{t\in[T]} \sum_{(j,k)\in\mathcal{E}} f_{j,k,t} = T \tag{2d}$$
$$\sum_{k:(j,k)\in\mathcal{E}} f_{k,j,t} = \sum_{k:(k,j)\in\mathcal{E}} f_{j,k,t+1} \qquad \forall j \in \mathcal{V} \tag{2e}$$
$$a_j \le \sum_{t\in[1]} \sum_{k:(j,k)\in\mathcal{E}} f_{j,k,t} \qquad \forall j \in \mathcal{V} \tag{2f}$$
$$a_j \ge f_{j,k,t} \qquad \forall j, k \in \mathcal{V} \tag{2g}$$
$$f_{j,k,t} \in \{0, 1\} \qquad \forall j, k \in \mathcal{V}, t \in [T] \tag{2h}$$
$$a_j \in \{0, 1\} \qquad \forall j \in \mathcal{V} \tag{2i}$$

3.3 Schedule-constrained CORMAB

Now suppose both actions (volunteer health workers) and arms (patients) have limited time windows for scheduling. We must assign workers to patients to form a valid schedule, where workers and patients are matched such that they have mutually agreed upon times. Each worker can only be assigned once. This problem resembles bipartite matching and can be formulated as follows:

$$\max_{\mathbf{x}, \boldsymbol{a}} \ Q(\boldsymbol{s}, \boldsymbol{a}) \tag{3a}$$
$$\text{s.t.} \sum_{j,s} x_{ijs} \le 1 \qquad \forall i \tag{3b}$$
$$\sum_{i} x_{ijs} \le 1 \qquad \forall j, s \tag{3c}$$
$$a_j \le \sum_{i,s} x_{ijs} \qquad \forall j \tag{3d}$$
$$a_j \ge x_{ijs} \qquad \forall i, j, s \tag{3e}$$
$$x_{ijs} \in \{0, 1\} \qquad \forall i, j, s \tag{3f}$$
$$a_j \in \{0, 1\} \qquad \forall j \tag{3g}$$

The scheduling constraints are encoded as binary matrices, where W_{is} and A_{js} denote whether resource i (or patient j) is available at timeslot s. We have decision variables x_{ijs} whenever a worker i and patient j are both available at

timeslot s, so x_{ijs} exists iff $W_{is} = A_{js} = 1$. Then, $x_{ijs} = 1$ indicates the assignment of worker i to arm j at timeslot s, and a_j indicates whether we "pull" arm j. Each worker can only be assigned once.

This problem also generalizes RMABs: each pulling action requires solving a matching problem to assign workers to each action. The standard RMAB problem would correspond to the setting where all workers and all patients are available for all timeslots.

3.4 Capacity-constrained CORMAB

Suppose we have N workers each with a budget constraint b_i, and the J arms each have some cost c_j to call them. This problem is an instance of the generalized assignment problem (Özbakir, Baykasoğlu, and Tapkan 2010), which is NP-hard with LP relaxations that offer a $(1-1/e)$–approximation (Fleischer et al. 2006).

The goal is to assign workers to patients while respecting each worker's capacity constraint. We have a decision variable x_{ij} for all i and j, indicating whether worker i gets assigned to arm j.

$$\max_{\mathbf{x},\boldsymbol{a}} \; Q(\boldsymbol{s}, \boldsymbol{a}) \tag{4a}$$

$$\text{s.t.} \sum_j c_j x_{ij} \leq b_i \qquad \forall i \tag{4b}$$

$$a_j \leq \sum_i x_{ij} \qquad \forall j \tag{4c}$$

$$a_j \geq x_{ij} \qquad \forall i, j \tag{4d}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \tag{4e}$$

$$a_j \in \{0, 1\} \qquad \forall j \tag{4f}$$

The standard RMAB setting would correspond to a single worker ($N = 1$) with a budget $b_i = B$.

As we are motivated by public health interventions, we use these as motivating descriptions in the descriptions, but these problem structures exist in many other applications. Importantly for public health and other applications, these constraints could be further specified incorporate additional desiderata, such as fairness constraints. For example, if we had demographic features associated with each patient, we could encode a requirement to visit at minimum some fraction of each subgroup, such as the most elderly.

4 Solving RMABs with combinatorial actions

We present a novel algorithm, SEQUOIA, which builds on deep Q-learning (DQN) by integrating a mixed-integer program (MIP) for combinatorial action selection. While we focus on the restless bandit problem setting, SEQUOIA generalizes to general sequential planning problems with combinatorial actions, provided that the restrictions on the actions can be represented as constraints in a mixed-integer program.

We start by introducing the basic DQN+MIP framework in Section 4.1, summarized in Algorithm 1. Then, in Section 4.2, we provide a set of strategies to find a good initialization for the Q-network and improve computational efficiency.

Algorithm 1: SEQUOIA for RL with combinatorial actions

Input: Restless bandit instance

Parameter: Epsilon-greedy parameter ϵ, target update frequency C

1: Replay memory $\mathcal{D}$
2: Initialize action–value function Q with weights θ
3: Strategically generate actions, and store state–action samples in memory $\mathcal{D}$
4: Pre-train Q with myopic observed reward, using state–action samples from $\mathcal{D}$
5: Initialize target network $\hat{Q}$: $\theta^- = \theta$
6: Construct MIP with current network weights θ
7: **for** episode $= 1, \ldots, E$ **do**
8: **for** $t = 1, \ldots, T$ **do**
9: With probability ϵ select random action $\boldsymbol{a}^{(t)}$
10: Otherwise, select $\boldsymbol{a}^{(t)} = \arg\max_{\boldsymbol{a}} \text{MIP}(\boldsymbol{s}^{(t)}, \boldsymbol{a}; \theta^-)$
11: Execute action $\boldsymbol{a}^{(t)}$ and observe reward $r^{(t)}$ and next state $\boldsymbol{s}^{(t+1)}$
12: Store transition $(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)}, r^{(t)}, \boldsymbol{s}^{(t+1)})$ in $\mathcal{D}$
13: Sample random minibatch of transitions $(\boldsymbol{s}^{(k)}, \boldsymbol{a}^{(k)}, r^{(k)}, \boldsymbol{s}^{(k+1)})$ from $\mathcal{D}$
14: Set $y^{(k)} = r^{(k)} + \gamma \max_{\boldsymbol{a}} \text{MIP}(\boldsymbol{s}^{(k)}, \boldsymbol{a}; \theta^-)$
15: Gradient descent step on $(y^{(k)} - Q(\boldsymbol{s}^{(k)}, \boldsymbol{a}^{(k)}; \theta))^2$
16: Every C steps, update $\hat{Q} = Q$
17: **return** Q-function Q

4.1 Q-learning with combinatorial actions

Deep Q-learning aims to estimate $Q(\boldsymbol{s}, \boldsymbol{a})$, the long-run value of action $\boldsymbol{a}$ in state $\boldsymbol{s}$, by parameterizing the estimated Q-function with a neural network. It samples (action, next state, reward) sequences via a mix of random actions and on-policy samples (i.e., actions that maximize the current estimate of Q). The samples are used to improve the Q-function via temporal difference (TD) updates which aim to minimize the residual in the Bellman equation.

This framework breaks down when the action space is exponentially large, as in our setting, leading us to modify the standard DQN setup. The first challenge is that typical uses of DQN output Q-values for all actions simultaneously by mapping the state to one network output per action. Since our domain has an exponentially large action space, we instead use DQN to estimate the Q-value of a given state–action *pair*, where the action is represented as a binary vector of length N.

While this allows for a concise representation of the Q-function, it leaves a critical bottleneck: computing the temporal difference loss requires identifying the best action according to the current Q-function (Algorithm 1 lines 10 and 14). That is, it requires solving problems of the form $\max_{\boldsymbol{a}} Q(\boldsymbol{s}, \boldsymbol{a}; \theta)$ where θ denotes the parameters of the Q-network. In the typical DQN, this problem would be solved by evaluating all actions and picking the best one, but brute-force enumeration is infeasible for combinatorial spaces.

Instead, we leverage recent advances that enable encoding a neural network into a mixed-integer program (MIP) to enable optimization (Fischetti and Jo 2018; Dumouchelle

et al. 2022). These methods incorporate an extra set of variables and constraints in the MIP which simulate a forward pass of the neural network to produce Q. Embedding the network in a MIP can be achieved using additional variables and constraints with size linear in the size of the network. In every new episode, we build a new MIP instance which adds components that represent the current state of the policy network Q along with the original constraints on the actions. Then the inner step of $\arg\max_{\boldsymbol{a}}$ (line 10) solves that MIP to pick a combinatorial action $\boldsymbol{a}$ for state $\boldsymbol{s}$ that maximizes the expected long-run return of pair $(\boldsymbol{s}, \boldsymbol{a})$, based on estimates from the neural network Q.

Our algorithm runs DQN on this modified Q-network and best action selection procedure. On top of these key changes to accommodate combinatorial actions, we incorporate standard best practices for DQN (Hessel et al. 2018), including double DQN, prioritized experience replay (Schaul et al. 2016), and gradient clipping. Once the network is trained, at inference time we simply optimize over the MIP encoding of the final Q-network to compute the action to take in each state that is encountered.

4.2 Smart action generation and other computational enhancements

We now introduce some algorithmic enhancements that leverage the structure of our domain to provide an informative initialization for the Q-network, greatly speeding up training and improving the quality of the learned policy.

There are two independent critical computational bottlenecks with integrating neural networks into MIPs. First, solving the MIP at every timestep is extremely expensive: for the discounted future reward estimate in line 14, we must perform that calculation for *every sample in the minibatch* at every timestep. For a training instance with E episodes, time horizon H, and minibatches of size M, we have EHM repeated MIP solves. Even a modest problem setting of $E = 1{,}000$, $H = 20$, and $M = 32$ requires a prohibitive 640,000 solves of the MIP. Second, Q-learning requires diverse samples of the state and action spaces to learn well, but both the state and action are combinatorial.

We design an initialization process, run before the main DQN algorithm, to help alleviate both of these computational bottlenecks. The main idea is to generate cheap but informative samples which provide approximate rewards and a wide diversity of actions, allowing us to warm-start the Q-network much more effectively before training with on-policy samples. We use three strategies for this process.

First, we initialize the Q-network to approximate the single-step expected rewards. This leverages our access to a cheap simulator for the state transitions and immediate rewards: we can draw a large training set of sampled state-reward pairs from the simulator and fit the Q-network to the immediate reward. This provides an informative initialization when we start learning the long-run rewards.

Second, we draw sampled actions according to the implied myopic policy to seed the training process with a set of "reasonable" actions. Specifically, we use the MIP embedding of the Q-network to find the action maximizing the learned approximation of the single-step reward. This allows the method to work out-of-the-box for different settings, without the user having to explicitly derive a MIP formulation for the optimal myopic action. We then train the network using temporal difference updates on the sampled actions and rewards to learn their long-term values.

Third, we introduce diversity into the sampled actions with additional random perturbations. One source of diversity is to randomly flip entries of the myopic action (we call this "perturbed myopic"). Another is to random sample *infeasible* actions by, e.g., starting with the all-ones or all-zeros vector for $\boldsymbol{a}$ and then randomly flipping a small number of entries (isolating the impact of including or knocking-out individual actions). This leverages the property that, in restless bandits, *we can simulate valid state transitions even for infeasible actions* because the state transitions are defined per-arm. Incorporating perturbed and even infeasible actions greatly increases the diversity of potential samples because for some combinatorial constraints it may be difficult to even find a single feasible solution (and otherwise may be difficult to explore parts of the action space). Similar issues arise if the per-step feasible action space is relatively small (e.g., if the budget is small with $N \gg B$). Including examples of infeasible actions provides a more diverse training set for the Q-network, encouraging better generalization even to feasible actions.

Throughout, we incorporate two additional strategies to lower variance and improve computational efficiency. First, we directly calculate the expected one-step reward (instead of using the observed reward after state transition), which reduces variance. Second, we memoize the solutions to MIPs seen multiple times in between updates to the Q-network. As we use experience replay with a reasonably small buffer size (10,000) we likely expect to see repeated samples, enabling us to store the optimal solution to avoid repeated solves. To avoid stale solves, we discard old samples at the same rate that we update the target function $\hat{Q}$.

5 Experiments

We evaluate the performance of SEQUOIA on the four different CORMAB settings from Section 3.

The need for sequential planning in RMABs Myopic policies can perform arbitrarily badly in restless bandits. In Fig. 3, we offer a clear example through a restless bandit with two arms, three states, and budget $B = 1$. Suppose the probability p of transitioning right one state is $p = 1$ when the arm is acted upon and $p = 0$ otherwise (in which case the arm will move leftward). If the arms begin at state $\boldsymbol{s}^{(0)} = (0, 0)$, the myopic policy would always act on arm 1, whereas the optimal policy would be to repeatedly act on arm 2. Thus in these experiments, we consider multi-state settings (specifically with $|\mathcal{S}| = 4$ states) to emphasize the sequential aspect of the problem.

Baselines We evaluate performance against several baselines, including a number of myopic policies that evaluate the best single-step action, ignoring expected future return. We implement an optimal MYOPIC policy as a MIP that di-

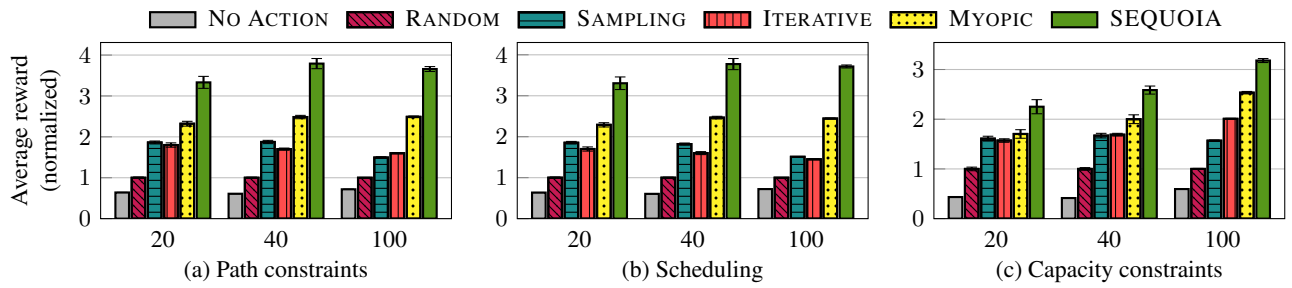

Figure 2: SEQUOIA achieves consistently higher performance compared to the three myopic policies across all problem settings. We evaluate with $J = \{20, 40, 100\}$ arms and $N = \{5, 10, 20\}$ workers for each setting. The vertical axis depicts the average per-timestep reward, normalized by the reward achieved by the RANDOM baseline.

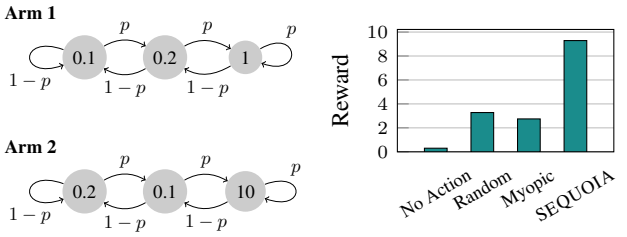

Figure 3: Even in a simple two-arm problem setting with budget $B = 1$, a myopic policy can lead to arbitrarily poor performance in restless bandits. Each arm has three states, with positive reward in each state. Suppose that the probability p of transitioning right one state is $p = 1$ when the arm is acted on and $p = 0$ otherwise. This problem instance leads to the following rewards on the right, where the gap between myopic and SEQUOIA can be arbitrarily large depending on the rewards at the rightmost state.

rectly encodes (as a linear objective) the expected reward of the immediate next state, based on the current state and transition probabilities (and no neural network). To understand the challenge of the problem, we also consider two additional myopic baselines. One is a SAMPLING approach (He et al. 2016) that randomly samples $k = 100$ actions of the $\binom{N}{B}$ possible combinatorial actions and picks the action with the largest expected myopic reward. The final myopic baseline is an ITERATIVE solution construction algorithm that greedily selects one feasible component (an action $i \in [N]$) at a time to build up the full action $\boldsymbol{a}$.

Lower bounds on performance are benchmarked by RANDOM which randomly samples a feasible action, and NO ACTION which always takes the null action $\boldsymbol{a} = \mathbf{0}$. We are not aware of additional RL algorithms from the literature that could be applied to CORMAB; this is due to the large combinatorial action space. For instance, many algorithms in the RLLib library (Liang et al. 2018) support discrete actions, but not combinatorial ones.

Experiment setup For each problem instance, we randomly generate transition probabilities, constraints, and initial states. We ensure consistency by enforcing, across all algorithms, that transition dynamics and initial states across every problem instance (across every episode) are consistent across all methods. We evaluate the expected reward of each algorithm across 50 episodes of length $H = 20$ and average results over 30 random seeds. Runtime analysis and implementation details, including hyperparameters, are in the appendix, along with additional results for the multiple-intervention setting (Section 3.1).

Results SEQUOIA achieves consistently strong performance over the optimal MYOPIC baseline, demonstrating the importance of accounting for long-run return, even in relatively simple MDP settings such as restless bandits. Additionally, note that the optimal MYOPIC approach can only be implemented in settings with simple (e.g., linear or quadratic) constraints and objectives that can be handled by an integer programming solver, whereas SEQUOIA can learn arbitrary objective functions.

To better understand the challenging combinatorial structure of the problem, we look at the myopic baselines which achieve significantly lower performance. For example the ITERATIVE myopic approach performs on average 14.6% lower than optimal MYOPIC—an important takeaway, given that many heuristic approaches for overcoming combinatorial action structure rely on iterative construction heuristics (Khalil et al. 2017; Barrett et al. 2020). The SAMPLING heuristic comes close to MYOPIC with 20 arms, but dramatically falls behind at 100 arms — an unsurprising result, considering the action space jumps many orders of magnitude from roughly $\binom{20}{5} = 15{,}504$ actions to $\binom{100}{20} \approx 5.4 \times 10^{20}$ (without considering feasibility constraints). This result underscores the challenge of effectively exploring combinatorially large action spaces.

To show generality and robustness of SEQUOIA, we used the same network architecture and training procedure for all of the different problem settings. Our method works well even without per-domain hyperparameter tuning, which would improve performance further.

6 Related work

Restless multi-armed bandits (RMABs) generalize multi-armed bandits by introducing arm states that transition depending on whether the arm is acted on. Even when transition probabilities are fully known, computing

an optimal RMAB policy is PSPACE-hard (Papadimitriou and Tsitsiklis 1994) due to the combinatorial state and action space. Heuristic approaches to solve RMABs center around the threshold-based Whittle index policy (Whittle 1988; Weber and Weiss 1990) which uses a Lagrangian relaxation to exploit the fact that the arms are *weakly coupled* (Adelman and Mersereau 2008; Hawkins 2003). However, these relaxations break down in strongly coupled action settings (Ou et al. 2022), our setting here.

Many other solution approaches for RMAB problems are variants of the Whittle index policy, including deep learning to estimate the Whittle index by training a separate network for each arm (Nakhleh et al. 2021); tabular Q-learning (Avrachenkov and Borkar 2022); and explicitly encoding the Bellman update as a MIP to overcome uncertainty in transition probabilities (Wang et al. 2023). Here, we introduce the first solution approach for restless bandits integrating both deep learning and mathematical programming.

RL and combinatorial optimization An iterative heuristic to solve a static combinatorial optimization problem can be represented as a Markov decision process. RL can then be used to obtain a good policy for constructing a feasible solution. By "static", we mean deterministic problems that are fully specified. Instances of this iterative approach include (Khalil et al. 2017; Barrett et al. 2020), with a more complete survey in (Mazyavkina et al. 2021). The action space in such approaches is *not* combinatorial: to construct a solution to a traveling salesperson problem, one needs to select the single next node to visit in every timestep of the decision process. Breaking with this approach, Delarue, Anderson, and Tjandraatmadja (2020) consider combinatorial actions in a capacitated vehicle routing problem, where at every timestep of the decision process a *subset of nodes* that form a tour and respect the vehicle's capacity constraint must be selected. This combinatorial action selection problem is formulated as a MIP whose objective function is the Q-value as estimated by a ReLU neural network.

In the RL literature, it is fairly common to deal with combinatorial *state spaces*, for example AlphaGo (Silver et al. 2016), but combinatorial action spaces have received far less attention. Dulac-Arnold et al. (2015) deal with large discrete (but not combinatorial) action spaces. Their action selection strategy is sub-linear in the number of actions, but this is still prohibitive when the number of actions is exponential as in our setting. He et al. (2016) consider simply sampling a fixed number of actions from the $\binom{N}{B}$ combinatorial action space. Tkachuk et al. (2023) provide a theoretical analysis of RL with combinatorial actions when the value function approximation is linear in the state–action pair. Because the reward functions in many practical applications may be non-linear, we opt for neural network function approximations which are beyond the scope of the aforementioned theory. Brantley et al. (2020) propose an algorithm for budget-constrained continuous action spaces in a tabular RL setting. However, our state spaces are also combinatorial, making a tabular state representation impossible (in addition to our discrete action space). Song et al. (2019) propose to split up combinatorial action selection into iterative non-combinatorial selections; an iterative policy must be learned for this purpose. In contrast, and similar to Tkachuk et al. (2023), we assume oracle access to an optimization solver that can find feasible combinatorial actions at each timestep, removing the need for any iterative decomposition of the combinatorial selection as the latter requires learning an additional policy.

Embedding neural networks in optimization models There has been considerable interest in embedding trained neural networks into larger mathematical optimization models. Two primary use cases are (1) adversarial machine learning problems such as finding perturbations of an input that worsen the prediction maximally (Fischetti and Jo 2018) or verifying network robustness (Tjeng, Xiao, and Tedrake 2018); (2) replacing a function that is difficult to describe analytically with a neural network approximation of it and then optimizing some decision variables over the approximation (Lombardi and Milano 2018). As feed-forward neural networks with ReLU activations are piecewise-linear functions in their inputs, they can be represented using a polynomial-size mixed-integer linear program (Fischetti and Jo 2018; Anderson et al. 2020; Ceccon et al. 2022). Our work falls in the second class of use cases. Closely related to our approach is the work of Delarue, Anderson, and Tjandraatmadja (2020) which focuses on static vehicle routing. While we leverage the same MIP representation of ReLU networks, our focus is on a wide range of coRMAB problems that are inherently sequential rather than static.

7 Conclusion

While much recent work has focused on RL *for* combinatorial optimization, here, we address RL *with* combinatorial optimization (over actions). Specifically, we consider offline planning for restless multi-armed bandits where action constraints prevent decoupling the problem. Beyond solving coupled MDPs such as restless bandits, this approach can also be applied to other general RL problems with per-timestep combinatorial actions as long as the set of feasible actions can be described in a MIP. Future work could explore additional applications enabled by our framework, both in the coRMAB setting or beyond in online combinatorial optimization.

Despite having proposed a set of effective computational strategies that accelerate RL training for coRMAB, we believe more work remains to be done in speeding up the solution of the combinatorial action selection MIP. This subproblem is solved in every timestep of every episode during training and testing; exploring heuristic or approximate solutions would be of great interest. While our contributions are algorithmic and computational, coRMAB could benefit from theoretical analysis, possibly extending the work of Tkachuk et al. (2023) in the linear function approximation setting to the more practical neural network setting.

References

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

Anderson, R.; Huchette, J.; Ma, W.; Tjandraatmadja, C.; and Vielma, J. P. 2020. Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming*, 183(1-2): 3–39.

Avrachenkov, K. E.; and Borkar, V. S. 2022. Whittle index based Q-learning for restless bandits with average reward. *Automatica*, 139: 110186.

Barrett, T.; Clements, W.; Foerster, J.; and Lvovsky, A. 2020. Exploratory combinatorial optimization with reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 3243–3250.

Bou, A.; Bettini, M.; Dittert, S.; Kumar, V.; Sodhani, S.; Yang, X.; Fabritiis, G. D.; and Moens, V. 2023. TorchRL: A data-driven decision-making library for PyTorch. arXiv:2306.00577.

Brantley, K.; Dudik, M.; Lykouris, T.; Miryoosefi, S.; Simchowitz, M.; Slivkins, A.; and Sun, W. 2020. Constrained episodic reinforcement learning in concave-convex and knapsack settings. *Advances in Neural Information Processing Systems*, 33: 16315–16326.

Ceccon, F.; Jalving, J.; Haddad, J.; Thebelt, A.; Tsay, C.; Laird, C. D.; and Misener, R. 2022. OMLT: Optimization & machine learning toolkit. *The Journal of Machine Learning Research*, 23(1): 15829–15836.

Degrave, J.; Felici, F.; Buchli, J.; Neunert, M.; Tracey, B.; Carpanese, F.; Ewalds, T.; Hafner, R.; Abdolmaleki, A.; de Las Casas, D.; et al. 2022. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897): 414–419.

Delarue, A.; Anderson, R.; and Tjandraatmadja, C. 2020. Reinforcement learning with combinatorial actions: An application to vehicle routing. *Advances in Neural Information Processing Systems*, 33: 609–620.

Dulac-Arnold, G.; Evans, R.; van Hasselt, H.; Sunehag, P.; Lillicrap, T.; Hunt, J.; Mann, T.; Weber, T.; Degris, T.; and Coppin, B. 2015. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*.

Dulac-Arnold, G.; Levine, N.; Mankowitz, D. J.; Li, J.; Paduraru, C.; Gowal, S.; and Hester, T. 2020. An empirical investigation of the challenges of real-world reinforcement learning. *arXiv preprint arXiv:2003.11881*.

Dumouchelle, J.; Patel, R.; Khalil, E. B.; and Bodur, M. 2022. Neur2SP: Neural Two-Stage Stochastic Programming. *Advances in Neural Information Processing Systems*, 35.

Feillet, D.; Dejax, P.; and Gendreau, M. 2005. Traveling salesman problems with profits. *Transportation Science*, 39(2): 188–205.

Fischetti, M.; and Jo, J. 2018. Deep neural networks and mixed integer linear optimization. *Constraints*, 23(3): 296–309.

Fleischer, L.; Goemans, M. X.; Mirrokni, V. S.; and Sviridenko, M. 2006. Tight approximation algorithms for maximum general assignment problems. In *SODA*, volume 6, 611–620. Citeseer.

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

He, J.; Ostendorf, M.; He, X.; Chen, J.; Gao, J.; Li, L.; and Deng, L. 2016. Deep reinforcement learning with a combinatorial action space for predicting popular Reddit threads. *EMNLP*.

Hessel, M.; Modayil, J.; Van Hasselt, H.; Schaul, T.; Ostrovski, G.; Dabney, W.; Horgan, D.; Piot, B.; Azar, M.; and Silver, D. 2018. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Kergosien, Y.; Lenté, C.; and Billaut, J.-C. 2009. Home health care problem: An extended multiple traveling salesman problem. In *Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA 2009)*, 85–92.

Khalil, E.; Dai, H.; Zhang, Y.; Dilkina, B.; and Song, L. 2017. Learning combinatorial optimization algorithms over graphs. *Advances in Neural Information Processing Systems*, 30.

Levy, D. T.; Bauer, J. E.; and Lee, H.-r. 2006. Simulation modeling and tobacco control: creating more robust public health policies. *American Journal of Public Health*, 96(3): 494–498.

Liang, E.; Liaw, R.; Nishihara, R.; Moritz, P.; Fox, R.; Goldberg, K.; Gonzalez, J.; Jordan, M.; and Stoica, I. 2018. RLlib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*, 3053–3062. PMLR.

Lombardi, M.; and Milano, M. 2018. Boosting Combinatorial Problem Modeling with Machine Learning. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)*, 5472–5478.

Marot, A.; Donnot, B.; Dulac-Arnold, G.; Kelly, A.; O'Sullivan, A.; Viebahn, J.; Awad, M.; Guyon, I.; Panciatici, P.; and Romero, C. 2021. Learning to run a power network challenge: a retrospective analysis. In *NeurIPS 2020 Competition and Demonstration Track*, 112–132. PMLR.

Mate, A.; Madaan, L.; Taneja, A.; Madhiwalla, N.; Verma, S.; Singh, G.; Hegde, A.; Varakantham, P.; and Tambe, M. 2022. Field study in deploying restless multi-armed bandits: Assisting non-profits in improving maternal and child health. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 12017–12025.

Mazyavkina, N.; Sviridov, S.; Ivanov, S.; and Burnaev, E. 2021. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134: 105400.

Nakhleh, K.; Ganji, S.; Hsieh, P.-C.; Hou, I.; Shakkottai, S.; et al. 2021. NeurWIN: Neural Whittle index network for

restless bandits via deep RL. *Advances in Neural Information Processing Systems*, 34: 828–839.

Niño-Mora, J. 2023. Markovian Restless Bandits and Index Policies: A Review. *Mathematics*, 11(7): 1639.

Ou, H.-C.; Siebenbrunner, C.; Killian, J.; Brooks, M. B.; Kempe, D.; Vorobeychik, Y.; and Tambe, M. 2022. Networked restless multi-armed bandits for mobile interventions. In *Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Özbakir, L.; Baykasoğlu, A.; and Tapkan, P. 2010. Bees algorithm for generalized assignment problem. *Applied Mathematics and Computation*, 215(11): 3782–3795.

Papadimitriou, C. H.; and Tsitsiklis, J. N. 1994. The complexity of optimal queueing network control. In *Proceedings of IEEE 9th Annual Conference on Structure in Complexity Theory*, 318–322. IEEE.

Schaul, T.; Quan, J.; Antonoglou, I.; and Silver, D. 2016. Prioritized experience replay. In *ICLR*.

Schoenebeck, G.; and Tao, B. 2019. Beyond worst-case (in) approximability of nonsubmodular influence maximization. *ACM Transactions on Computation Theory (TOCT)*, 11(3): 1–56.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587): 484–489.

Silvestro, D.; Goria, S.; Sterner, T.; and Antonelli, A. 2022. Improving biodiversity protection through artificial intelligence. *Nature Sustainability*, 5(5): 415–424.

Song, H.; Jang, H.; Tran, H. H.; Yoon, S.-e.; Son, K.; Yun, D.; Chung, H.; and Yi, Y. 2019. Solving continual combinatorial selection via deep reinforcement learning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI)*.

Tjeng, V.; Xiao, K. Y.; and Tedrake, R. 2018. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *International Conference on Learning Representations*.

Tkachuk, V.; Bakhtiari, S. A.; Kirschner, J.; Jusup, M.; Bogunovic, I.; and Szepesvári, C. 2023. Efficient Planning in Combinatorial Action Spaces with Applications to Cooperative Multi-Agent Reinforcement Learning. In *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, 6342–6370. PMLR.

Treloar, N. J.; Fedorec, A. J.; Ingalls, B.; and Barnes, C. P. 2020. Deep reinforcement learning for the control of microbial co-cultures in bioreactors. *PLoS Computational Biology*, 16(4): e1007783.

Villar, S. S.; Bowden, J.; and Wason, J. 2015. Multi-armed bandit models for the optimal design of clinical trials: benefits and challenges. *Statistical Science: A Review Journal of the Institute of Mathematical Statistics*, 30(2): 199.

Wang, K.; Xu, L.; Taneja, A.; and Tambe, M. 2023. Optimistic Whittle Index Policy: Online Learning for Restless Bandits. In *Proc. Thirty-Seventh AAAI Conference on Artificial Intelligence (AAAI-23)*.

Weber, R. R.; and Weiss, G. 1990. On an index policy for restless bandits. *Journal of Applied Probability*, 27(3): 637–648.

Whittle, P. 1988. Restless bandits: Activity allocation in a changing world. *Journal of Applied Probability*, 25(A): 287–298.

8 Notation table

List of symbols used in the paper and their description.

Table 1: Symbols used in the paper.

Symbol	Description
	Problem description
$s_j \in \mathcal{S}$	state of arm $j \in [J]$
$a_j \in \mathcal{A}$	action taken on arm j
$P_j(s_j, a_j, s'_j)$	probability that arm j in state s_j transitions to state s'_j under action a_j
$r_j(s_j)$	reward at time t for arm j in state s_j
$R^{(t)}(\boldsymbol{s})$	reward for restless bandit instance with joint state $\boldsymbol{s}$
B	budget
H	time horizon
$\boldsymbol{s} \in \mathcal{S}^{\times}$	joint state of all arms
$\boldsymbol{a} \in \mathcal{A}^{\times}$	joint action of all arms
$\mathcal{C} \subseteq \mathcal{A}^{\times}$	feasible action space, as determined by set of constraints
$P^{\times}$	joint transition probability across all arms
$Q(\boldsymbol{s}, \boldsymbol{a})$	Q-value for action $\boldsymbol{a}$ under state $\boldsymbol{s}$
$V(\boldsymbol{s})$	value function for state $\boldsymbol{s}$
γ	discount factor
	Notation for SEQUOIA *algorithm*
ϵ	epsilon-greedy parameter
$\mathcal{D}$	dataset of historical trajectories stored in replay memory
Q	Q-network
$\hat{Q}$	target Q-network
θ	neural network parameters
θ^{-}	target neural network parameters
C	target network update frequency
	Problem-specific symbols
ϕ	link function
ω	weights of edges for "multiple intervention" probability transition function
$\mathcal{V}$	set of vertices in graph
$\mathcal{E}$	set of edges in graph
🏠	source node for network flow problem
$f_{j,k,t}$	network flow from node j to k at step t
T	max path length

9 Experimental details

To evaluate our sequential planning algorithm, we design a simulator that is designed to ensure the myopic policy is non-optimal. Thus across all experimental settings, we consider restless bandit problems with $|S| = 4$ states, where a fraction of the arms are long-run beneficial to act on (but myopically non-optimal), and the remainder of the arms are short-run beneficial to act on but myopically non-optimal.

Specifically, the bad arms have the following reward for each of the four states: $[0.1, 0.15, 0.2, r_{\text{bad}}]$ where r_{bad} is drawn randomly from $\{4, 5, 6\}$, and the good arms have reward $[0.2, 0.15, 0.1, r_{\text{good}}]$ where r_{good} is drawn randomly from $\{1, 2, 3\}$.

For the bad arms, the transition probability of moving up a state is drawn uniformly at random between $[0.0, 0.2]$, and the probability of moving down a state is drawn from $[0.7, 0.9]$.

10 Implementation

Our implementation uses PyTorch and TorchRL (Bou et al. 2023) libraries. We build off code from Dumouchelle et al. (2022) to embed a neural network with single-layer ReLU activations as a MIP (Fischetti and Jo 2018).

For consistency in results, the random seeds are set between 1 and 30 for the results.

We use the following hyperparameters:

Table 2: Hyperparameters used in implementation to train SEQUOIA

Hyperparameter	Setting
Discount factor γ	0.99
Epsilon greedy ϵ	0.9 starting; final 0.05; 1000 rate of exponential decay
Learning rate	2.5×10^{-4}
Adam ϵ	1.5×10^{-4}
Target update rate τ	1×10^{-6}
Minibatch size	32
Replay memory size	10,000
Number of training episodes	100
Network architecture	Two hidden layers of size 32 each

11 Runtime

Experiments are run on a cluster running CentOS with Intel(R) Xeon(R) CPU E5-2683 v4 @ 2.1 GHz with 8GB of RAM using Python 3.9.12. The MIP was solved using Gurobi optimizer 10.0.2.

Empirically, we noticed that a major computational bottleneck was the size of the Q-network. Even with only two hidden layers, training a network with 128 hidden units took an order of magnitude longer than with 32 hidden units, underlining the tradeoff between expressivity of the Q-network and runtime.

The average running time of the training procedure of SEQUOIA:

Table 3: Total runtime (in minutes)

$J = 20$	$J = 40$	$J = 100$
25.5	76.7	475.2