Q-Learning Lagrange Policies for Multi-Action Restless Bandits

Jackson A. Killian
jkillian@g.harvard.edu
Harvard University
Cambridge, MA, USA

Arpita Biswas
arpitabiswas@seas.harvard.edu
Harvard University
Cambridge, MA, USA

Sanket Shah
sanketshah@g.harvard.edu
Harvard University
Cambridge, MA, USA

Milind Tambe
milind_tambe@harvard.edu
Harvard University
Cambridge, MA, USA

ABSTRACT

Multi-action restless multi-armed bandits (RMABs) are a powerful framework for constrained resource allocation in which N independent processes are managed. However, previous work only study the offline setting where problem dynamics are known. We address this restrictive assumption, designing the first algorithms for learning good policies for Multi-action RMABs online using combinations of Lagrangian relaxation and Q-learning. Our first approach, MAIQL, extends a method for Q-learning the Whittle index in binary-action RMABs to the multi-action setting. We derive a generalized update rule and convergence proof and establish that, under standard assumptions, MAIQL converges to the asymptotically optimal multi-action RMAB policy as $t \to \infty$. However, MAIQL relies on learning Q-functions and indexes on two timescales which leads to slow convergence and requires problem structure to perform well. Thus, we design a second algorithm, LPQL, which learns the well-performing and more general Lagrange policy for multi-action RMABs by learning to minimize the Lagrange bound through a variant of Q-learning. To ensure fast convergence, we take an approximation strategy that enables learning on a single timescale, then give a guarantee relating the approximation's precision to an upper bound of LPQL's return as $t \to \infty$. Finally, we show that our approaches always outperform baselines across multiple settings, including one derived from real-world medication adherence data.

CCS CONCEPTS

• **Computing methodologies → Reinforcement learning**.

KEYWORDS

Multi-action Restless Bandits; Q-learning; Lagrangian Relaxation

ACM Reference Format:
Jackson A. Killian, Arpita Biswas, Sanket Shah, and Milind Tambe. 2021. Q-Learning Lagrange Policies for Multi-Action Restless Bandits. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3447548.3467370

1 INTRODUCTION

Restless Multi-Armed Bandits (RMABs) are a versatile sequential decision making framework in which, given a budget constraint, a planner decides how to allocate resources among a set of independent processes that evolve over time. This model, diagrammed in Fig. 1, has wide-ranging applications, such as in healthcare [4, 21, 23], anti-poaching patrol planning [26], sensor monitoring tasks [12, 17], machine replacement [27], and many more. However, a key limitation of these approaches is they only allow planners a binary choice—whether or not to allocate a resource to an arm at each timestep. However, in many real world applications, a planner may choose among multiple actions, each with varying cost and providing varying benefits. For example, in a community health setting (e.g., Figure 1), a health worker who monitors patients' adherence to medication may have the ability to provide interventions via text, call, or in-person visit. Such *multi-action* interventions require varying amount of effort (or cost), and cause varying effects on patients' adherence. Given a fixed budget, the problem for a health worker is to decide what interventions to provide to each patient and when, with the goal of maximizing the overall positive effect (e.g., the improvement of patients' adherence to medication).

Owing to the improved generality of *multi-action RMABs* over binary-action RMABs, this setting has gained attention in recent years [11, 16, 18]. However, critically, all these papers have assumed the *offline* setting, in which the dynamics of all the underlying processes are assumed to be known before planning. This assumption is restrictive since, in most cases, the planner will not have perfect information of the underlying processes, for example, how well a patient would respond to a given type of intervention.

To address this shortcoming in previous work, this paper presents the first algorithms for the *online* setting for multi-action RMABs. Indeed, the online setting for even binary-action RMABs has received only limited attention, in the works of Fu et al. [8], Avrachenkov and Borkar [3], and Biswas et al. [5, 6]. These papers adopt variants of the Q-learning update rule [29, 30], a well studied reinforcement learning algorithm, for estimating the effect of each action across changing dynamics of the systems. These methods aim to learn Whittle indexes [32] over time and use them for choosing actions. In the offline version, it has been shown that these indexes lead to an optimal selection policy when the RMAB instances meet the *indexability* condition. However, these methods only apply to binary-action RMABs. Our paper presents two new algorithms for online multi-action RMABs to address the shortcomings of previous work and presents an empirical comparison of the approaches. The paper provides three key contributions:

(1) **We design Multi-action Index-based Q-learning (MAIQL)**. We consider a multi-action notion of indexability where the index for each action represents the "fair charge" for taking that action [11]. If the dynamics of the underlying systems were known beforehand, an optimal policy for multi-action indexable RMABs would choose actions based on these indexes when a linear structure on the action costs is assumed [16]. We establish that, when these dynamics are unknown and are required to be learned over time, MAIQL provably converges to these indexes for any multi-action RMAB instance following the assumptions on cost and indexability. However, these assumptions can be limiting, and in addition, the algorithm requires a two-timescale learning procedure that can be slow and unstable.
(2) **We propose a more general algorithm, Lagrange Policy Q-learning (LPQL)**. This method takes a holistic back-to-the-basics approach of analyzing the Lagrangian relaxation of the multi-action RMAB problem and learning to play the *Lagrange policy* using the estimated Q values which are updated over time. This policy converges more quickly than MAIQL and other benchmark algorithms, is applicable to problems with arbitrary cost structures, and does not require the indexability condition.
(3) **We demonstrate the effectiveness of MAIQL and LPQL as compared to various baselines on several experimental domains,** including two synthetically generated domains and derived from a real-world dataset on medication adherence of tuberculosis patients. Our algorithms converge to the state-of-the-art offline policy much faster than the baselines, taking a crucial step toward real-world deployment in online settings.[1]

2 RELATED WORK

The *restless multi-armed bandit* (RMAB) problem was introduced by Whittle [32] where he showed that a relaxed version of the offline RMAB problem can be solved optimally using a heuristic called the *Whittle index policy.* This policy is shown to be asymptotically optimal when the RMAB instances satisfy the *indexability* condition [31]. Moreover, Papadimitriou and Tsitsiklis [24] established that solving RMABs is PSPACE-hard, even for the special case when the transition rules are deterministic.

Since then, a vast literature have studied various subclasses of RMABs and provided algorithms for computing the Whittle index. Lee et al. [21] study the problem of selecting patients for screening with the goal of maximizing early-stage cancer detection under limited resources. Mate et al. [23] consider bandits with two states to model a health intervention problem, where the uncertainty collapses after an active action. They showed that the model is indexable and gave a mechanism for computing the Whittle index policy. Bhattacharya [4] models the problem of maximizing the coverage and spread of health information with limited resources as an RMAB and proposes a hierarchical policy. Similarly, several other papers [12, 22, 28] give Whittle indexability results for different subclasses of RMABs where there are only two possible actions.

For more than two actions, Glazebrook et al. [11, 16] extended Whittle indexability to multi-action RMABs where the instances are assumed to have special monotonic structure. Along similar lines, Killian et al. [18] proposed a method that leverages the convexity of an approximate Lagrangian version of the multi-action RMAB problem.

Also related to multi-action RMABs are weakly coupled Markov decision processes (WCMDP). The goal of a WCMDP is to maximize reward subject to a set of constraints over actions, managing a finite number of independent Markov decision processes (MDPs). Hawkins [15] studied a Lagrangian relaxation of WCMDPs and proposed an LP for minimizing the Lagrange bound. On the other hand, Adelman and Mersereau [2] provide an approximation algorithm that achieves a tighter bound than the Lagrange approach to WCMDPs, trading off scalability. A more scalable approximation method is provided by Gocgun and Ghate [13].

However, these papers focused only on the offline versions of the problem in which the dynamics (transition and observation models) are known apriori. In the online setting, there has been some recent work on binary-action RMABs. Gafni and Cohen [9] propose an algorithm that learns to play the arm with the highest *expected* reward. However, this is suboptimal for general RMABs since rewards are state- and action-dependent. Addressing this, Biswas et al. [5] give a Q-learning-based based algorithm that acts on the arms that have the largest difference between their active and passive Q values. Fu et al. [8] take a related approach that adjust the Q values by some λ, and use it to estimate the Whittle index. Similarly, Avrachenkov and Borkar [3] provide a two-timescale algorithm that learns the Q values as well as the index values over time. However, their convergence proof requires indexability and that all arms are homogeneous with the same underlying MDPs. We use the two-timescale methodology and define a multi-action indexability criterion to provide a general framework to learn multi-action RMABs with provable convergence guarantees. Our work is the first to address the multi-action RMAB setting online.

3 PRELIMINARIES AND NOTATIONS

A Multi-action RMAB instance consists of N arms and a budget B on the total cost. Each arm $i \in [N]$ follows an MDP [25]. We define an MDP $\{\mathcal{S}, \mathcal{A}, \mathcal{C}, r, T, \beta\}$ as a finite set of states $\mathcal{S}$, a finite set of M actions $\mathcal{A}$, a finite set of action costs $\mathcal{C} := \{c_j\}_{j\in\mathcal{A}}$, a reward function $r : \mathcal{S} \to \mathbb{R}$, a transition function $T(s, a, s')$ denoting the probability of transitioning from state s to state s' when action a is taken, and a discount factor $\beta \in [0, 1)$[2]. An MDP *policy* $\pi : \mathcal{S} \to \mathcal{A}$ maps states to actions. The long-term *discounted reward* of arm i starting from state s is defined as

$$J^i_{\beta,\pi^i}(s) = E\left[\sum_{t=0}^{\infty} \beta^t r^i(s^i_t) | \pi^i, s^i_0 = s\right] \tag{1}$$

where $s^i_{t+1} \sim T(s^i_t, \pi^i(s^i_t), \cdot)$. For ease of exposition, we assume the action sets and costs are the same for all arms, but our methods will apply to the general case where each arm has arbitrary (but

[1]Code available at: https://github.com/killian-34/MAIQL_and_LPQL

[2]β is only included under the discounted reward case, as opposed to the average reward case which we address later.

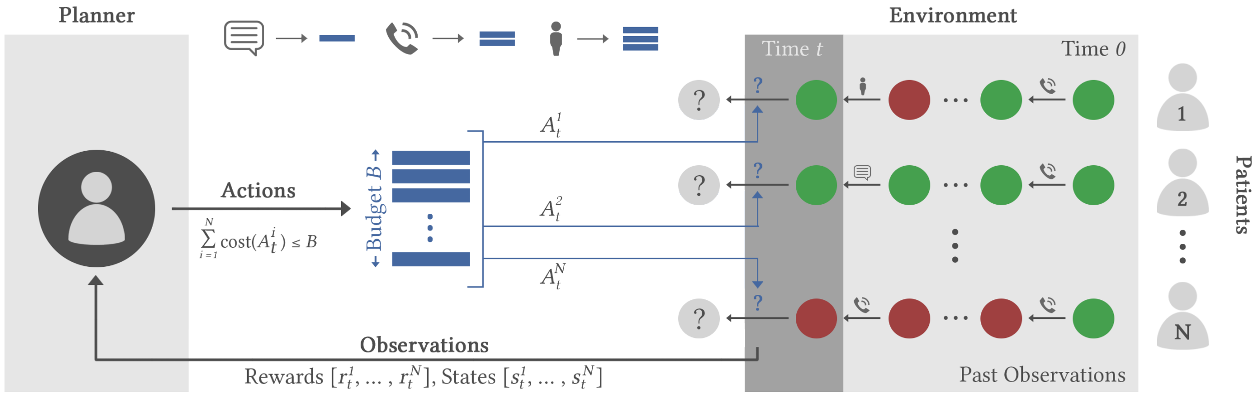

Figure 1: Schematic of a multi-action RMAB. At each timestep, t, the planner (e.g., health worker) takes one action on each of N processes (e.g., patients). The sum cost of actions each timestep must not exceed a budget, B. After taking actions at each timestep, the planner observes the rewards and state transitions of the processes, which the planner uses to improve their action selection in the future. The goal is to maximize reward.

finite) state, action, and cost sets. Without loss of generality, we also assume that the actions are numbered in increasing order of their costs, i.e., $0 = c_0 \le c_1 \le \dots, c_M$. Now, the planner must take decisions for all arms jointly, subject to two constraints each round: (1) select one action for each arm and (2) the sum of action costs over all arms must not exceed a given budget B. Formally, the planner must choose a decision matrix $\boldsymbol{A} \in \{0,1\}^{N \times M}$ such that:

$$\sum_{j=1}^{M} \boldsymbol{A}_{ij} = 1 \quad \forall i \in [N] \qquad\qquad \sum_{i=1}^{N}\sum_{j=1}^{M} \boldsymbol{A}_{ij} c_j \le B \tag{2}$$

Let $\overline{\mathcal{A}}$ be the set of decision matrices respecting the constraints in 2 and let $\boldsymbol{s} = (s^1, ..., s^N)$ represent the initial state of each arm. The planner's goal is to maximize the total discounted reward of all arms over time, subject to the constraints in 2, as given by the constrained Bellman equation:

$$J(\boldsymbol{s}) = \max_{\boldsymbol{A} \in \overline{\mathcal{A}}} \left\{ \sum_{i=1}^{N} r^i(s^i) + \beta E[J(\boldsymbol{s}')|\boldsymbol{s}, \boldsymbol{A}] \right\} \tag{3}$$

However, this corresponds to an optimization problem with exponentially many states and combinatorially many actions, making it PSPACE-Hard to solve directly [24]. To circumvent this, we take the Lagrangian relaxation of the second constraint in 2 [15]:

$$J(\boldsymbol{s}, \lambda) = \max_{\boldsymbol{A}} \left\{ \sum_{i=1}^{N} r^i(s^i) + \lambda\Big(B - \sum_{i=1}^{N}\sum_{j=1}^{M} \boldsymbol{A}_{ij} c_j\Big) + \beta E[J(\boldsymbol{s}', \lambda)|\boldsymbol{s}, \boldsymbol{A}] \right\} \tag{4}$$

Since this constraint was the only term coupling the MDPs, relaxing this constraint decomposes the problem except for the shared term λ. So Eq. 4 can be rewritten as (see [2]):

$$J(\boldsymbol{s}, \lambda) = \frac{\lambda B}{1-\beta} + \sum_{i=1}^{N} \max_{a_j^i \in \mathcal{A}^i} \{(Q^i(s^i, a_j^i, \lambda)\} \tag{5}$$

$$\text{where } Q^i(s^i, a_j^i, \lambda) = r^i(s^i) - \lambda c_j + \beta \sum_{s'} T(s^i, a_j^i, s') \max_{a_j \in \mathcal{A}^i} \{(Q^i(s', a_j, \lambda)\} \tag{6}$$

In Eq. 6, each arm is effectively decoupled, allowing us to solve for each arm independently for a given value of λ. The choice of λ, however, affects the resulting optimal policies in each of the arms. One intuitive interpretation of λ is that of a "penalty" associated with acting – given a fixed budget B, a planner must weigh the cost of acting λc_j against its ability to collect higher rewards. Thus, as λ is increased, the optimal policies on each arm will tend to prefer actions that generate the largest "value for cost".

The challenges we address are two-fold: (1) How to learn policies online that can be tuned by varying λ and (2) How to make choices for the setting of λ that lead to good policies. Our two algorithms in Sections 4 and 5 both build on Q-Learning to provide alternative ways of tackling these challenges – where MAIQL builds on the rich existing literature of "index" policies, LPQL goes "back to basics" and provides a more fundamental approach based on the Lagrangian relaxation discussed above.

4 ALGORITHM: MAIQL

Our first algorithm will reason about λ's influence on each arm's value function independently. Intuitively, this is desirable because it simplifies one size-N problem to N size-one problems that can be solved quickly. Our goal will be to compute *indexes* for each action on each arm that capture a given action's value, then greedily follow the indexes as our policy. Such an *index policy* was proposed by Whittle [32] for binary-action RMABs, in which the index is a value of λ such that the optimal policy is indifferent between acting and not acting in the given state. This policy has been shown to be asymptotically optimal under the indexability condition [31].

Glazebrook et al. [11] and Hodge and Glazebrook [16] extended the definition and guarantees, respectively, of the Whittle index to multi-action RMABs that satisfy the following assumptions:

(1) Actions have equally spaced costs, i.e., after normalization, the action costs can be expressed as $\{0, 1, \dots, M-1\}$.
(2) The utility of acting is submodular in the cost of the action.

For such multi-action RMABs, Glazebrook et al. [11] defines multi-action indexability and the multi-action index as follows:

Definition 4.1 (Multi-action indexability). An arm is multi-action indexable if, for every given action $a_j \in \mathcal{A}$, the set of states in which it is optimal to take an action of cost c_j or above decreases monotonically from $\mathcal{S} \to \emptyset$ as λ increases from $-\infty \to \infty$.

Definition 4.2 (Multi-action index, λ^*_{s,a_j}). For a given state s and action a_j, the multi-action index is the minimum λ^*_{s,a_j} that is required to become indifferent between the actions a_j and a_{j-1}:

$$\lambda^*_{s,a_j} = \inf_{\lambda}\{Q(s,a_j,\lambda) \le Q(s,a_{j-1},\lambda)\} \tag{7}$$

$$= \lambda,\ s.t.\ Q(s,a_j,\lambda) = Q(s,a_{j-1},\lambda) \tag{8}$$

where $Q(s,a_j,\lambda)$ is the Q-value of taking action a_j in state s with current and future rewards adjusted by λ.

Given these multi-action indices, Hodge and Glazebrook [16] suggest a way to greedily allocate units of resources that is asymptotically optimal – assume that the arms are in some state $\boldsymbol{s}$, then iterate from $1 \dots B$ and in each round allocate a unit of resource to the arm with the highest multi-action index associated with the next unit of resource. Specifically, if $\boldsymbol{\theta} = \langle \theta^1 \dots \theta^N \rangle$ units of resource have been allocated to each arm so far, then we allocate the next unit of resource to the arm with the highest $\lambda^*_{s^i,a_{\theta i}}$. Given that the action utilities (λ^*_{s,a_j}) are submodular in a_j by assumption, this *multi-action index policy* leads to the allocation in which the sum of multi-action index values across all the arms are maximised.

Given the policy's theoretical guarantees, an index-based solution to the multi-action RMAB problem is attractive. The question then is how to calculate the value of the multi-action indices λ^*_{s,a_j}. In the online setting (when the RMAB dynamics are unknown apriori), Avrachenkov and Borkar [3] proposes a method for estimating the Whittle indexes for binary-action RMABs and, in addition, proves that this algorithm's estimate converges to the Whittle index.

In this section, we describe the **M**ulti-**A**ction **I**ndex **Q**-**L**earning (MAIQL) algorithm. Our algorithm generalizes the update rule of the learning algorithm proposed by Avrachenkov and Borkar [3]. We consider the notion of multi-action indexability from Glazebrook et al. [11] to create an update rule that allows us to estimate the multi-action indexes (Section 4.1). In addition, we use the multi-action indexability property to show that the convergence guarantees from Avrachenkov and Borkar [3] are preserved in this multi-action extension (Section 4.2).

4.1 Algorithm

From Equation 8, we observe that if we could estimate the Q values for all the possible values of λ, we would know the value of λ^*_{s,a_j}. This is not possible in general, but we can convert this insight into an update rule for estimating λ^*_{s,a_j} in which we update the current estimate in the direction such that Eq. 8 is closer to being satisfied. Based on this, we propose an iterative scheme in which Q values and λ^*_{s,a_j} are learned together.

An important consideration is that, because the Q and λ^*_{s,a_j} values are inter-dependent, it is not straightforward to learn them together, since updating the estimate of one may adversely impact our estimate of the other. To combat this, we decouple the effects of learning each component by relegating them to separate time-scales. Concretely, this means that an adaptive learning rate $\alpha(t)$ for the Q values and $\gamma(t)$ for λ-values are chosen such that $\lim_{t\to\infty} \frac{\gamma(t)}{\alpha(t)} \to 0$, i.e., the Q values are learned on a fast-time scale in which λ values can be seen as quasi-static (details in the appendix). The resultant two time-scale approach is given below.

To calculate the multi-action index, for a given state $s \in \mathcal{S}$ and action $a_j \in \mathcal{A}$, we store two sets of values: (1) the Q values for all states and actions, $Q(s,a_j)\ \forall s \in S,\ a_j \in \mathcal{A}$, and (2) the current estimate of the multi-action index λ_{s,a_j}. All the Q and λ values are initiated to zero. Then, for a given state s in which we take action a_j, we observe the resultant reward r and next state s', then perform the following updates:

(1) **Q-update:** At a fast time-scale (adjusted by $\alpha(\nu(s,a_j,t))$), update to learn the correct Q values as in standard Q-learning:

$$Q^{t+1}_{\lambda}(s,a_j) = Q^t_{\lambda}(s,a_j) + \alpha(\nu(s,a_j,t))\big[[r(s) - \lambda^t_{s,a_j}c_j - f(Q^t_{\lambda}) + \max_{a'_j} Q^t_{\lambda}(s',a'_j)] - Q^t_{\lambda}(s,a_j)\big] \tag{9}$$

where $\nu(s,a_j,t)$ is a "local-clock" that stores how many times the specific $Q_\lambda(s,a_j)$ value has been updated in the past, and $f(Q^t_\lambda) = \frac{\sum_{s,a_j} Q^t_\lambda(s,a_j)}{\sum_{s,a_j} 1}$ is a function whose value converges to the optimal average reward [1]. We give the average reward case to align with the traditional derivation of binary-action Whittle indexes, but this update (and related theory) can be extended easily to the discounted reward case.

(2) **λ-update:** Then, at a slower time-scale (adjusted by a function $\gamma(t)$), we update the value of λ^t_{s,a_j} according to:

$$\lambda^{t+1}_{s,a_j} = \lambda^t_{s,a_j} + \gamma(t)\cdot(Q^t_\lambda(s,a_j) - Q^t_\lambda(s,a_{j-1})) \tag{10}$$

Note that the updates described in the paragraph above correspond to the estimation of a single multi-action index. To efficiently estimate $\lambda^*(s,a_j)\ \forall s,a$, we make use of the fact that our algorithm, like the Q-learning algorithm on which it is based, is off-policy – an off-policy algorithm does not require collecting samples using the policy that is being learned. As a result, rather than learn each of these multi-action index values sequentially, we learn them in parallel based on the samples drawn from a single policy.

Specifically, since learning each index value requires imposing the current estimate λ on all current and future action costs, and since a separate index is learned for all arms, states, and non-passive actions, $N(M-1)|\mathcal{S}|$ separate Q-functions (each a table of size $|\mathcal{S}| \times M$) and λ-values must be maintained, requiring $O(NM^2|\mathcal{S}|^2)$ memory. However, since the estimation of each index is independent, each round, the index and its Q-function can be updated in parallel, keeping the process efficient, but requiring $O(NM|\mathcal{S}|)$ time if computed in serial. To take actions, we follow an ϵ-greedy version of the multi-action index policy – which, when not acting randomly, greedily selects indices in increasing size order for each arm's current state, taking $O(NM)$ time – and store the resultant $\langle s,a,r,s' \rangle$ tuple in a replay buffer. The replay buffer is important because, in the multi-action setting, each (s,a) pair is not sampled equally often; specifically, especially when B is small, it is less likely to explore more expensive actions. After every fixed number of time-steps of running the policy, we randomly pick some $\langle s,a,r,s' \rangle$ tuples from the replay buffer with probability weighted inversely to the number of times the tuple has been used for training, and update the Q values associated with each of the multi-action indexes and the $\lambda_{s,a}$ estimate for the sampled (s, a).[3] The resulting algorithm is guaranteed to converge to the multi-action indexes. Pseudocode is given in the appendix.

4.2 Theoretical Guarantees

The attractiveness of the MAIQL approach comes from the fact that, if the problem is multi-action indexable, the indexes can always be found. Formally, we show:

Theorem 4.3. *MAIQL converges to the optimal multi-action index $\lambda^*_{s,a}$ for a given state s and action a under Assumptions 1, 2, 3, and the problem being multi-action indexable.*

Proof Sketch. **At the fast time-scale:** We can assume $\lambda_{s,a}$ to be static. Then, for a given value of $\lambda_{s,a} = \lambda'$, the problem reduces to a standard MDP problem, and the Q-learning algorithm converges to the optimal policy.

At the slow time-scale: We can consider the fast-time scale process to have converged, and we have the optimal Q values $Q^*_{\lambda'}$ corresponding to the current estimate of $\lambda_{s,a}$. Then, by the multi-action indexability property, we know that if $\lambda < \lambda^*$ an action of weight a or higher is preferred. As a result, $Q^*_{\lambda_{s,a}}(s, a) - Q^*_{\lambda_{s,a}}(s, a-1) > 0$, and so $\lambda^{t+1} > \lambda^t$. When $\lambda > \lambda^*$, the opposite is true and so $\lambda^{t+1} < \lambda^t$. As a result, we constantly improve our estimate of λ such that we eventually converge to the optimal multi-action index, i.e., $\lim_{t\to\infty} \lambda^t \to \lambda^*_{s,a}$. □

The detailed proof follows along the lines of Avrachenkov and Borkar [3], and can be found in the appendix. However, while they consider convergence in the binary-action case, our approach generalizes to the multi-action setting. The crux of the proof lies in showing how the multi-action index generalizes the properties of the Whittle index in the multi-action case, and leads to convergence in the slow time-scale.

4.3 MAIQL Limitations

The main limitations of MAIQL are (1) it assumes multi-action indexability and equally-spaced action costs to be optimal and (2) it learns on two time-scales, making convergence slow and unstable in practice. i.e., for the convergence guarantees to hold, MAIQL must see "approximately" infinitely many of all state-action pairs before updating λ once. This can be difficult to ensure in practice for arms with transition probabilities near 0 or 1, and for problems where the budget is small, since many more samples of (s,a) pairs with cheap actions will be collected than ones with expensive actions.

5 ALGORITHM: LPQL

In this section, we provide a more fundamental approach by studying the problem of minimizing $J(\cdot, \lambda)$ (Equation 5) over λ. By minimizing this value, we aim to compute a tight bound on Eq. 3, the value function of the original, non-relaxed problem, then follow the policies implied by the bound, i.e., the Lagrange policy. However, computing $J(\cdot, \cdot)$ requires the $Q^i(s, a, \lambda)$ values which in turn require the knowledge of transition probabilities (as shown in Equation 6). In absence of the knowledge of transition probabilities, we propose a method, called **L**agrange **P**olicy **Q**-**L**earning (LPQL). This method learns a representation of $Q^i(s, a_j, \lambda)$ by using samples obtained from the environment via a mechanism similar to MAIQL. However, rather than estimating $Q^i(s, a_j, \lambda)$ with the purpose of estimating some downstream value of λ (i.e., indexes), now the goal is to estimate the entire curve $Q^i(s, a_j, \lambda)$ with respect to λ. It is straightforward to show that $Q^i(s, a_j, \lambda)$ is convex decreasing in λ [15], meaning that once we have a representation of $Q^i(s, a_j, \lambda)$, minimizing $J(\cdot, \cdot)$ simply corresponds to a one-dimensional convex optimization problem that can be solved extremely quickly.

In addition to its speed, this approach is desirable because it is designed for RMAB instances without specific structures, i.e., LPQL accommodates arbitrary action costs and needs no assumption on indexability. It does so by computing the Lagrange policy, which is asymptotically optimal for binary-action RMABs regardless of indexability [31], and works extremely well in practice for multi-action settings [18]. LPQL enjoys these benefits, and further, is designed to work on a single learning timescale, making its convergence faster and more stable than MAIQL.

In the offline setting, $J(\cdot, \cdot)$ can be minimized by solving this linear program (LP), which can be derived directly from Eq. 5 [15]:

$$\begin{aligned}
&\min_{\lambda} J(\mathbf{s}, \lambda) = \min_{V^i(s^i,\lambda),\lambda} \frac{\lambda B}{1-\beta} + \sum_{i=0}^{N-1} \mu^i(s^i) V^i(s^i, \lambda) \\
&\text{s.t. } V^i(s^i, \lambda) \geq r^i(s^i) - \lambda c_j + \beta \sum_{s^{i\prime}} T(s^i, a^i_j, s^{i\prime}) V^i(s^{i\prime}, \lambda) \\
&\forall i \in \{0, ..., N-1\}, \quad \forall s^i \in \mathcal{S}, \quad \forall a_j \in \mathcal{A}, \text{ and } \lambda \geq 0
\end{aligned} \tag{11}$$

where $\mu^i(s^i) = 1$ if s^i is the start state for arm i and is 0 otherwise and $V^i(s^i, \lambda) = \max_{a_j}\{Q^i(s^i, a_j, \lambda)\}$. To learn $Q^i(s^i, a_j, \lambda)$ in the offline setting, we will build a piecewise- linear convex representation of the curve by estimating its value at various points λ_p. To do this, we keep a three-dimensional vector for each arm $Q(s, a_j, \lambda) \in \mathbb{R}^{|\mathcal{S}|\times M\times n_{lam}}$ where n_{lam} is the number of points λ_p at which to estimate the curve. For now, we choose the set of λ_p to be an equally spaced grid between 0 and some value $\lambda_{\max}$. Since $V^i(s, \lambda)$ is convex decreasing in λ, the largest possible value of λ that could be a minimizer of $J(\cdot, \cdot)$ is the λ where $\frac{dQ^i(s,a_j,\lambda)}{d\lambda} = 0$. Killian et al. [18] show that this value is no greater than $\frac{\max\{r\}}{\min\{C\}(1-\beta)}$, so this will serve as $\lambda_{\max}$ unless otherwise specified.

On each round, an (s, a_j, r, s') tuple is sampled for each arm. We store estimates of Q for each state, action, and λ_p value, requiring $O(Nn_{lam}|\mathcal{S}|M)$ memory. The update rule for $Q(s, a_j, \lambda_p)$ is:

$$\begin{aligned}
Q^{t+1}(s, a_j, \lambda_p) &= Q^t(s, a_j, \lambda_p) + \alpha(\nu(s, a_j, n)) * \\
&\Big[\big[r(s) - \lambda_p c_j + \beta \max_{a'_j \in \mathcal{A}} Q^t(s', a'_j, \lambda_p)\big] - Q^t(s, a_j, \lambda_p)\Big]
\end{aligned} \tag{12}$$

Where β is the discount factor. Each round, we sample a (s, a_j, r, s') tuple per arm, and for each arm loop to update $Q^{t+1}(s, a_j, \lambda_p)$ $\forall p$. As in MAIQL, this update can be parallelized but requires $O(Nn_{lam})$ time if computed serially. To choose a policy each round, we compute the minimum of Eq. 5 by finding the point at which increasing λ_p (stepping from 0, $\frac{\lambda_{\max}}{n_{lam}}, \ldots, \lambda_{\max}$) results in zero or

[3]all algorithms in this paper will be equipped with the replay buffer for fairness of comparison.

positive change in objective value, as computed via our estimates $Q(s, a_j, \lambda_p)$, taking $O(n_{lam})$ time. As our estimates $Q(s, a, \lambda_p)$ converge, we approximate points exactly on the true $Q(s, a_j, \lambda)$ curve. Even at convergence, there will be some small approximation error in the slope of the line that will manifest as error in the objective value, but in the next subsection, we show that the approximation error can be made arbitrarily small as n_{lam} increases.

Once the minimizing value of λ (λ_{min}) is found, we follow the knapsack from [18] to select actions, i.e., we input $Q(s, a_j, \lambda_{min})$ as values in a knapsack where the costs are the corresponding c_j and the budget is B. We then use the Gurobi optimizer software [14] to solve the knapsack, then carry out the policy in accordance with the selected actions, taking $O(NMB)$ time in total [18]. Pseudocode for LPQL is given in the appendix.

5.1 Theoretical Guarantees

We establish that, given a $\lambda_{\max}$, a higher n_{lam} results in a better approximation of the upper bound of the policy return, given in Eq. 5. We show that, given a state profile $\boldsymbol{s} = \{s^1, \ldots, s^N\}$, the asymptotic values of $V^i(s^i, \lambda)$ obtained at equally spaced discrete set of λ values (over-)approximates Equation 11. The smaller the intervals are, the closer is the approximated value of $J(\boldsymbol{s}, \lambda)$ at all λ points that are not at the interval points. Before stating the theorem formally, we define the *Chordal Slope* Lemma. For ease of representation, we drop the notations $\boldsymbol{s}$ and s^i from functions $V()$ and $J()$ and also remove the superscript i.

Lemma 5.0.1 (The Chordal Slope Lemma [10]). *Let F be a convex function on (a, b). If $x_1 < x < x_2$ are in (a, b), then for points $P_1 = (x_1, F(x_1))$, $P = (x, F(x))$, and $P2 = (x_2, F(x_2))$, the slope of the straight line P_1P is less than or equal to the slope of the straight line $P1P2$.*

Theorem 5.1. *Let $V'(\cdot)$ be a convex piecewise-linear function over equally spaced intervals ($\Lambda := \{0, x, 2x, 3x, \ldots\}$) that approximates the convex decreasing function $V(\lambda)$, such that*

$$V(\lambda) = V'(\lambda) \text{ for all } \lambda \in \Lambda.$$

If values $V(\cdot)$ are replaced by values $V'(\cdot)$, then $J(\cdot)$ (Equation 11) is better approximated when the interval length x is small.

Proof. $V(\cdot)$ are convex functions of λ which implies that the function $J(\lambda)$, the sum of convex functions, is also a convex function of λ. Let us assume that the convex decreasing function $V(\lambda)$ is approximated by a convex continuous piecewise-linear function $V'(\lambda)$, over equally spaced values, taken from the set $\Lambda := \{0, x, 2x, 3x, \ldots\}$, such that $V(\lambda) = V'(\lambda)$ for all $\lambda \in \Lambda$. Thus, using $V'(\cdot)$ values instead of $V(\cdot)$ values, we obtain an approximation $J'(\cdot)$ of the convex function $J(\lambda)$. The function $J'(\lambda)$ is a convex function with $J(\lambda) = J'(\lambda)$ for all $\lambda \in \Lambda$.

Now, let us assume two different values of x, say x_1 and x_2, where $x_1 < x_2$. The corresponding sets are $\Lambda_1 := \{0, x_1, 2x_1, \ldots\}$ and $\Lambda_2 := \{0, x_2, 2x_2, \ldots\}$. Considering Λ_1, and two points $\lambda_0 \geq 0$ and $\lambda_1 = \lambda_0 + x_1$, $J'(\cdot)$ is over approximated by a straight line $\bar{J}(\cdot)$ that connects $(\lambda_0, J(\lambda_0))$ and $(\lambda_1, J(\lambda_1))$. The equation for the line is given by:

$$\bar{J}_{\lambda_0,x_1,\lambda_1}(\lambda) = J(\lambda_0) + \frac{\lambda - \lambda_0}{x_1}(J(\lambda_1) - J(\lambda_0)) \; \forall \, \lambda_0 \leq \lambda \leq \lambda_1. \quad (13)$$

Similarly, considering Λ_2, the point λ_0, and point $\lambda_2 = \lambda_0 + x_2$ (where $x_1 < x_2$), $J'(\cdot)$ can be over approximated by a straight line $\bar{J}(\cdot)$ that connects $(\lambda_0, J(\lambda_0))$ and $(\lambda_2, J(\lambda_2))$. Thus, for any value of $\lambda \in [\lambda_0, \lambda_2]$, the difference $J'(\lambda) - J(\lambda)$ is given by:

$$\bar{J}_{\lambda_0,x_2,\lambda_2}(\lambda) = J(\lambda_0) + \frac{\lambda - \lambda_0}{x_2}(J(\lambda_2) - J(\lambda_0)) \; \forall \, \lambda_0 \leq \lambda \leq \lambda_2. \quad (14)$$

For a given $\lambda \in [\lambda_0, \lambda_1]$, the difference between the approximation obtained by Equation 14 and 13 is:

$$\begin{aligned} &(\lambda - \lambda_0)\left(\frac{J(\lambda_2) - J(\lambda_0)}{x_2} - \frac{J(\lambda_1) - J(\lambda_0)}{x_1}\right) \\ &\geq 0 \qquad\qquad (\because \lambda \geq \lambda_0 \text{ and } \textit{Lemma } 5.0.1) \quad (15) \end{aligned}$$

Thus, smaller the length of each interval, the corresponding surrogate $V'(\cdot)$ values can be used to obtain a better approximation of $J(\cdot)$ values. □

5.2 Extending LPQL Update Technique to Approximate MAIQL

The same tactic of approximating $Q(s, a_j, \lambda_p)$ can be used to create an **approximate version of MAIQL (MAIQL-Aprx)** that learns on a single timescale and is thus more sample efficient and stable. The algorithm follows much in the same way as LPQL, except that $Q(s, a, \lambda_p)$ are not used to minimize the LP. Instead, for each arm on each round, we compute the multi-action index for a given (s, a_j) by finding the $\text{argmin}_{\lambda_p} |Q(s, a_j, \lambda_p) - Q(s, a_{j-1}, \lambda_p)|$. We then choose actions according to the same greedy policy as MAIQL. We can show with the same logic as the LPQL approximation proof that with a large enough n_{lam}, the indexes can be approximated to an arbitrary precision. We investigate whether, due to its single timescale nature, this algorithm will have improved sample efficiency and convergence behavior compared to standard MAIQL.

6 EXPERIMENTAL RESULTS

In this section, we compare our algorithms against both learning baselines (**WIBQL** (Avrachenkov and Borkar [3]) and **QL-$\boldsymbol{\lambda}$=0**), and offline baselines (**Oracle LP**, **Oracle $\boldsymbol{\lambda}$=0**, and **Oracle-LP-Index**).

WIBQL is designed to learn Whittle indexes for *binary*-action RMABs, but we adapt it to the multi-action setting by allowing it to plan using two actions, namely the passive action a_0 and a non-passive action a_j ($j > 0$) for the entire simulation. Clearly, this will be suboptimal in general, so we also design a stronger, multi-action baseline, **QL-$\boldsymbol{\lambda}$=0**. This uses standard Q-learning to learn state-action values for each individual arm without reasoning about future costs or the shared budget between arms (i.e., $\lambda = 0$). At each step, the actions are chosen according to the knapsack procedure of LPQL. **Oracle $\boldsymbol{\lambda}$=0** is the offline version of QL-λ=0 (i.e., it knows the transition probabilities). **Oracle LP** is the offline version of LPQL that solves Eq.11 using an LP solver, then follows the same knapsack procedure as LPQL. **Oracle-LP-Index** is an offline version of MAIQL that computes the multi-action indexes using an LP (see appendix). Since the oracles are computationally expensive, they are run for 1000 timesteps to allow their returns to converge, then are extrapolated.

All algorithms follow an ϵ-greedy paradigm for exploration where ϵ decays each round according to $\epsilon_0/\lceil \frac{t}{D} \rceil$ where ϵ_0 and D are constants. All algorithms were implemented in Python 3.7.4

and LPs were solved using Gurobi version 9.0.3 via the gurobipy interface [14]. All results are presented as the average (solid line) and interquartile range (shaded region) over 20 independently seeded simulations and were executed on a cluster running CentOS with Intel(R) Xeon(R) CPU E5-2683 v4 @ 2.1 GHz with 4GB of RAM.

6.1 Two Process Types

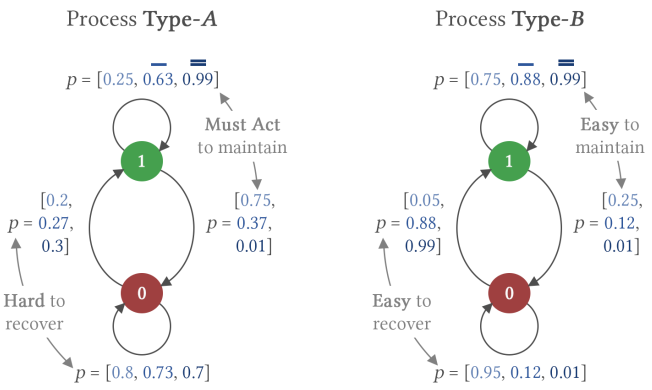

Figure 2: Two Process domain. Type-A arms need constant actions to stay in the good state (reward 1), whereas Type-B arms stay in the good state for many rounds after an action.

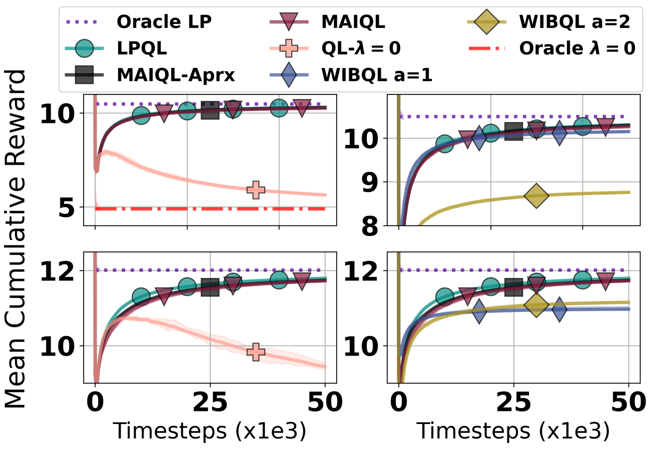

Figure 3: Results from Type-A v.s. Type-B domain with $N = 16$ and $B = 4$ (top row) and $B = 8$ (bottom row). Experiments in a row are the same, with different algorithms shown. Budget-agnostic learning converges to a highly suboptimal policy. Our algorithms converge to the best oracle policy, LPQL doing so the quickest. Binary-action planning underperforms except when a small budget forces the optimal policy to only use the cheapest action.

In the first experiment, we demonstrate how failing to account for cost and budget information while learning (i.e., QL-λ=0) can lead to poorly performing policies. The setting has two types of processes (arms), as in Fig. 2. Each has 3 actions, with costs 0, 1, and 2. Both arms have a good and bad state that gather 1 and 0 reward, respectively. The **Type-A** arm must be acted on every round while in the good state to stay there. However, in the bad state it is difficult to recover. This leads QL-λ=0 to learn that $Q(1, a_{j>0}, \lambda = 0) - Q(1, a_0, \lambda = 0)$ is large, i.e., acting in the good state is important for Type-A arms. Conversely, the **Type-B** arm will tend to stay in the good state even when not acted on, and when in the bad state, it can be easily recovered with any action. This leads QL-λ=0 to learn that $Q(1, a_{j>0}, \lambda = 0) - Q(1, a_0, \lambda = 0)$ is small. Thus QL-λ=0 will prefer to act on Type-A arms. However, if the number of Type-B arms is larger than the available budget, it is clearly better to spend the budget acting on Type-B arms since the action "goes farther", i.e., they may spend several rounds in the good state following only a single action, v.s. Type-A arms which are likely to only spend one round in the good state per action. Our budget-aware learning algorithms learn this tradeoff to converge to well-performing policies that greatly outperform cost-unaware planning.

We report the mean cumulative reward of each algorithm, i.e., its cumulative reward divided by the current timestep, averaged over all seeds. Fig. 3 shows the results with $N = 16$, 25% of arms as Type-A and 75% Type-B, over 50000 timesteps. The top and bottom rows use $B = 4$ and $B = 8$, respectively. For ease of visual representation, each column shows different combinations of algorithms – please note that the y-axis scales for each plot may be different. Fig. 4 shows results for the same arm type split and simulation length with $B = 8$, varying $N \in [16, 32, 48]$ (top to bottom). Parameter settings for each algorithm are included in the appendix. We see that each of our algorithms beat the baselines and converge in the limit to the Lagrange policy – equivalent to the multi-action index policy in this case – with the single-timescale algorithms converging quickest. Since the rewards obtained using Oracle-LP-Index coincide with Oracle LP, we do not plot the results for Oracle-LP-Index. Further, the plots demonstrate that the WIBQL algorithms underperform in general, except in cases where budgets are so small that the optimal policy effectively becomes binary-action (e.g., Fig 3 top right; $B = 4$). In the remaining experiments, WIBQL is similarly dominated and so is omitted for visual clarity. In both figures, interestingly, QL-λ=0 performs well at first while ϵ is large, suggesting that a random policy would outperform the $\lambda = 0$ policy. However, it eventually converges to Oracle-λ=0 as expected.

6.2 Random Matrices

The second experimental setting demonstrates LPQL's superior generality over index policies and its robustness to increases in the number of actions and variations in cost structure. In this setting, all transition probabilities, rewards, and costs are sampled uniformly at random, ensuring with high probability that the submodular action effect structure required for MAIQL's good performance will not exist. What remains to investigate is whether LPQL will be able to learn better policies than MAIQL in such a setting. Specifically, rewards for each state on each arm are sampled uniformly from $[0, 1]$, with $|\mathcal{S}| = 5$. Action costs are sampled uniformly from $[0, 1]^{|\mathcal{A}|}$, then we apply a cumulative sum to ensure that costs are increasing (but c_0 is set to 0). Fig. 5 shows results for $N = 16$ and $B = N|\mathcal{A}|/2$ as $|\mathcal{A}|$ varies in $[2, 5, 10]$ (top to bottom) over 50000 timesteps. Note that B scales with $|\mathcal{A}|$ to ensure that optimal policies will include the additional action types, since the costs of the

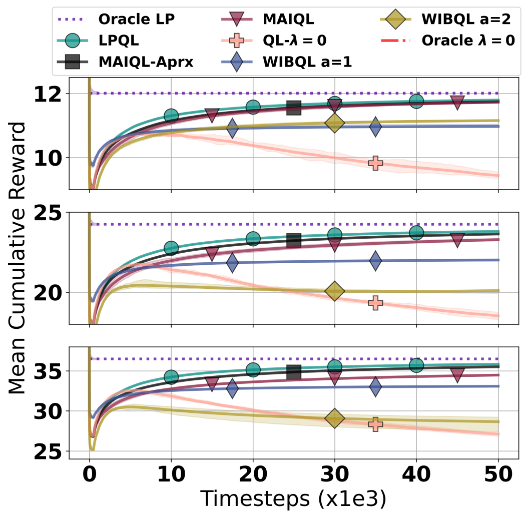

Figure 4: Results from the Two Process domain with $B = 8$ and $N \in [16, 32, 48]$ (top to bottom). Budget-agnostic converges to highly suboptimal policies, while our algorithms converge to the best oracle policy, with single-timescale versions doing so the quickest. Binary action planning underperforms with the $a = 2$ adaptation deteriorating as the budget becomes more constrained. Oracle $\lambda = 0$ (not shown) is dominated by all lines.

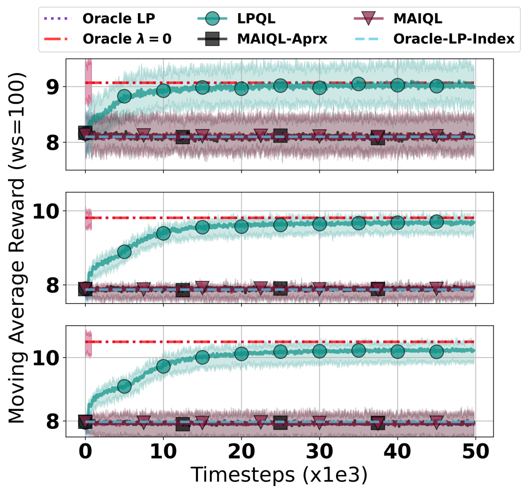

Figure 5: Moving average rewards from the random domain, for $|\mathcal{A}| \in [2, 5, 10]$ (top to bottom) using a window size (ws) of 100. Oracle LP and Oracle $\lambda = 0$ perform the same, as do MAIQL and MAIQL-Aprx. Oracle-LP-Index computes the index solution offline, demonstrating that MAIQL(-Aprx) are converging correctly, but the index policy performs poorly. LPQL converges quickly even as $|\mathcal{A}|$ increases.

additional action types also scale with $|\mathcal{A}|$. Rewards are shown as a moving average with a windows size of 100, which gives a clearer view of between-seed variance than the cumulative view. Fig. 5 shows that not only is LPQL able to learn much better policies than MAIQL and MAIQL-Aprx, which themselves converge to their oracle upper bound (Oracle-LP-Index), it does so with convergence behavior that is robust to increases in the number of actions, achieving near-optimal average returns at around 10k steps in each setting. Parameter settings for the different algorithms are again included in the appendix.

6.3 Medication Adherence

Finally, we run an experiment using data derived in-part from a real medication adherence domain [19]. The data contains daily 0-1 records of adherence from which transition probabilities can be estimated, assuming a corresponding 0-or-1 state (partial state history can also be accommodated). However, the data contains no records of actions and so must be simulated. In this experiment, we simulate actions that assume a natural "diminishing returns" structure in accordance with the assumption in section 4. One drawback is this estimation procedure creates uniform action effects across arms in expectation, i.e., a single "mode". However, in the real world we expect there to be multiple modes, representing patients' diverse counseling needs and response rates to various intervention types. To obtain multiple modes in a simple and interpretable way, we sample 25% of arms as Type-A arms from section 6.1, since they also have a binary state structure and are easily extended to accommodate partial state history. More details are given in the appendix. Note that, similar to Section 6.1, Oracle LP coincides with Oracle-LP-Index and hence, we do not plot the results for Oracle-LP-Index separately. Fig. 1 visualizes this domain.

Fig. 6 shows the results for the medication adherence domain with history lengths of 2, 3, and 4 (top to bottom), $N = 16$, $B = 4$, and 3 actions of cost 0, 1, and 2, over 100000 timesteps. Please see the appendix for parameter settings. This demonstrates concretely that learning on a single timescale (LPQL and MAIQL-Aprx) clearly improves speed of convergence, and this becomes more pronounced as the size of the state space increases. To understand why, we analyzed the estimated transition matrices and found that many patients had values near 0 or 1. This makes it very rare to encounter certain states, making it difficult to obtain sufficient numbers of samples across all state action pairs for MAIQL's assumptions to hold, impeding its learning.

7 CONCLUSION

To the best of our knowledge, we are the first to provide algorithms for learning Multi-action RMABs in an online setting. We show that by following the traditional approaches to RMAB problems, i.e., seeking index policies in domains with structural assumptions, MAIQL is guaranteed to converge to the optimal solution as $t \to \infty$. However, it is not efficient, due to its two-timescale structure, and is limited in scope, due to its indexability assumption. We solve these challenges by going back to the fundamentals of RMABs to develop LPQL which works well regardless of the problem structure, and outperforms all other baselines in terms of both convergence rate and obtained reward. Towards a real-world RMAB deployment, our models would apply to settings that allow many repeat interactions over a long horizon, e.g., life-long medication adherence regimens [7]. However, since our algorithms require thousands of samples to learn, more work is needed to apply to many settings which may have short horizons. Still, this work lays a methodological and theoretical foundation for future work in online multi-action RMABs, a crucial step toward their real-world deployment.

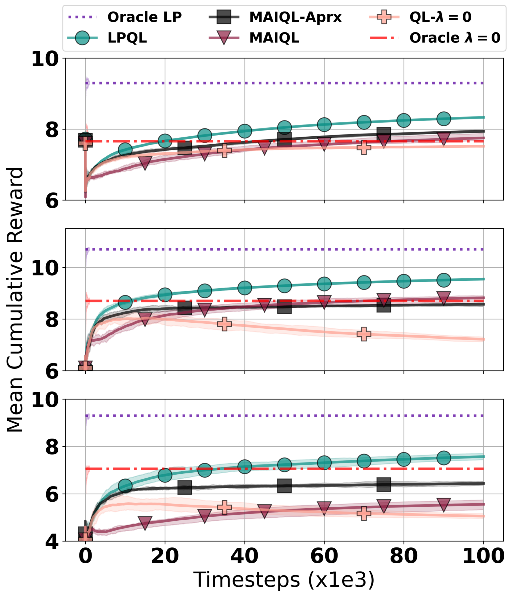

Figure 6: Mean cumulative reward on medication adherence domain with 16 patients, $B = 4$ and history length of 2, 3, and 4 (top to bottom). LPQL is the fastest to converge and converges to the best policies across all history lengths. MAIQL is slower to learn but does so eventually, where its approximate variant that learns on a single-timescale is more stable as the state size increases.

ACKNOWLEDGMENTS

This work was supported in part by the Army Research Office by Multidisciplinary University Research Initiative (MURI) grant number W911NF1810208. J.A.K. was supported by an NSF Graduate Research Fellowship under grant DGE1745303. A.B. was supported by the Harvard Center for Research on Computation and Society.

REFERENCES

[1] Jinane Abounadi, Dimitrib Bertsekas, and Vivek S Borkar. 2001. Learning algorithms for Markov decision processes with average cost. *SIAM J. Control Optim.* 40, 3 (2001), 681–698.
[2] Daniel Adelman and Adam J Mersereau. 2008. Relaxations of weakly coupled stochastic dynamic programs. *Oper. Res.* 56, 3 (2008), 712–727.
[3] Konstantin Avrachenkov and Vivek S Borkar. 2020. Whittle index based q-learning for restless bandits with average reward. *arXiv preprint arXiv:2004.14427* (2020).
[4] Biswarup Bhattacharya. 2018. Restless bandits visiting villages: A preliminary study on distributing public health services. In *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies.* 1–8.
[5] Arpita Biswas, Gaurav Aggarwal, Pradeep Varakantham, and Milind Tambe. 2021. Learn to Intervene: An Adaptive Learning Policy for Restless Bandits in Application to Preventive Healthcare. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence.*
[6] Arpita Biswas, Gaurav Aggarwal, Pradeep Varakantham, and Milind Tambe. 2021. Learning Index Policies for Restless Bandits with Application to Maternal Healthcare. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems.* 1467–1468.
[7] José Côté, Marie-Chantal Fortin, Patricia Auger, Geneviève Rouleau, Sylvie Dubois, Nathalie Boudreau, Isabelle Vaillant, and Élisabeth Gélinas-Lemay. 2018. Web-based tailored intervention to support optimal medication adherence among kidney transplant recipients: pilot parallel-group randomized controlled trial. *JMIR Form. Res.* 2, 2 (2018), e14.
[8] Jing Fu, Yoni Nazarathy, Sarat Moka, and Peter G Taylor. 2019. Towards Q-learning the Whittle Index for Restless Bandits. In *2019 Australian & New Zealand Control Conference (ANZCC).* IEEE, 249–254.
[9] Tomer Gafni and Kobi Cohen. 2020. Learning in restless multi-armed bandits via adaptive arm sequencing rules. *IEEE Trans. Automat. Control* (2020).
[10] Robert Gardner. 2017. Convex Functions. https://faculty.etsu.edu/gardnerr/5210/Beamer-Proofs/Proofs-6-6-print.pdf. Accessed: 2021-01-15.
[11] Kevin D Glazebrook, David J Hodge, and Chris Kirkbride. 2011. General notions of indexability for queueing control and asset management. *Ann. Appl. Probab.* (2011), 876–907.
[12] Kevin D Glazebrook, D. Ruiz-Hernandez, and Chris Kirkbride. 2006. Some indexable families of restless bandit problems. *Adv. Appl. Probab.* 38, 3 (2006), 643–672.
[13] Yasin Gocgun and Archis Ghate. 2012. Lagrangian relaxation and constraint generation for allocation and advanced scheduling. *Computers & Operations Research* 39, 10 (2012), 2323–2336.
[14] LLC Gurobi Optimization. 2021. Gurobi Optimizer Reference Manual. http://www.gurobi.com
[15] Jeffrey Thomas Hawkins. 2003. *A Langrangian decomposition approach to weakly coupled dynamic optimization problems and its applications.* Ph.D. Dissertation. Massachusetts Institute of Technology.
[16] David J Hodge and Kevin D Glazebrook. 2015. On the asymptotic optimality of greedy index heuristics for multi-action restless bandits. *Adv. Appl. Probab* 47, 3 (2015), 652–667.
[17] Fabio Iannello, Osvaldo Simeone, and Umberto Spagnolini. 2012. Optimality of myopic scheduling and whittle indexability for energy harvesting sensors. In *2012 46th Annual Conference on Information Sciences and Systems (CISS).* IEEE, 1–6.
[18] Jackson A Killian, Andrew Perrault, and Milind Tambe. 2021. Beyond "To Act or Not to Act": Fast Lagrangian Approaches to General Multi-Action Restless Bandits. In *Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems.*
[19] Jackson A Killian, Bryan Wilder, Amit Sharma, Vinod Choudhary, Bistra Dilkina, and Milind Tambe. 2019. Learning to prescribe interventions for tuberculosis patients using digital adherence data. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.* 2430–2438.
[20] Chandrashekar Lakshminarayanan and Shalabh Bhatnagar. 2017. A stability criterion for two timescale stochastic approximation schemes. *Automatica* 79 (2017), 108–114.
[21] Elliot Lee, Mariel S Lavieri, and Michael Volk. 2019. Optimal screening for hepatocellular carcinoma: A restless bandit model. *Manuf. Serv. Oper. Manag.* 21, 1 (2019), 198–212.
[22] Keqin Liu and Qing Zhao. 2010. Indexability of restless bandit problems and optimality of whittle index for dynamic multichannel access. *IEEE Trans. Inf. Theory* 56, 11 (2010), 5547–5567.
[23] Aditya Mate, Jackson A Killian, Haifend Xu, Andrew Perrault, and Milind Tambe. 2020. Collapsing Bandits and Their Application to Public Health Interventions. In *Neural Information Processing Systems, NeurIPS.*
[24] Christos H Papadimitriou and John N Tsitsiklis. 1994. The complexity of optimal queueing network control. In *Proceedings of IEEE 9th Annual Conference on Structure in Complexity Theory.* IEEE, 318–322.
[25] Martin L Puterman. 2014. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons.
[26] Yundi Qian, Chao Zhang, Bhaskar Krishnamachari, and Milind Tambe. 2016. Restless poachers: Handling exploration-exploitation tradeoffs in security domains. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems.* 123–131.
[27] Diego Ruiz-Hernández, Jesús M Pinar-Pérez, and David Delgado-Gómez. 2020. Multi-machine preventive maintenance scheduling with imperfect interventions: A restless bandit approach. *Comput. Oper. Res.* 119 (2020), 104927.
[28] Bejjipuram Sombabu, Aditya Mate, D Manjunath, and Sharayu Moharir. 2020. Whittle index for AoI-aware scheduling. In *IEEE International Conference on Communication Systems & Networks (COSMSNETS).* IEEE.
[29] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3-4 (1992), 279–292.
[30] Christopher John Cornish Hellaby Watkins. 1989. Learning from delayed rewards. (1989).
[31] Richard R Weber and Gideon Weiss. 1990. On an index policy for restless bandits. *J. Appl. Probab.* 27, 3 (1990), 637–648.
[32] Peter Whittle. 1988. Restless bandits: Activity allocation in a changing world. *J. Appl. Probab.* 25, A (1988), 287–298.

A PROOF OF CONVERGENCE FOR MAIQL

In this section, we provide a detailed proof of the convergence for MAIQL. We begin by stating 2 standard assumptions for establishing the convergence guarantee of Q-learning in the average-reward setting, and then add a third that's required for two time-scale convergence.

Assumption 1 (Uni-chain Property). *There exists a state s_0 that is reachable from any other state $s \in S$ with a positive probability under any policy.*

This property formalises the notion that there aren't any 'forks' in the MDP, in each of which very different outcomes could occur. This is important because, if there were a fork, the notion of 'average' reward would be ill-defined as it would depend on which 'fork' gets taken.

Assumption 2 (Asynchronous Update Step-Size). *The sequence of step-sizes $\{\alpha(t)\}$ satisfy the following properties for any $x \in (0,1)$:*

$$\sup_t \frac{\alpha(\lfloor xt \rfloor)}{\alpha(t)} < \infty$$
$$\sup_{y\in[x,1]} \left| \frac{\sum_{m=0}^{\lfloor yt \rfloor} \alpha(m)}{\sum_{m=0}^{t} \alpha(m)} - 1 \right| \to 0$$

This is a condition that is required to show that updating $Q(s, a_j)$ values one at a time with an ϵ-greedy policy is equivalent to updating all the $Q(s, a_j)$ values together, in expectation.

Assumption 3 (Relative Step-Size). *The two sequences of step-sizes, $\{\alpha(t)\}$ and $\{\gamma(t)\}$, satisfy the following properties:*

$$\text{(A) } \textit{Fast Time-Scale:} \qquad \sum_{t=0}^{\infty} \alpha(t) \to \infty, \qquad \sum_{t=0}^{\infty} \alpha^2(t) < \infty$$
$$\text{(B) } \textit{Slow Time-Scale:} \qquad \sum_{t=0}^{\infty} \gamma(t) \to \infty, \qquad \sum_{t=0}^{\infty} \gamma^2(t) < \infty$$
$$\text{(C)} \quad \lim_{t\to\infty} \frac{\gamma(t)}{\alpha(t)} \to 0$$

An example of possible step sizes for which this condition is true is $\alpha(t) = \frac{1}{t}$ and $\gamma(t) = \frac{1}{t \log t}$. In our experiments we use $\alpha(t) = \frac{C}{\lceil \frac{t}{D} \rceil}$, and $\gamma(t) = \frac{C'}{1+\lceil \frac{t \log(t)}{D} \rceil}$.

We then detail the proof for Theorem 4.3 below. This proof involves mapping the discrete Q and λ updates from the MAIQL algorithm (Section 4) to updates in an equivalent continuous-time Ordinary Differential Equation (ODE). This conversion then allows us to use the analysis tools created to analyse the evolution of two-timescale ODEs to show that our coupled updates converge. The proof detailed below broadly follows along the lines of Avrachenkov and Borkar [3], but where they discuss convergence in the binary action case, we generalize their proof to the multi-action scenario by using the notion of multi-action indexability from [12].

Theorem 4.3. *MAIQL converges to the optimal multi-action index $\lambda^*_{s,a}$ for a given state s and action a under Assumptions 1, 2, 3, and the problem being multi-action indexable.*

Proof. To convert these discrete updates to ODEs, we map a given time-step t to a point $\tau = T(t)$ in a continuous time, such that any time $T(t) = \sum_{m=0}^{t} \alpha(t)$. Because we're parameterising the time with α (rather than γ) we call τ the fast time-scale. To make this more concrete, we define $Q(\tau)$ as a function of the Q-value with time, and set $Q(T(t)) = Q^t$ to the value of the Q-function after t updates . Then, for values of $T(t) < \tau < T(t+1)$, $Q(\tau)$ is assumed to be linearly interpolated between Q^t and Q^{t+1}, creating a continuous function of τ. Similarly, we define $\lambda(\tau)$ such that $\lambda(T(t)) = \lambda^t$

We can then re-arrange the terms in Equation 9 to create an ODE that characterises the value of $Q(\tau)$:

$$Q^{t+1}(s, a_j) = Q^t(s, a) + \alpha(t)\big[[r(s) - \lambda^t_{s,a_j} c_j - f(Q^t) + \max_{a'_j\in\{0,1\}} Q^t(s', a'_j)] - Q^t(s, a_j)\big]$$

$$\Rightarrow \underbrace{\frac{Q^{t+1}(s, a_j) - Q^t(s, a_j)}{\alpha(t)}}_{\dot{Q}(\tau)} = [r(s) - \lambda^t_{s,a_j} c_j - f(Q^t) + \max_{a'_j\in\{0,1\}} Q^t(s', a'_j)] - Q^t(s, a_j)$$

where $\dot{Q}(\tau)$ is the derivative of $Q(\tau)$ and corresponds to the slope of the interpolated function in the range $(T(t), T(t+1))$.
Similarly, we can re-arrange Equation 10 to get the ODE for $\lambda(\tau)$:

$$\lambda^{t+1}_{s,a_j} = \lambda^t_{s,a_j} + \alpha(t)\left(\frac{\gamma(t)}{\alpha(t)}\right)(Q^t(s, a_j) - Q^t(s, a_{j-1}))$$

$$\Rightarrow \underbrace{\frac{\lambda^{t+1}_{s,a_j} - \lambda^t_{s,a_j}}{\alpha(t)}}_{\dot{\lambda}(\tau)} = \left(\frac{\gamma(t)}{\alpha(t)}\right)(Q^t(s, a_j) - Q^t(s, a_{j-1})) \tag{16}$$

Then, if look at Equation 16, we see $\lim_{\tau\to\infty} \dot{\lambda}(\tau) \to 0$ because, by Assumption 3 (c), $\lim_{t\to\infty} \frac{\gamma(t)}{\alpha(t)} \to 0$ and, by Assumption 3 (A), $T(\infty) = \sum_{t=0}^{\infty} \alpha(t) \to \infty$. Therefore, $\lambda(\tau)$ can be seen as quasi-static w.r.t. $Q(\tau)$ at the fast time-scale. As a result, the updates in this time-scale correspond to standard Q-Learning for a fixed MDP defined by the value of $\lambda(\tau)$. Given Assumptions 1, 3 (A), and 2, this is known to converge to the optimal Q-values Q^*_λ for the given value of $\lambda(\tau)$ [1].

Now, at the slow time-scale τ', we can repeat this continuous-time re-parameterisation, except with $T'(t) = \sum_{m=0}^{t} \gamma(t)$. Then, re-arranging Equation 9 in a similar way as above, we get:

$$\underbrace{\frac{Q^{t+1}(s, a_j) - Q^t(s, a_j)}{\gamma(t)}}_{\dot{Q}(\tau')} = \left(\frac{\alpha(t)}{\gamma(t)}\right)[r(s) - \lambda^t_{s,a_j} c_j - f(Q^t) + \max_{a'_j\in\{0,1\}} Q^t(s', a'_j)] - Q^t(s, a_j)$$

Now, given that $\lim_{t\to\infty} \frac{\alpha(t)}{\gamma(t)} \to \infty$, and from the argument above about the Q-values converging in the fast time-scale, we can see the interpolated $\lambda(\tau')$ value as tracking the converged Q-values $Q^*_{\lambda(\tau')}$ (for that value of $\lambda(\tau')$). Then, we can write the ODE for $\lambda(\tau')$ as:

$$\dot{\lambda}(\tau') = Q^*_{\lambda(\tau')}(s, a_j) - Q^*_{\lambda(\tau')}(s, a_{j-1})$$

where $Q^*_{\lambda(\tau')}$ corresponds to the optimal Q-values corresponding to the given value of $\lambda(\tau')$.

Now, if $\lambda(\tau') < \lambda^*_{s,a_j}$ (the multi-action index for state s and action a_j), by the definition of the multi-action index from the main text, we know that an action of weight c_j or higher is preferred. As a result, we see that $\dot{\lambda}(\tau') > 0$ in that case. If $\lambda(\tau') > \lambda^*_{s,a_j}$, the opposite is true and so $\dot{\lambda}(\tau') < 0$. Then, because $\lambda(0) = 0$ is bounded and given the step-sizes in Assumption 3 (B), $\lambda(\tau')$ converges to an equilibrium in which $Q^*_\lambda(s, a_j) - Q^*_\lambda(s, a) \rightarrow 0$.

Given that, by definition, $\lambda^*(s, a_j)$ is the value at which $Q^*_\lambda(s, a_j) = Q^*_\lambda(s, a_{j-1})$, $\lambda(\tau')$ converges to the multi-action index. □

This is a high-level proof, but the specific conditions for convergence can be seen in Lakshminarayanan and Bhatnagar [20]. They require 5 conditions: (1) Lipschitzness, (2) Bounded 'noise', (3) Properties about the relative step-sizes, (4) Convergence of fast time-scale, and (5) Convergence of slow time-scale.

Of these, (1)-(4) proceed in much the same way as in Avrachenkov and Borkar [3] because they do not depend on the multi-action extension of indexability. In addition, it is easy to show that the proof of (5) from Avrachenkov and Borkar [3] extends to the multi-action case which considers the limiting value of $Q(s, a_j) - Q(s, a_{j-1})$ rather than $Q(s, 1) - Q(s, 0)$. As a result, we refer the reader to Avrachenkov and Borkar [3] for the complete proof.

B REPRODUCIBILITY

Code is available at https://github.com/killian-34/MAIQL_and_LPQL. All the Q and λ values are initiated to zero in all the experiments. The parameter settings used for the two process type, random, and medication adherence data experiments are included in Tables 1, 2, and 3, respectively. C is the multiplier for the size of the Q-value updates. C' is the multiplier for the size of the index value updates. "Rp/dream" is the number of replays per dream. "Rp T" is the replay period (replay every T steps). λ-bound is the upper bound (and negative of the lower bound) imposed on values of the indices for WIBQL and MAIQL during learning – placing these bounds sometimes helps prevent divergent behavior in early rounds when updates are large – $\lambda_{\max}$ is the upper bound value that an index could take, as defined by the problem parameters, i.e., $\frac{\max\{r\}}{\min\{C\}(1-\beta)}$ [18]. D is the divisor of the decaying ϵ-greedy function as well as the divisor of $\alpha(t)$ and $\gamma(t)$, the decaying functions defining the size of the updates of Q-values and index values, defined in the previous section. ϵ_0 is the multiplier for the ϵ-greedy function. n_{lam} is the number of points in λ-space used to approximate the Q(s, a, λ)-functions in LQPL and MAIQL-Aprx. All values were determined via manual tuning – empirically we found that most parameter settings led to similar long-term performance between algorithms, as long as the settings did not cause the algorithms to diverge. In the tables, M-Aprx stands for MAIQL-Aprx.

C ALGORITHM PSEUDOCODES

See Algorithms 1 and 2 for the update and action selection steps of MAIQL and Algorithms 3 and 4 for the update and action selection steps of LPQL. The linear program for Oracle-LP-Index for a given current state s_{cur} and action a_k is given below:

	WIBQL	QL-λ=0	MAIQL	M-Aprx	LPQL
C	0.1	0.2	0.1	0.4	0.4
C'	0.2	-	0.2	-	-
Rp/dream	NA	1000	1000	1000	NA
Rp T	1E+06	100	10	100	1E+06
λ-bound	3	-	3	3	3
D	500	500	500	500	500
ϵ_0	0.99	0.99	0.99	0.99	0.99
n_{lam}	-	-	-	3000	3000

Table 1: Parameter settings for two process experiment.

	WIBQL	QL-λ=0	MAIQL	M-Aprx	LPQL
C	-	-	0.2	0.8	0.8
C'	-	-	0.4	-	-
Rp/dream	-	-	1000	NA	NA
Rp T	-	-	100	1E+06	1E+06
λ-bound	-	-	$\lambda_{\max}$	$\lambda_{\max}$	$\lambda_{\max}$
D	-	-	500	500	500
ϵ_0	-	-	0.99	0.99	0.99
n_{lam}	-	-	-	2000	2000

Table 2: Parameter settings for random data experiment.

	WIBQL	QL-λ=0	MAIQL	M-Aprx	LPQL
C	-	0.8	0.05	0.8	0.8
C'	-	-	0.1	-	-
Rp/dream	-	1000	1000	1000	1000
Rp T	-	10	5	5	5
λ-bound	-	-	$\lambda_{\max}$	$\lambda_{\max}$	$\lambda_{\max}$
D	-	1000	2000	1000	1000
ϵ_0	-	0.99	0.99	0.99	0.99
n_{lam}	-	-	-	2000	2000

Table 3: Parameter settings for adherence data experiment.

$$\begin{aligned}
&\min_{V^i(s^i,\lambda^i),\lambda^i} \sum_{i=0}^{N-1} \frac{\lambda^i B}{1-\beta} + \sum_{i=0}^{N-1} \mu^i(s^i) V^i(s^i,\lambda^i) \\
&\text{s.t. } V^i(s^i,\lambda^i) \geq r^i(s^i) - \lambda^i c_j + \beta \sum_{s^{i\prime}} T(s^i, a^i_j, s^{i\prime}) V^i(s^{i\prime},\lambda^i) \\
&\qquad \forall i \in \{0,...,N-1\}, \quad \forall s^i \in \mathcal{S}, \quad \forall a_j \in \mathcal{A} \\
&\quad r^i(s^i_{cur}) - \lambda^i c_k + \beta \sum_{s^{i\prime}} T(s^i_{cur}, a^i_k, s^{i\prime}) V^i(s^{i\prime},\lambda^i) = \\
&\quad r^i(s^i_{cur}) - \lambda^i c_{k-1} + \beta \sum_{s^{i\prime}} T(s^i_{cur}, a^i_{k-1}, s^{i\prime}) V^i(s^{i\prime},\lambda^i) \\
&\qquad \forall i \in \{0,...,N-1\} \\
&\quad \lambda^i \geq 0 \quad \forall i \in \{0,...,N-1\}
\end{aligned} \tag{17}$$

The LP is similar to Eq. 11, but differs in two ways. First, instead of having a single λ value across all arms, each arm has its own independent λ^i value. Second, the second group of constraints is new, and forces the λ^i values to be set such that the planner would be indifferent between taking the action in question a_k or the action that is one step cheaper a_{k-1}, which follows exactly the definition of the multi-action indexes. Note that although the indexes can each be computed independently, for convenience, we compute the index for a given a_k for each arm simultaneously to reduce overhead, as given in the above LP.

The ActionKnapsackILP referenced in Algorithm 4 is the same as the modified knapsack given in Killian et al. [18], reproduced below:

$$\begin{aligned}
\max_{\boldsymbol{A}} \; & \sum_{i=0}^{N-1}\sum_{j=0}^{M-1} \boldsymbol{A}_{ij} Q_{\boldsymbol{s},\lambda_{ind}}(i, a_j) \\
\text{s.t.} \; & \sum_{i=0}^{N-1}\sum_{j=0}^{M-1} \boldsymbol{A}_{ij} c_j \le B \\
& \sum_{j=0}^{M-1} \boldsymbol{A}_{ij} = 1 \quad \forall i \in 0, \dots, N-1 \\
& \boldsymbol{A}_{ij} \in \{0, 1\}
\end{aligned} \tag{18}$$

where $Q_{\boldsymbol{s},\lambda_{ind}}(i, a_j)$ is the Q-function for each arm filtered to the current state of the arms, $\boldsymbol{s}$, and minimizing value λ_{ind}, as given by the penultimate line of Algorithm 4.

RandomAction, referenced in Algorithms 2 and 4, chooses random actions through the following iterative procedure: (1) randomly choose an arm with uniform probability, (2) randomly choose an action with probability inversely proportional to one plus its cost (must add one to avoid dividing by 0 for no-action). The procedure iterates until the budget is exhausted.

Algorithm 1: MAIQL Update

```
Data: $Q \in \mathbb{R}^{N\times|\mathcal{S}|\times(|\mathcal{A}|-1)\times|\mathcal{S}|\times|\mathcal{A}|}$,  // Need one copy
      of $Q[s,a]$ for each index on each arm
  $\lambda \in \mathbb{R}^{N\times|\mathcal{S}|\times(|\mathcal{A}|-1)}$,           // multi-action index
      estimates
  Batch, $C$,        // Experience tuples, action costs
  $t, \nu(\cdot)$,          // iteration, state-action counter
  $\mathcal{S}, \mathcal{A}, N$  // state space, action space, # of arms
Hyperparameters: $\beta, C, C', D$            // See maintext
for $(n, s, a, r, s') \in Batch$ do
    $\alpha = \frac{C}{\lceil \frac{\nu(s,a,n)}{D} \rceil}$
    for $i \in 0, \dots, |\mathcal{S}|$ do
        for $j \in 1, \dots, |\mathcal{A}|$ do
            $Q[n,i,j,s,a] \mathrel{+}= \alpha(r - C[a] * \lambda[i,j] + \beta *$
              $\max\{Q[n,i,j,s']\} - Q[n,i,j,s,a])$
    if $a \neq 0$ & $t \pmod{N} == 0$ then
        $\gamma = \frac{C'}{1 + \lceil \frac{\nu(s,a,n)\log\nu(s,a,n)}{D} \rceil}$
        $\lambda[s,a] \mathrel{+}= \frac{\gamma(Q[n,s,a,s,a]) - Q[n,s,a,s,a-1])}{C[a]-C[a-1]}$
return $Q, \lambda$
```

Algorithm 2: MAIQL Action Select

```
Data: $\lambda \in \mathbb{R}^{N\times|\mathcal{S}|\times(|\mathcal{A}|-1)}$,      // multi-action index
      estimates
  $\boldsymbol{s} \in \mathbb{R}^N$              // current state of all arms
  $t, N, B$   // current iteration, # of arms, budget
if EpsilonGreedy($t$) then
    return RandomAction()
else
    $\boldsymbol{a}$ = [0 for _ in range($N$)]
    $\lambda_f$ = FilterCurrentState($\lambda, \boldsymbol{s}$)  // $\lambda_f \in \mathbb{R}^{N\times(|\mathcal{A}|-1)}$
    for $i \in 0 \dots B$ do
        $i = \arg\max(\lambda_f[\boldsymbol{a}+1] - \lambda_f[\boldsymbol{a}])$ // $\boldsymbol{a}$ is a vector
          index, arg max ignores out of bounds
          indexes
        $a[i] \mathrel{+}= 1$
return $\boldsymbol{a}$
```

Algorithm 3: LPQL Update

```
Data: $Q \in \mathbb{R}^{N\times n_{lam}|\mathcal{S}|\times|\mathcal{A}|}$,      // Need one copy of
      $Q[s,a]$ for each of the $n_{lam}$ test points on
      each arm
  Batch, $C$,       // Experience tuples, action costs
  $\lambda_{\max}$,          // Max $\lambda$ at which to estimate $Q$
  $n_{lam}$,   // # of $\lambda$ points at which to estimate $Q$
  $\nu(\cdot)$                        // state-action counter
Hyperparameters: $\beta, C, D$              // See maintext
for $(n, s, a, r, s') \in Batch$ do
    $\alpha = \frac{C}{\lceil \frac{\nu(s,a,n)}{D} \rceil}$
    for $i \in 0, \dots, n_{lam}$ do
        $\lambda_p = \frac{i * \lambda_{\max}}{n_{lam}}$
        $Q[n,i,s,a] \mathrel{+}=$
          $\alpha(r - C[a] * \lambda_p + \beta * \max\{Q[n,i,s']\} - Q[n,i,s,a])$
return Q
```

EpsilonGreedy(t), also referenced in Algorithms 2 and 4, draws a uniform random number between 0 and 1 and returns true if it is less than $\epsilon_0 / \lceil \frac{t}{D} \rceil$ and false otherwise.

D MEDICATION ADHERENCE SETTING DETAILS

We used the following procedure to estimate transition probabilities from the medication adherence data from Killian et al. [19]. First, we specify a history length of L. This gives a state space of size 2^L for each arm. Then, for each patient in the data, we count all of the occurrences of each state transition across a treatment regimen of 6 months (168 days). If L was small (e.g., 1 or 2), we could take a frequentist approach and simply normalize these counts appropriately to get valid transition probabilities to sample for experiments. However, as the history length L gets larger, the number of non-zero entries in the count data for state transitions become large. We take two steps to account for this sparsity. (1) We run K-means clustering over all patients, using the count data as features, then

Algorithm 4: LPQL Action Select

```
Data: $Q \in \mathbb{R}^{N \times n_{lam} |\mathcal{S}| \times |\mathcal{A}|}$, // $Q$-functions for each
        of the $n_{lam}$ test points on each arm
  $s \in \mathbb{R}^N$                 // current state of all arms
  $\lambda_{\max}$,           // Max $\lambda$ at which $Q$ is estimated
  $n_{lam}$,  // # of $\lambda$ points at which $Q$ is estimated
  $t, \beta$                   // iteration, discount factor
  $N, C, B$         // # of arms, action costs, budget
if EpsilonGreedy($t$) then
    return RandomAction()
$a$ = [0 for _ in range ($N$)]
$Q_f$ = FilterCurrentState($Q, s$)   // $Q_f \in \mathbb{R}^{N \times n_{lam} \times |\mathcal{A}|}$
$\lambda_{ind} = -1$
/* The min of Eq. 11 occurs at the point where
   the negative sum of slopes of all $V^i = \max\{Q^i_\lambda\}$
   is $\leq B/(1-\beta)$, so we will iterate through our
   estimates of $Q^i_\lambda$ and stop our search at the
   first point where that is true. */
for $i \in 0, \ldots, n_{lam}$ do
    $\lambda_p^0 = \frac{i * \lambda_{\max}}{n_{lam}}$
    $\lambda_p^1 = \frac{(i+1) * \lambda_{\max}}{n_{lam}}$
    $m_V = \frac{\max_a\{Q_f[:, i+1]\} - \max_a\{Q_f[:, i]\}}{\lambda_p^1 - \lambda_p^0}$        // $m_V \in \mathbb{R}^N$
    if $\sum_n \{m_V\} \geq -\frac{B}{1-\beta}$ then
        $\lambda_{ind} = i$
        break
$a$ = ActionKnapsackILP($Q_f[:, \lambda_{ind}, :], C, B$)
return $a$
```

combine the counts for all patients within a cluster. Intuitively, the larger the K, the more "peaks" of the distribution of patient adherence modes we will try to approximate, but the fewer data points are available to estimate the distribution in each cluster — however, it may be desirable to have more clusters to allow for some samples to come from uncommon but "diverse" modes that may be challenging to plan for. In this paper, we set K to 10. (2) We then take a Bayesian approach, rather than a frequentist approach for sampling patients/processes from the clustered counts data. That is, we treat the counts as priors of a beta distribution, then sample transition probabilities from those distributions according to the priors. Finally, to simulate action effects, since actions were not recorded in the available adherence data, we scale the priors multiplicatively according to the index of the action, i.e., larger/more expensive actions increase the priors associated with moving to the adhering state.

In summary, to get a transition function for a single simulated arm in the medication adherence experimental setting, we do the following. First, randomly choose a cluster, with probability weighted by the number of patients in the cluster. Then, build up a transition matrix by sampling each row according to its own beta distribution with priors given by the counts data (i.e., actual observations of $s \rightarrow s'$ transitions), scaled by the action effects.

This process was desirable for producing simulated arms with transition functions tailored to resemble that of a real world dataset, while allowing for some randomness via the sampling procedure, as well as a straightforward way to impose simulated action effects. However, one downside of this approach is that, since each row of the transition matrix is sampled independently, this may produce simulated arms whose probability of adherence changes in a non-smooth manner as a function of history. For example, in the real-world, we would expect that $P(0111 \rightarrow 1111)$ is correlated with $P(1011 \rightarrow 0111)$ and that $P(0000 \rightarrow 0000)$ is correlated with $P(1000 \rightarrow 0000)$, but our procedure would not necessarily enforce these relationships if there were not sufficient occurrences of each transition in the counts data.

The python code used to execute this procedure is included in the repository at https://github.com/killian-34/MAIQL_and_LPQL.