Fast Intervention Scheduling via Lagrangian Solutions to Multi-Action Restless Bandits

Jackson A. Killian,[1] **Andrew Perrault,** [1] **Milind Tambe** [1]
[1] Harvard University, Cambridge, MA, USA
jkillian@g.harvard.edu, aperrault@g.harvard.edu, milind_tambe@harvard.edu

Abstract

This paper presents new algorithms and theoretical results for solutions to *Multi-action* Multi-armed Restless Bandits, an important but insufficiently studied generalization of traditional Multi-armed Restless Bandits (MARBs). Though MARBs are popular for modeling problems in many domains, including healthcare, they are restricted to binary actions. This renders them unable to capture critical complexities faced by real planners, e.g., when health workers must decide among various interventions to induce behavior change in their patients. Previous work on Multi-action MARBs has been limited only to specialized sub-problems. Here we derive multiple algorithms for use on general Multi-action MARBs using Lagrangian relaxation techniques, leading to the following contributions: (i) We develop BLam, a bound optimization algorithm which leverages problem convexity to quickly and provably converge to the well-performing Lagrange policy; (ii) We develop SampleLam, a fast sampling technique for estimating the Lagrange policy, and derive a concentration bound to investigate its convergence properties; (iii) We derive best and worst case computational complexities for our algorithms as well as our main competitor; (iv) We provide experimental results comparing our algorithms to baselines on simulated distributions, including one motivated by a real-world medication adherence intervention task. Our approach achieves significant, up to ten-fold speedups over more general methods without sacrificing performance.

1 Introduction

MARBs have been studied extensively for solving a diverse set of problems including machine replacement (Glazebrook, Ruiz-Hernandez, and Kirkbride 2006; Ruiz-Hernández, Pinar-Pérez, and Delgado-Gómez 2020), wireless network scheduling (Bagheri and Scaglione 2015; Modi, Mary, and Moy 2019), anti-poaching patrol scheduling (Qian et al. 2016), and more recently, healthcare (Lee, Lavieri, and Volk 2019; Mate et al. 2020; Bhattacharya 2018). In a MARB, a planner must select k out of N arms on which to act for each of L rounds in a way that maximizes reward produced by the arms. Further, the reward on each arm depends on the action as well as an internal state that evolves according to an independent two-action Markov Decision Process (MDP). It has been shown that this problem is, in general, PSPACE-hard to solve exactly (Papadimitriou and Tsitsiklis 1999), but highly effective heuristics are known to exist (Whittle 1988; Bertsimas and Niño-Mora 2000).

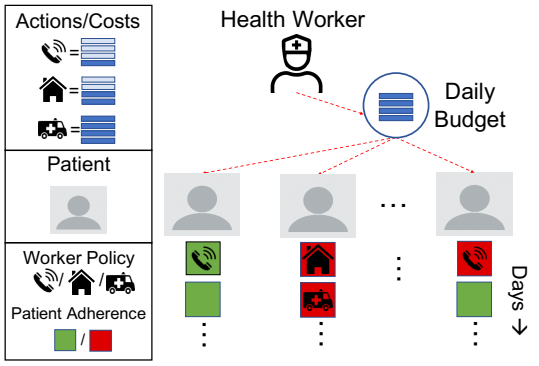


Figure 1: Medication adherence Multi-action MARB: resource-constrained health worker plans actions with various costs/effects to improve medication adherence of patient cohort.

However, a critical limitation of MARB frameworks is they only allow for 2 actions: act or not act. This is restrictive for many real-world cases where planners have various action choices with varying degrees of cost and effect. For example, in anti-poaching, the planner could allocate different levels of patrol effort to different targets, where more effort has higher cost and higher deterrent effect on poachers (Nguyen et al. 2013). In public health, a community health worker tasked with improving patient medication adherence could have several options for intervening, such as calling, visiting in person, or escalating patients to a more intense treatment (WHO et al. 2018). Traditional MARBs simply cannot model these complexities, restricting planners to a world where their only choices are to, e.g., call or not call. Rather, the planner needs to simultaneously optimize the use of all of the tools in their toolbelt each day, subject to a per-day time or cost budget B. This process is visualized for an example health task in Fig. 1.

To model such problems, we consider an under-examined generalization of MARBs that allow for multiple action types per arm, which we call *Multi-Action* MARBs ((MA)^{2}RBs). Previous work has considered extending the 2-action MARB notion of *indexability* and corresponding *index policies* to (MA)^{2}RBs (Glazebrook, Hodge, and Kirkbride 2011). In both 2-action and Multi-action MARBs, index policies are desirable because: (1) they decompose the problem in a manner that scales well and (2) when indexability holds, they are asymptotically optimal (Weber and Weiss 1990; Hodge and Glazebrook 2015). However, both

deriving index policies and verifying indexability is notoriously difficult, and largely requires special problem structure (Glazebrook, Ruiz-Hernandez, and Kirkbride 2006; Glazebrook, Hodge, and Kirkbride 2011) which may not exist, especially in healthcare settings where patients may have heterogeneous responses to different interventions. Our goal is thus to develop fast, well-performing policies for a broader class of (MA)^{2}RBs where no structure is assumed and indexability cannot be readily verified. We bypass the task of deriving index policies by taking a more general Lagrangian relaxation approach that leads to an auxiliary problem of computing a policy that minimizes the Lagrange bound. Computing this "Lagrange policy" is desirable because it recovers the index policies when they exist, but is readily computable regardless of problem structure.

The Weakly Coupled MDP (WCMDP) literature offers a method to compute the Lagrange policy for WCMDPs. Here, we recognize WCMDPs as a generalization of (MA)^{2}RBs and identify that this approach can be used to compute the Lagrange policy for (MA)^{2}RBs. However, the approach relies on solving a large linear program (LP) that scales quadratically in the number of arms and states. Setting up and solving this LP quickly becomes infeasible for large problem sizes as we show later in experiments. To address this issue, we investigate and utilize basic structural properties of general (MA)^{2}RBs to create scalable algorithms for computing the Lagrange policy on any (MA)^{2}RB problem, leading to the following contributions:
(i) Bound optimization algorithm: We develop BLam, an iterative bound optimization method for computing the Lagrange policy. BLam leverages problem convexity to derive progressively tighter upper and lower bounds on the Lagrange policy via a series of small LPs. We provide key technical results that prove this method converges to the policy that minimizes the Lagrange bound and provide experimental evaluation of its runtime.
(ii) Sampling algorithm: We develop a sampling-based algorithm, SampleLam, which trades off the guarantees of BLam for speed. SampleLam chooses a random subset of arms, rapidly computes a statistic about the desirability of allocating budget to each arm, then combines the statistics to construct an estimated Lagrange policy for the full problem. We derive a concentration bound to prove the method converges, then use insights from the bound to inform how the algorithm carries out sampling.
(iii) Complexity Results: We derive best- and worst-case computational complexities for our methods as well as our main competitor. BLam, achieves $\approx \sqrt{N}$ improvement and SampleLam achieves a factor of N improvement in the best case.
(iv) Experiments: We compare our algorithms to baselines on synthetic distributions with different underlying structure, including one motivated by a real-world medication adherence intervention task. Our algorithms scale up to ten times better than a more general baseline without sacrificing performance, and readily adapt to each problem with minimal tuning. Thus our work newly makes available multiple avenues for computing well-performing policies on new (MA)^{2}RBs at scale, including those in complex but critical domains such as health behavior change.

2 Related Work

Previous work extends the traditional MARB notion of indexability to (MA)^{2}RBs (Glazebrook, Hodge, and Kirkbride 2011; Hodge and Glazebrook 2015). However, their analysis is restricted to a subclass of (MA)^{2}RBs with special monotonic structure, whereas we build algorithms for general (MA)^{2}RBs. "Superprocesses" are an alternative multi-action extension where a primary planner distributes a limited set of sub-planners who act on arms without constraint (Mahajan and Teneketzis 2008; Zayas-Caban, Jasin, and Wang 2019; Verloop et al. 2016). This structure does not generally apply to (MA)^{2}RBs since they do not constrain the number of agents that can be acted on each round. (Meshram and Kaza 2020) designs a Monte-Carlo rollout approach for estimating 2- and multi-action MARB policies when a restricted set of "threshold" policies are optimal, but our algorithms do not assume this structure.

Also related are WCMDPs in which a planner operates N independent MDPs subject to a set of arbitrary constraints over actions. (Meuleau et al. 1998) derive methods for handling "global" resource constraints over all rounds, whereas we address round-by-round constraints. (Hawkins 2003), the main baseline we compare against, derive a Lagrangian relaxation on the general form of a WCMDP and give an LP for minimizing the Lagrange bound. In contrast, we leverage the single constraint nature of (MA)^{2}RBs to greatly speed up the computation of the Lagrange bound compared to (Hawkins 2003). (Adelman and Mersereau 2008) give an approximate dynamic programming method that achieves a tighter bound and better performing policies than the Lagrange approach to WCMDPs. However, it scales exponentially, restricting it to small problem sizes.

Finally, our work is related to a large body of work developing Lagrangian methods for solving traditional MARBs (Whittle 1988; Nino-Mora 2001; Glazebrook, Ruiz-Hernandez, and Kirkbride 2006). We generalize these settings to allow for multiple actions. Moreover, the methods we develop here reduce in the 2-action case to the widely-used, well-performing, Whittle index policy (Whittle 1988).

3 Preliminaries

A (MA)^{2}RB consists of a set of N arms, each associated with a Markov Decision Process (MDP) (Puterman 2014). An MDP $\{\mathcal{S}, \mathcal{A}, r, T, \beta\}$ consists of a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a state-dependent bounded reward function $r : \mathcal{S} \to \mathbb{R}$, a transition function T, where $T(s,a,s')$ gives the probability of transitioning to state s' when action a is taken from state s, and a discount factor $\beta \in [0,1)$. An MDP *policy* $\pi : \mathcal{S} \to \mathcal{A}$ maps states to actions. The long-term *discounted reward* starting from state $s_0 = s$ is defined as

$$R^{\pi}_{\beta}(s) = E\left[\sum_{t=0}^{\infty} \beta^t r(s_{t+1} \sim T(s_t, \pi(s_t), s'_t)) | \pi, s_0 = s\right] \quad (1)$$

Each arm i in a (MA)^{2}RB is an MDP with an action set $\mathcal{A}^i$ of size M^i and corresponding action cost vector $\boldsymbol{C}^i \in \mathbb{R}^{M^i}$. We assume all action sets and costs are the same for all arms (and henceforth drop the subscript i), but all techniques in this paper extend in a straightforward manner to general action sets and costs. Without loss of generality, we assume that the elements c_j of $\boldsymbol{C}$ are ordered ascending. Also, to align with the standard bandit assumption that an arm can be "not played" at no cost, we set $c_0 = 0$. Each round, the planner must select one action for each of the N arms such that the sum cost of all actions do not

exceed a budget B. Formally, the planner must choose a decision matrix $\boldsymbol{X} \in \{0,1\}^{N \times M}$ with elements denoted $x_{i,j}$ such that

$$\sum_{j=0}^{M-1} x_{i,j} = 1 \quad \forall i \in 0...N-1 \tag{2}$$

$$\sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x_{i,j} c_j \leq B \tag{3}$$

where the first constraint enforces one action per arm and the second enforces the budget. The planner's goal is to maximize their discounted reward across the arms over time, subject to these constraints. Let $\boldsymbol{s} = [s^0, s^1, ..., s^{N-1}]$ represent the vector of all arm states. The planner's goal can be represented by the following constrained Bellman equation

$$J(\boldsymbol{s}) = \max_{\boldsymbol{X}} \{ \sum_{i=0}^{N-1} r^i(s^i) + \beta E[J(\boldsymbol{s}') | \boldsymbol{s}, \boldsymbol{X}] | \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x_{i,j} c_j \leq B \} \tag{4}$$

While Eq. 4 could be solved directly via value iteration, $J(s) \in \mathcal{S}^N$, and the number of feasible actions over which to take the max for each $J(\boldsymbol{s})$ is also exponential in N, making this approach intractable for non-trivial problem sizes. The key insight, though, is that the value functions and actions are only coupled due to the shared budget constraint over all arms. Therefore to simplify the problem, we relax the budget constraint and add it as a penalty to the objective with a Lagrange multiplier λ as follows:

$$J(\boldsymbol{s}, \lambda) = \max_{\boldsymbol{X}} \{ \sum_{i=0}^{N-1} r^i(s^i) + \lambda (B - \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x_{i,j} c_j) + \beta E[J(\boldsymbol{s}')] \} \tag{5}$$

A straightforward argument by induction shows that the value functions decouple as desired, giving:

$$J(\boldsymbol{s}, \lambda) = \frac{\lambda B}{1-\beta} + \sum_{i=0}^{N-1} V^i(s^i, \lambda) \tag{6}$$

$$\text{where } \forall i, V^i(s^i, \lambda) = \max_{a_j^i \in \mathcal{A}} \{ r^i(s^i) - \lambda c_j + \beta \sum_{s^{i\prime}} T(s^i, a_j^i, s^{i\prime}) V^i(s^{i\prime}, \lambda) \} \tag{7}$$

See (Adelman and Mersereau 2008) for a complete proof. Notice that for a given value of λ, Eq. 6 can be solved using a fast method like value iteration to solve for the individual V^is, where V^i and its corresponding actions are now in $\mathcal{S}$ and $\mathcal{A}$, respectively. However, the choice of λ will be critical when using the resulting value functions to derive policies for our bandits. For instance, $\lambda = 0$ would correspond to ignoring the budget constraint while planning which clearly will not be optimal in general. Alternatively, as $\lambda \to \infty$, the optimal policy in each value function is to never act since all actions will have effectively infinite cost except for $c_0 = 0$. To gain insight about how to set the value of λ we recast the problem as an LP, rewriting Eq. 6 by leveraging the known LP solution to the value function (Puterman 2014):

$$\begin{aligned} J(\boldsymbol{s}, \lambda) = & \min_{V^i(s^i,\lambda), \lambda} \frac{\lambda B}{1-\beta} + \sum_{i=0}^{N-1} \mu^i(s^i) V^i(s^i, \lambda) \\ & \text{s.t. } V^i(s^i, \lambda) \geq r^i(s^i) - \lambda c_j + \beta \sum_{s^{i\prime}} T(s^i, a_j^i, s^{i\prime}) V^i(s^{i\prime}, \lambda) \\ & \forall i \in \{0, ..., N-1\}, \ \forall s^i \in \mathcal{S}, \ \forall a_j \in \mathcal{A}, \text{ and } \lambda \geq 0 \end{aligned} \tag{8}$$

Where $\mu^i(s^i) = 1$ if s^i is the start state for arm i and is 0 otherwise. That we minimize over λ and that $\lambda \geq 0$ is a classic Lagrangian result, motivated by making $J(\boldsymbol{s}, \lambda)$ a tight-as-possible upper bound on $J(\boldsymbol{s})$. Intuitively, and matching how problem-specific index policies have been derived in previous work (Whittle 1988; Glazebrook, Hodge, and Kirkbride 2011), we want to derive a policy from the V^is that provide the tightest bound on $J(s)$. So that our algorithms can generally apply to any (MA)^{2}RB, our approaches will solve Eq. 8 in its general form.

The above derivation was first given by (Hawkins 2003) for WCMDPs, and as suggested therein, clearly one can directly solve Eq. 8 using any LP solver. However, Eq. 8 has $N|\mathcal{S}|+1$ variables and $N|\mathcal{S}||\mathcal{A}|$ constraints. Further, the current lowest known computational complexity for solving an LP is $\mathcal{O}(n^{2+\frac{1}{18}})$, where n is the number of variables (Jiang et al. 2020), implying that directly solving Eq. 8 has computational complexity $\approx \mathcal{O}(N^2 |S|^2)$ (derived in section 4). The key to our approach will be separating the computation of the λ that minimizes Eq. 8, henceforth λ_{min}, and the corresponding V^is that solve Eq. 8 in a way that provides vast speedups. Herein we derive exact and heuristic methods for computing λ_{min}, each of which has an improved best case complexity in N by a factor of $\sqrt{N}$ or better.

4 Bound Optimization With BLam

BLam is our exact approach to computing the Lagrange policy. We first give an overview, noting theorems where relevant that are derived in the next section. The main idea is rooted in the form of the functions $V^i(s^i, \lambda)$ in Eq. 8, visualized in blue in Fig. 2. To exactly compute Eq. 8 requires adding $|\mathcal{S}||\mathcal{A}|$ constraints and $|\mathcal{S}|$ variables to the LP for each of the N arms' value functions $V^i(s^i, \lambda)$. Instead, we will build special approximations to each $V^i(s^i, \lambda)$ that are represented in the LP each with just one variable and a constant number of constraints, achieving vast speedups. The approximations are constructed by rapidly testing for the slope of $V^i(s^i, \lambda)$ at various test points λ_{test} using value iteration, then creating a piecewise linear combination of the slopes. The key is we construct two special types of approximations: one that upper bounds the slope of $V^i(s^i, \lambda)$ and one that lower bounds it, shown in Fig. 2 in green and red, respectively.

We then use the insight that the $V^i(s^i, \lambda)$ in Eq. 8, are indeed convex decreasing functions of λ (Prop. 4.1), implying that Eq. 8 is minimized when the combined per-unit *decrease* to the objective brought by the convex $V^i(s^i, \lambda)$ functions is equal to or less than the constant per-unit *increase* to the objective brought by $\frac{\lambda B}{1-\beta}$. In other words, λ_{min} is the point where the negative sum of slopes of $V^i(s^i, \lambda)$ is equal to $\frac{B}{1-\beta}$ (Prop. 4.2). Crucially, if we replace any $V^i(s^i, \lambda)$ with a convex function with strictly more negative slope (i.e., a lower bound), the value of λ at which the negative sum of slopes equals $\frac{B}{1-\beta}$ could only increase, giving

Algorithm 1: BLamPrecompute

Data: T,R,C,N,G,β
$\mathcal{D}=[]$; // hold slopes at arm test points
$\epsilon_0=$1e-3;
for $i=1,...,N$ **do**
  for $j=1,...,|G[i]|$ **do**
    $\lambda_{test}=G[i,j]$;
    $R_\lambda,R_{\lambda+\epsilon_0}=R[i]$;
    for $x\in 1,...,|C|$ **do** subtract action costs
      $R_\lambda[x]-=\lambda_{test}*C[x]$;
      $R_{\lambda+\epsilon_0}[x]-=(\lambda_{test}+\epsilon_0)*C[x]$;
    $\mathcal{D}[i,j]$ = (VI($T[i],R_{\lambda+\epsilon_0},\beta$) - VI($T[i],R_\lambda,\beta$))/$\epsilon_0$
$\mathcal{U},\mathcal{L}$ = BuildBounds($\mathcal{D}$);
return $\mathcal{U},\mathcal{L}$

an upper bound on λ_{min}. The converse also holds, i.e., replacing with upper bound convex functions gives a lower bound on λ_{min} (Thm. 4.3). This constitutes the core tradeoff in our approach: the more $V^i(s^i,\lambda)$ are replaced with approximations in the LP, the faster it will execute, but the looser the bounds will be. We handle this by first "bounding out", i.e., replacing $V^i(s^i,\lambda)$ with its approximation, all but a small number K processes to get loose bounds on λ_{min} rapidly. We then iteratively add back $V^i(s^i,\lambda)$s to the LP until the bounds on λ_{min} are with a pre-specified ϵ. With minimal tuning, the test points can be set to create tight enough bounds that BLam will converge after only a small number of iterations, leading to great speed increases.

The algorithm proceeds in two parts. In BLAMPRECOMPUTE, given in Alg. 1, we compute the upper and lower bound approximations of of the arms, passing in the MDP parameters of the arms, along with a list G of points λ_{test} at which to approximate the slopes. VI in Alg. 1 denotes value iteration and $\mathcal{U}/\mathcal{L}$ will contain the pieces of the piecewise upper and lower bounds for $V^i(s^i,\lambda)$ for all arms and states. BLAMPRECOMPUTE runs once at the beginning of simulation.

BLAM, given in Alg. 2, runs on each round of the (MA)^{2}RB to compute λ_{min} for the current set of arm states $\boldsymbol{s}$ (line 2 of Alg. 2 selects the bounding functions for the current state of each arm). Using the piecewise bounded versions of $V^i(s^i,\lambda)$, it constructs a special LP, BLAMLP, given in Eq.9 below, that produces upper and lower bounds on λ_{min} by replacing $V^i(s^i,\lambda)$ with their bounded counterparts. It loops, replacing successively more $V^i(s^i,\lambda)$ in lieu of their bounded forms, until the resulting bounds on λ_{min} are within ϵ. BLAM terminates by running one final value iteration with the appropriate λ_{min}, the result of which solves Eq. 8 *without constructing or solving the full LP, leading to vast speed ups*. The resulting value functions will be used to construct a final policy in section 6.

BLam: Derivation

To bound the slope of $V^i(s^i,\lambda)$, we rely on it having a convex form.

Proposition 4.1. *$V^i(s^i,\lambda)$ is convex decreasing in λ, and as $\lambda\to\infty$, $\frac{dV^i(s^i,\lambda)}{d\lambda}\to 0$*

Proof. This follows directly from Eq. 7, but can be shown via induction that since $V^i(s^i,\lambda)$ is a max over piecewise linear convex functions of λ, it is also piecewise linear convex, and since $\lambda c_j\geq 0$, it must be weakly decreasing in λ. Furthermore, since $c_0=0$ in (MA)^{2}RBs, as $\lambda\to\infty$, the one time charge λc_j of any action a_j s.t. $j>0$ becomes greater than any long-term achievable reward, therefore the optimal policy must always choose not to act. At that point, $V^i(s^i,\lambda)=E[\sum_{t=0}^{\infty}\beta^t r(s)|\pi(a)=0,\forall a]$ which does not depend on λ. □

Algorithm 2: BLam

Data: T,R,C,N,B,β, s, $G,\mathcal{U},\mathcal{L}$, ϵ, kStep
/* Only get bounds for current states */
GetCoeffsForState($\mathcal{U}$, $\mathcal{L}$, s);
Sort($\mathcal{U}$, $\mathcal{L}$, T, R);
st = PickStart($\mathcal{L}$,$\sqrt{N}$);
for $k\in$[*st, st+kStep, ..., N*] **do**
  λ_u=BLamLP($T[:k],R[:k],B,C,\beta,s,\mathcal{L}$);
  λ_ℓ=BLamLP($T[:k],R[:k],B,C,\beta,s,\mathcal{U}$);
  if $\lambda_u-\lambda_\ell\leq\epsilon$ **then** break ;
$V(i,s)=[]$ // $Nx|\mathcal{S}|$ array of value functions
$\lambda_{min}=(\lambda_u-\lambda_\ell)/2$;
for $i=1,...,N$ **do**
  $R_\lambda=R[i]$;
  for $x\in 1,...,|C|$ **do** subtract action costs
    $R_\lambda[x]-=\lambda_{min}*C[x]$;
  $V[i]$=VI($T[i],R_\lambda,\beta$);
return V

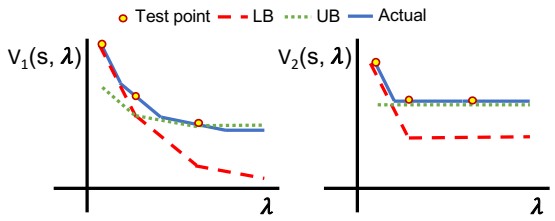


Figure 2: Constructing bounds on the slope of $V^i(s^i,\lambda)$ for two different arms with three test points. Note: bounds are with respect to the *slope*, not the value of the function.

Let λ_u (λ_ℓ) correspond to the λ which solves Eq. 8 when $V^i(s^i,\lambda)$ are replaced in the objective by $\mathcal{L}$ ($\mathcal{U}$). Note that the lower bound functions $\mathcal{L}$ will be used to derive *upper bounds* on the value of λ_{min} and vice versa.

Next, we give a helpful intermediate result.

Proposition 4.2. *The optimal solution to Eq. 8 will be found at the value of λ in which the negative sums of the slopes of $V^i(s^i,\lambda)$ w.r.t. λ become less than or equal to $\frac{B}{1-\beta}$.*

Proof. Assume λ^* corresponds to an optimal solution to Eq. 8 and the negative sums of the slopes of convex decreasing $V^i(s^i,\lambda)$ are greater than $\frac{B}{1-\beta}$. Then λ^* can be increased by ϵ and the objective value would decrease, i.e., $J(\boldsymbol{s},\lambda^*+\epsilon)<J(\boldsymbol{s},\lambda^*)$ giving a contradiction. □

We now can prove our main result:

Theorem 4.3. $\lambda_\ell \le \lambda_{min} \le \lambda_u$

Proof. The proof is best seen by considering λ_{min} which solves $J(\boldsymbol{s},\lambda)$, i.e., Eq. 8. We start with $\lambda_{min} \le \lambda_u$: Let $\mathcal{V}$ denote the set of $V^i(s^i,\lambda)$ in the objective of Eq. 8. Further, let $\mathcal{V}^b$ denote the set of $V^i(s^i,\lambda)$ which will be replaced by $\mathcal{L}^b \subset \mathcal{L}$. Now replace all $\mathcal{V}^b$ with their corresponding $\mathcal{L}^b$, name this new LP $J^{\lambda_u}(\boldsymbol{s},\lambda)$ and name its optimal solution λ_u. By definition, at all values of λ, the slope of $V^i(s^i,\lambda)$ is greater than the slope of $\mathcal{L}^b$. Thus, at λ_{min}, the negative sums of the slopes of $V^i \in \mathcal{V}\backslash\mathcal{V}^b$ plus $\mathcal{L}^b$ is weakly greater than the negative sums of the slopes of $V^i \in \mathcal{V}$. By Prop. 4.2, we must have that $J^{\lambda_u}(\boldsymbol{s},\lambda) \le J(\boldsymbol{s},\lambda)$, and respectively $\lambda_u \ge \lambda_{min}$.
$\lambda_{min} \ge \lambda_l$: The proof follows similarly. □

We now describe a quick method for computing $\mathcal{U}$ and $\mathcal{L}$. To construct these piecewise linear convex functions that will serve as the bounds, we make use of the insight that the slope of $V^i(s^i,\lambda)$ can be rapidly computed at any *test point* λ_{test} by calculating $(V^i(s^i,\lambda=\lambda_{test}+\epsilon_0)-V^i(s^i,\lambda=\lambda_{test}))/\epsilon_0$ where both $V^i(s^i,\lambda)s$ can be quickly computed via value iteration and $\epsilon_0 \approx 0$. Let G^i represent the set of test points for a given arm. The larger G^i, the tighter the bounds will be, but the higher the up-front computational cost. Thus the choice of both the size and the exact elements of G^i represent a set of parameters that can be tuned to maximize performance on a given distribution of V^is. However, at minimum, G^i must include $\lambda_{test}=0$ since proposition 4.1 implies that the minimum slope of any V^i occurs at $\lambda=0$. Once the slope at all the test points are computed, they are used to construct $\mathcal{U}$ and $\mathcal{L}$ via standard linear equations—the exact process is recorded in Appendix 9. Let $\mathcal{U}^i(G^i_k,m)$ and $\mathcal{U}^i(G^i_k,b)$ be the slopes and intercepts, respectively, for each piece k of the upper bounding function $\mathcal{U}^i$ for arm i. Define $\mathcal{L}^i(G^i_k,*)$ similarly.

Now, we can compute λ_u and λ_ℓ. To start, we choose K arms to include in Eq. 8 in their $V^i(s^i,\lambda)$ form, while the other $N-K$ arms will be replaced by their bounded counterparts. To compute λ_u, we replace the $N-K$ arms with $\mathcal{L}$ to get the following LP:

$$
\begin{aligned}
J^{\lambda_u}(\boldsymbol{s},\lambda) = \min_{V^i,\lambda,z^j} \; & \frac{\lambda B}{1-\beta} + \sum_{i=0}^{K} \mu^i(s^i) V^i(s^i,\lambda) + \sum_{j}^{N-K} z^j \\
\text{s.t. } & V^i(s^i,\lambda) \ge r^i(s^i) - \lambda c_j + \beta \sum_{s^{i\prime}} T(s^i,a^i_j,s^{i\prime}) V^i(s^{i\prime},\lambda) \\
& \qquad \forall i \in \{0,...,K\}, \;\; \forall s^i \in \mathcal{S}, \;\; \forall a_j \in \mathcal{A} \qquad (9) \\
& z^j \ge \mathcal{L}^j(G^j_k,m) * \lambda + \mathcal{L}^j(G^j_k,b) \\
& \qquad \forall k \in \{0,...,|G^j|\}, \;\; \forall j \in \{0,...,N-K\} \\
& \lambda \ge 0
\end{aligned}
$$

where z^j are auxiliary variables to represent the piecewise linear convex functions $\mathcal{L}^j$ via the $|G^j|$ constraints on z^j. To compute λ_ℓ we construct a similar LP using $\mathcal{U}^i(G^i_k,*)$.

One important choice is in selecting the first K arms. Intuitively, the best $V^i(s^i,\lambda)$ to include in Eq. 9 are those with the loosest bounds. One proxy for looseness is the slope of the last segment, i.e., the steeper the slope, the looser the bound, since we know the slope of all $V^i(s^i,\lambda)$ go to 0 eventually (Prop. 4.1). Therefore, we first sort arms in ascending order by this criteria (line 3 in Alg. 2). To set K, we note that Prop. 4.2 implies that the negative sum of slopes of $V^i(s^i,\lambda)$ and $\mathcal{L}^i$ must be less than or equal to $B/(1-\beta)$ for some value of λ to find a solution. Since $\mathcal{L}^i$ are convex decreasing, if the negative sum of slopes of all the *trailing* segments of $\mathcal{L}^i$ are greater than $B/(1-\beta)$, then the LP will be unbounded. Thus, to guarantee the existence of a bounded solution, we set K to pick the first K arms in slope sorted order, such that the negative sum of slopes of all the trailing segments of $\mathcal{L}^i$ is less than $B/(1-\beta)$. We then set $K = \max(K,\sqrt{N})$ (line 4 Alg. 2).

Once λ_u and λ_ℓ are computed once, we iterate to include K_{step} more arms in the LP such that $K \mathrel{+}= K_{step}$ then repeat until the algorithm converges to within a difference ϵ. A straightforward induction argument shows that as K grows (and the set of bounded arms shrinks), the bounds become progressively tighter and are guaranteed to be exact when $K=N$. Once λ_{min} is determined, we use value iteration to rapidly solve Eq. 8, the result of which we will use to derive feasible policies in Section 6.

BLam: Computational Complexity

In BLAMPRECOMPUTE, BLam computes $\mathcal{U}^i(G^i_k,*)$ and $\mathcal{L}^i(G^i_k,*)$ for all $V^i(s^i,\lambda)$, which requires two runs of value iteration for each arm for each test point G^i_k. Assuming all arms use the same number of test points, states and actions, this scales as $\mathcal{O}(NG^iVI(|\mathcal{S}|,|\mathcal{A}|))$ where $VI()$ is the computational complexity of value iteration. While an exact complexity of value iteration is elusive, it is known to be much faster than the LP formulation (Puterman 2014). Thus, its complexity will be dominated by the LP solves that occur in BLAM— the same applies for the value iteration that runs at the end of BLAM each round.

To compute a policy for each round, BLAM constructs Eq. 9 as an LP which has $K|S|+(N-K)$ variables, $K|S||A|$ constraints with $|S|$ terms, and $(N-K)G^i$ constraints with two terms. Although the constraints associated with the $(N-K)$ auxiliary variables only have two non-zero coefficients, we conservatively assume that the matrix for this LP is dense in order to adopt the best known LP complexity result (Jiang et al. 2020). In the best case, BLam would provide tight bounds on λ_{min} after just one iteration. So setting $K=\sqrt{N}$ and assuming $G^i \ll N$, the per-round complexity is

$$\Omega(\sqrt{N}|S|^2|A|+N|S|^2+N^{\frac{3}{2}}|S|+N^2) \qquad (10)$$

Where the first term is the LP setup time to add constraints (dominates the time to add variables) and the last three terms are the LP solve complexity, which is approximately square in the number of variables. Applying the same reasoning to the direct LP solve approach, which has $N|S|$ variables and $N|S||A|$ constraints gives the following best (and worst) case complexity

$$\mathcal{O}(N|S|^2|A|+N^2|S|^2) \qquad (11)$$

Thus, BLam has a strictly better best-case complexity in the problem size. However, in the worst case, setting $K_{step}=\sqrt{N}$, BLam would require the full $\sqrt{N}$ iterations to get a tight bound on λ_{min}. In this case, the LP setup time would match the naive LP approach, but successive solves would become more expensive. Using basic summation, this gives a worst-case complexity of

$$\mathcal{O}(N|S|^2|A|+N^{\frac{5}{2}}|S|^2) \qquad (12)$$

Algorithm 3: SampleLam

Data: T,R,C,N,B,β, r_{max}, c_{min}

$N_{samples} = \frac{\log(N) r_{max}}{c_{min}}$;
inds = RandomChoice([1,...,N],$N_{samples}$);
T, R = T[inds], R[inds];
$\lambda^i_{list} = []$;
for $i = 1...,N_{samples}$ **do**
  λ^i = QuickLP($T[i]$,$R[i]$,$\frac{B}{N}$,C,β);
  λ^i_{list}.append(λ^i)
$V(i,s) = []$ // $Nx|\mathcal{S}|$ array of value functions
λ_{min} = Mean(λ^i_{list});
for $i = 1,...,N$ **do**
  $R_\lambda = R[i]$;
  for $x \in 1,...,|C|$ **do** subtract action costs
    $R_\lambda[x] -= \lambda_{min} * C[x]$;
  $V[i]$ = VI($T[i]$,R_λ,β);
return V

Which, handily, is only $\sqrt{N}$ worse than the naive approach. However, we show in experiments that the typical run time and scaling of BLam is much faster than the naive approach in practice.

5 SampleLam

In some cases, especially very large problem sizes, speed can be more critical than performance. Thus, next, we give an algorithm for quickly computing a heuristic estimate of λ_{min} based on sampling. The approach is grounded in the mathematical interpretation of λ_{min}, i.e., that λ_{min} captures the willingness to violate the budget B given all V^i processes. In this algorithm, we will estimate our willingness to violate the budget for each arm *individually* given equal shares of the budget, solving a series of singleton LPs, then combine that knowledge to generate an estimate for λ_{min}

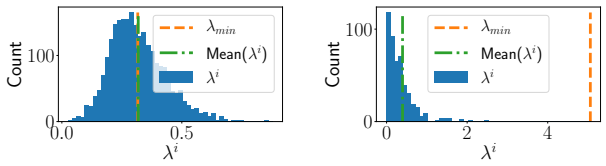
The algorithm is given in Alg. 3. It first chooses K processes at random to run through QUICKLP, which solves Eq. 8 with a single arm and modified budget $\frac{B}{N}$ giving a value λ^i that estimates the value of playing that arm. It then creates an estimate of λ_{min} by taking the sample mean of λ^i. Finally, it uses this estimate plus value iteration to solve for Eq. 8, which again we will use to derive feasible policies in section 6.

Although this method is not guaranteed to converge to the value of λ_{min}, it is very fast, and works well in practice on distributions which have an approximately normal distribution of budget across arms under the true λ_{min} policy. Fig. 3a shows such a distribution and the SampleLam estimate of λ_{min}.

SampleLam: Concentration Bound and Complexity

To understand SampleLam's convergence properties, i.e., convergence to the sample mean of λ^i, we derive a concentration bound below. The derivation first relies on showing that the distribution of λ^is is sub-Gaussian.

Theorem 5.1. *λ^i are $\frac{\sigma^2}{n}$-sub-Gaussian where* $\sigma^2 = \frac{1}{4}\left(\frac{r_{max}}{c_{min}(1-\beta)}\right)^2$



(a) (b)

Figure 3: (a) λ^i is normally distributed about a mean equal to λ_{min} (b) λ_{min} is determined by a select few "important" arms, not equal to the sample mean. BLam is better suited for this case.

The proof involves showing $0 \leq \lambda^i \leq \frac{r_{max}}{c_{min}(1-\beta)}$ and is given in Appendix 10. We can now use sub-Gaussianness to derive a concentration bound relating the number of samples to a confidence parameter $1-\delta$ on the estimate of the sample mean of λ^i.

Theorem 5.2. *The number of samples n needed to estimate the sample mean of λ^i within an error ϵ and with confidence $1-\delta$ is lower bounded as:*

$$n \geq \frac{1}{2\epsilon^2}\left(\frac{r_{max}}{c_{min}(1-\beta)}\right)^2 \log\left(\frac{1}{\delta}\right) \tag{13}$$

Proof.

$$P(\hat{\lambda^i} \geq \lambda^i + \epsilon) \leq \exp\left(-\frac{n\epsilon^2}{2\sigma^2}\right) \leq \delta \quad \text{Hoeffding bound} \tag{14}$$

$$n \geq \frac{2\sigma^2}{\epsilon^2}\log\left(\frac{1}{\delta}\right) \tag{15}$$

$$n \geq \frac{1}{2\epsilon^2}\left(\frac{r_{max}}{c_{min}(1-\beta)}\right)^2 \log\left(\frac{1}{\delta}\right) \tag{16}$$

□

Where the last step uses Thm. 5.1. This bound gives the insight that the greater the reward to cost ratio, the more samples we need to well-estimate the mean. We account for this by including a factor of $\frac{r_{max}}{c_{min}}$ in the setting for K in the SampleLam algorithm. The drawback of this approach is that λ_{min} is not guaranteed to be close to the sample mean of λ^i in general. An adversarial case is shown in Fig. 3b in which SampleLam would compute an arbitrarily bad estimate for λ_{min}. Setting $K = \log(N)\frac{r_{max}}{c_{min}}$, the best and worst case complexity for SampleLam is:

$$\mathcal{O}\left(\log(N)\frac{r_{max}}{c_{min}}|S|^2|A| + N * VI(|\mathcal{S}|,|\mathcal{A}|)\right) \tag{17}$$

Where the first term is the cost of building $\log(N)\frac{r_{max}}{c_{min}}$ LPs (dominates solve time), and the second is the value iteration cost.

6 Computing a Policy

Finally, once λ_{min} is finalized, and the resulting value functions from Eq. 8 have been computed, we use the value functions to compute the one-step greedy policy implied by the bound. To do this, we expand the value functions to compute the *action-value function*, Q, which captures the long term value for acting in

a given state in each arm. We then choose actions by solving a modified knapsack where $Q^i(s^i,a,\lambda_{min})$ are the values subject to their respective action costs, the budget B, and a constraint that ensures only one action is taken per arm. The knapsack LP is given in Appendix 11, with an algorithm for computing $Q^i(s^i,a,\lambda_{min})$ from value functions.

7 Experiments

We test our algorithms on two synthetic settings. In each, we compare the discounted sum of rewards, using discount factor 0.95, averaged over all arms N, over $L=40$ rounds. All results are averaged over 25 simulations. We compare our methods against the following baselines: **Nobody**: Take a_0 which has no cost on every arm; **VfNc**: Solves Eq. 8 with $\lambda=0$, effectively ignoring all future constraints, then follows Section 6; **Hawkins**: Solves Eq. 8 directly using an LP solver, then follows Section 6. We also include several versions of BLam using various stopping criterion ϵ, noted in each plot as BLam$\{\epsilon\}$. Larger ϵ will lead to faster running times but looser bounds on λ_{min}, and thus worse performance in general. All algorithms were implemented in Python 3.6 and use Gurobi version 9.0.3 to solve LPs via the gurobipy interface. All value iterations were computed using a lightly modified version of pymdptoolbox version 4.0b3 (Cordwell 2015). Code will be made available upon publication.

First, we explore an example distribution where VfNc and Haw-kins fail arbitrarily in terms of performance and runtime respectively. In this distribution, there are three types of agents: **(1) Greedy:** Must take increasingly expensive actions to collect increasingly high reward. Once the required action is not taken, the agent never produces reward again. This is modeled with a single chain of states, each with unit-increasing reward, reachable only by an action with unit-increasing cost. Failure to take the next action leads to a dead state. **(2) Reliable:** Must take the cheapest non-zero action every round to achieve reward 1. If the arm is not played for any round, it never produces reward again. This is modeled with a simple 2-state chain, in which the final state recurs with the proper action, otherwise it goes to a dead state. **(3) Easy:** Always gives reward of 1 regardless of action. We make the proportion of (1) and (2) equal and set the budget so that all of (1) or (2) could be played (or some mix), but not more. Clearly, the optimal policy is to always play the Reliable agents since committing to the Greedy agents will eventually leave the planner only collecting reward from the Easy agents. However, the Greedy agents will look most attractive to VfNc since, without accounting for cost constraints, it will wrongly assume it can always pay the future cost to obtain increasingly larger reward. Hawkins and BLam, using their constraint-based reasoning, will instead commit to the Reliable agents. However, BLam will automatically detect the significant structure within the problem, such as the existence of Easy agents as well as a simple form for the V^i of Reliable agents, to build tight bounds on the Lagrange policy using only a small subset of agents in the coupled LP, leading to a significant speedup over Hawkins. The performance and runtimes of the algorithms tested on a population with 0.25, 0.25, 0.5 mix across Greedy, Reliable, and Easy agents with a budget of $0.25N$ and 30 actions (subsequently, 31 states) are shown in Fig. 4 and 5 confirming these insights. For BLam, all arms used test points $G^i=\{0\}$. Here, SampleLam gives good but variable performance in

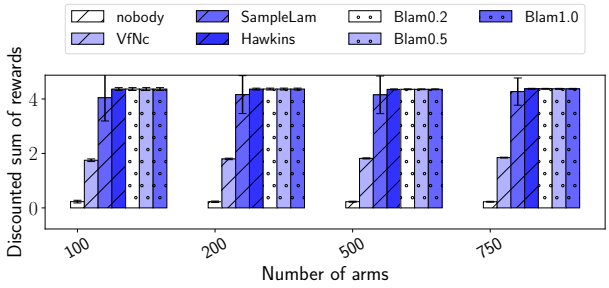


Figure 4: Ignoring future constraints leads to bad policies.

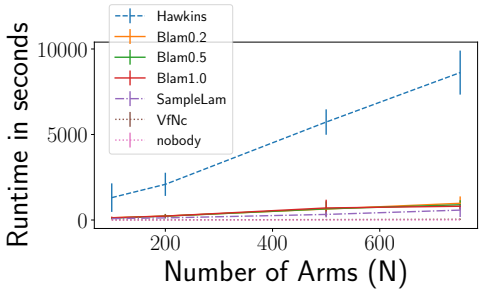


Figure 5: BLam and SampleLam scale better than Hawkins.

exchange for running approximately twice as fast as BLam.

Finally, we test our algorithms on a more rigorous simulation motivated by a real-world health behavior change task, namely, tuberculosis care in India. In this real-world setting, a single community health worker manages up to 200 patients throughout the course of their 6-month antibiotic regimen, monitoring and encouraging patients to take their daily medications (Killian et al. 2019). The health worker has a range of actions they can take on each patient aimed at improving their adherence, each with varying cost and effectiveness: call the patient (cheap), visit the patient in their home (moderate), escalate the patient (expensive). Because the worker's time is limited, the number and types of actions they can take each day across all patients are also limited.

We model this problem as follows. In the simulation, each patient state is a tuple of (adherence level, treatment phase, day of treatment). The first entry captures the patient's previous d days of adherence. The second entry is binary and captures the "phase" of treatment: the intensive phase which lasts for the first IPL rounds, and the continuation phase which lasts from round IPL to the end. During the intensive phase, patients tend to have better adherence and are more responsive to intervention. During the continuation phase, both effects tend to degrade and patients may drop out (i.e., adherence of 0). The final entry captures time and can take any of $IPL+2$ values. The first IPL values count days in the intensive phase and the next two are recurrent states that represent the continuation phase and the dropout state.

In one relevant dataset that captured daily treatment adherence of TB patients in India over the course of a year (Killian et al. 2019), patients followed four distinct modes: **(1) High adherence:** adhere daily regardless of health worker action. This makes up the majority of the data; **(2) Low adherence:** Very low adherence regardless of health worker action. **(3)**

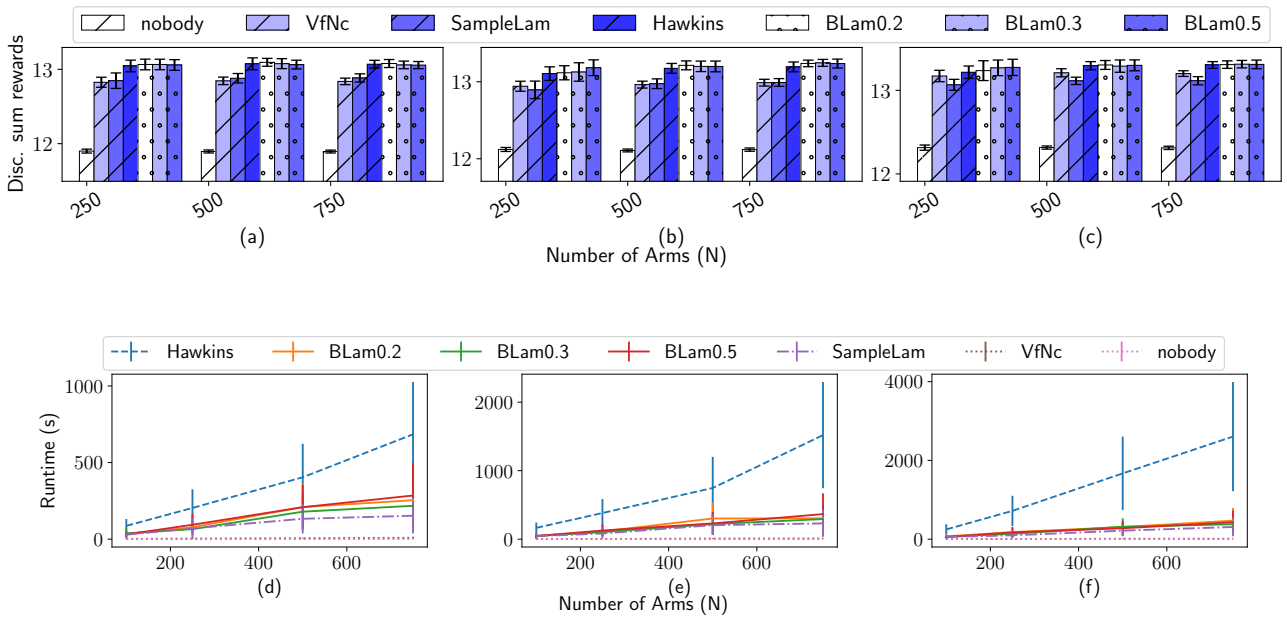


Figure 6: Rewards (top row) and runtimes (bottom row) on the health care dataset with budget $0.1N$. Columns represent $d=3,4,5$ adherence levels, respectively. At all values of ϵ, BLam significantly outperforms VfNc. Further, the Hawkins LP scales quadratically in the number of states on each arm, while BLam identifies problem structure that keep the underlying LPs small, making speedups more dramatic as the problem size increases.

Receptive patients: Irregular adherence but can benefit from intervention. On average, their adherence drops during the continuation phase, suggesting that interventions become less effective. **(4) Dropout patients:** Like receptive patients but have probability of dropping out during the continuation phase.

We implement each of these patient types in our simulation and include them in the following mix respectively: **0.64, 0.01, 0.175, 0.175**. This mix matches the number of High and Low adherence patients observed in the data, and splits the remaining portions evenly. At the start of simulation, each patient is in the maximum adherence state since in the real world, patients begin treatment in person. We run experiments with $d=3,4,5$ adherence levels and set $IPL=2d$. The health worker's action types are as follows: **(1) No action:** Take no action (c=0); **(2) Call:** Moderately increased probability that patient will increase adherence state by 1 (c=1). **(3) Visit:** Significant increased probability that patient will increase adherence state by 1 (c=2). **(4) Escalate:** Near-certain probability that patient will return to the maximum adherence state. If a patient is in the dropout state, there is a small probability they return to the continuation phase (c = B). Finally, rewards are defined as (adherence level)/d, so more rewards are received for patients at higher adherence levels.

We simulate this setting for many parameter combinations. For BLam we report results using test points $G^i=\{0,0.1,0.2,0.5\}$, though we found that, in general, most sets of 3 or 4 evenly spaced points worked well. Fig. 6 shows the performance and runtime for the dataset with budget of $0.1N$ for $d=3,4$, and 5 adherence levels. With such a small budget, the tradeoff between individual actions is important. Fig. 6 shows that all versions of BLam significantly outperform VfNc, and crucially, scale much better than Hawkins. In fact, as the number of states in the underlying problem grows, the speed ups become even more dramatic, ranging from a 2 times speedup with $d=3$ to a 5 times speedup with $d=5$. This is because the Hawkins LP scales quadratically in the number of states of each arm, while the BLam algorithms are able to identify problem structure that keep the underlying LPs small with its bounding techniques, making speedups more dramatic as the problem size increases.

In Appendix 12, we run experiments varying the budget between $0.1N$,$0.2N$,$0.5N$, with $d=4$ adherence levels. As the budget increases, resources are less constrained, so all methods tend to collapse to the same reward. However, BLam's adaptivity allows it to recognize when the problem is less constrained to automatically converge rapidly to the optimal solution.

These results demonstrate the exemplary ability for our approach to scale well without sacrificing performance on a dataset whose technical structural conditions have no been established a priori. That is, our algorithm can perform exceptionally with minimal tuning, while avoiding undertaking the considerable effort of deriving an index policy and the existence thereof.

8 Conclusion

Our work makes available multiple avenues for computing well-performing policies on new $(MA)^2RBs$ at scale. We demonstrate via a simulated medication adherence intervention task that our algorithms offer vast speedups and can be readily adapted to new, complex settings, a key requirement for planning health behavior change interventions. These advances make multi-action MARBs newly accessible, laying the groundwork for wider study of this important framework.

References

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

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

Bertsimas, D.; and Niño-Mora, J. 2000. Restless bandits, linear programming relaxations, and a primal-dual index heuristic. *Operations Research* 48(1): 80–90.

Bhattacharya, B. 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.

Cordwell, S. A. 2015. Python Markov Decision Process (MDP) Toolbox. PyPi, https://pypi.org/project/pymdptoolbox/.

Glazebrook, K. D.; Hodge, D. J.; and Kirkbride, C. 2011. General Notions of Indexability for Queueing Control and Asset Management. *The Annals of Applied Probability* 21(3): 876–907. ISSN 10505164. URL http://www.jstor.org/stable/23033358.

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

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

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

Jiang, S.; Song, Z.; Weinstein, O.; and Zhang, H. 2020. Faster dynamic matrix inverse for faster lps. *arXiv preprint arXiv:2004.07470* .

Killian, J. A.; Wilder, B.; Sharma, A.; Choudhary, V.; Dilkina, B.; and Tambe, M. 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.

Lee, E.; Lavieri, M. S.; and Volk, M. 2019. Optimal screening for hepatocellular carcinoma: A restless bandit model. *Manufacturing & Service Operations Management* 21(1): 198–212.

Mahajan, A.; and Teneketzis, D. 2008. Multi-armed bandit problems. In *Foundations and applications of sensor management*, 121–151. Springer.

Mate, A.; Killian, J. A.; Xu, H.; Perrault, A.; and Tambe, M. 2020. Collapsing Bandits and Their Application to Public Health Interventions. In *Advances in Neural Information Processing Systems*.

Meshram, R.; and Kaza, K. 2020. Simulation Based Algorithms for Markov Decision Processes and Multi-Action Restless Bandits. *arXiv preprint arXiv:2007.12933* .

Meuleau, N.; Hauskrecht, M.; Kim, K.-E.; Peshkin, L.; Kaelbling, L. P.; Dean, T. L.; and Boutilier, C. 1998. Solving very large weakly coupled Markov decision processes. In *AAAI/IAAI*, 165–172.

Modi, N.; Mary, P.; and Moy, C. 2019. Transfer restless multi-armed bandit policy for energy-efficient heterogeneous cellular network. *EURASIP Journal on Advances in Signal Processing* 2019(1): 46.

Nguyen, T. H.; Sinha, A.; Gholami, S.; Plumptre, A.; Joppa, L.; Tambe, M.; Driciru, M.; Wanyama, F.; Rwetsiba, A.; and Critchlow, R. 2013. Capture: A new predictive anti-poaching tool for wildlife protection .

Nino-Mora, J. 2001. Restless bandits, partial conservation laws and indexability. *Advances in Applied Probability* 76–98.

Papadimitriou, C. H.; and Tsitsiklis, J. N. 1999. The complexity of optimal queuing network control. *Math. Oper. Res.* 24(2): 293–305.

Puterman, M. L. 2014. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

Qian, Y.; Zhang, C.; Krishnamachari, B.; and Tambe, M. 2016. Restless poachers: Handling exploration-exploitation tradeoffs in security domains. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, 123–131.

Ruiz-Hernández, D.; Pinar-Pérez, J. M.; and Delgado-Gómez, D. 2020. Multi-machine preventive maintenance scheduling with imperfect interventions: A restless bandit approach. *Computers & Operations Research* 119: 104927.

Verloop, I. M.; et al. 2016. Asymptotically optimal priority policies for indexable and nonindexable restless bandits. *The Annals of Applied Probability* 26(4): 1947–1995.

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

Whittle, P. 1988. Restless bandits: Activity allocation in a changing world. *J. Appl. Probab.* 25(A): 287–298.

WHO; et al. 2018. *WHO guideline on health policy and system support to optimize community health worker programmes*. World Health Organization.

Zayas-Caban, G.; Jasin, S.; and Wang, G. 2019. An asymptotically optimal heuristic for general nonstationary finite-horizon restless multi-armed, multi-action bandits. *Advances in Applied Probability* 51(3): 745–772.

9 Constructing $\mathcal{U}$ and $\mathcal{L}$

Algorithm 4: BuildBounds

Data: D,G,N

$\mathcal{U}$ = [] ; // list of dicts for UB pieces

$\mathcal{L}$ = [] ; // list of dicts for LB pieces

for $i \in 0,...,N-1$ **do**

  /* Computing UBs, start from back */

  $j = |G[i]|-1$;

  $\mathcal{U}[i,j]['m'] = 0$;

  // last slope always 0 for LB

  $\mathcal{U}[i,j]['b'] = 0$;

  // last intercept is arbitrary

  $\lambda_{test} = G[i,j]$;

  /* set up the next line: y=mx+b */

  $y = \mathcal{U}[i,j]['m'] * \lambda_{test} + \mathcal{U}[i,j]['b']$;

  for $j \in |G[i]|-2,...,0$ **do**

    $\mathcal{U}[i,j]['m'] = D[i,j+1]$;

    /* b=y-mx */

    $\mathcal{U}[i,j]['b'] = y - \mathcal{U}[i,j]['m'] * \lambda_{test}$;

    $\lambda_{test} = G[i,j]$;

    /* set up the next line: y=mx+b */

    $y = \mathcal{U}[i,j]['m'] * \lambda_{test} + \mathcal{U}[i,j]['b']$;

  /* Computing LBs, start from front */

  $y = 0$;

  $\lambda_{test} = G[i,0]$;

  for $j \in 0,...,|G[i]|-1$ **do**

    $\mathcal{L}[i,j]['m'] = D[i,j]$;

    /* b=y-mx */

    $\mathcal{L}[i,j]['b'] = y - \mathcal{L}[i,j]['m'] * \lambda_{test}$;

    /* set up the next line: y=mx+b */

    $y = \mathcal{L}[i,j]['m'] * G[i,j+1] + \mathcal{L}[i,j]['b']$;

return $\mathcal{U},\mathcal{L}$

10 Proof of Thm 5.1

Theorem 10.1. *λ^i are $\frac{\sigma^2}{n}$-sub-Gaussian where* $\sigma^2 = \frac{1}{4}\left(\frac{r_{max}}{c_{min}(1-\beta)}\right)^2$

Proof. By Hoeffding's Lemma, if the values of a random variable can be bounded almost surely by a lower bound a and upper bound b, and its expected value is 0, then the random variable is $\frac{(b-a)^2}{4}$-sub-Gaussian. Note that shifting the expected value to be centered around the true mean introduces the factor n in the denominator, the number of samples used to estimate the mean. It remains to show that λ^i is bounded. The lower bound of λ^i is 0 due to the nature of the budget constraint, i.e., the budget constraint is $\leq$. To upper bound λ^i, we consider the Lagrange LP (i.e., Eq. 8 in the main text) for a single arm. We assume there always exists a 0 cost action (bandit assumption) which achieves at least 0 zero reward. Then if any single action charge λc_{min} is greater than any possible long term future reward $\frac{r_{max}}{1-\beta}$, the optimal policy will always choose the no-cost action, since the difference of the future reward and the charge will always be negative otherwise. Thus lambda cannot be increased further in the objective of the LP than $\frac{r_{max}}{c_{min}(1-\beta)}$, and is thus the upper bound on λ^i. □

11 Modified Knapsack

The modified knapsack LP used to compute policies in Section 6 of the main text is given below.

$$\max_X \sum_{i=0}^{N-1}\sum_{j=0}^{M-1} x_{i,j} Q^i(s^i, a_j, \lambda_{min}) \tag{18}$$

$$\text{s.t.} \sum_{i=0}^{N-1}\sum_{j=0}^{M-1} x_{i,j} c_j \leq B \tag{19}$$

$$\sum_{j=0}^{M-1} x_{i,j} = 1 \quad \forall i \in 0...N-1 \tag{20}$$

$$\tag{21}$$

$Q^i(s^i, a_j, \lambda_{min})$ is the action value function associated with arm i. Note that $Q^i(s^i, a_j, \lambda_{min})$ can be readily computed using the value functions returned by BLam and SampleLam via this algorithm:

Algorithm 5: Compute Action Value Function

Data: V,T,R,C,λ,β

Q = [] ; // hold the action value function

for $s \in \mathcal{S}$ **do**

  for $a \in \mathcal{A}$ **do**

    $Q[s,a] = R[s] - \lambda * C[a] + \beta * \sum_{s' \in \mathcal{S}} V[s'] * T[s,a,s']$

return Q

12 Additional Figures

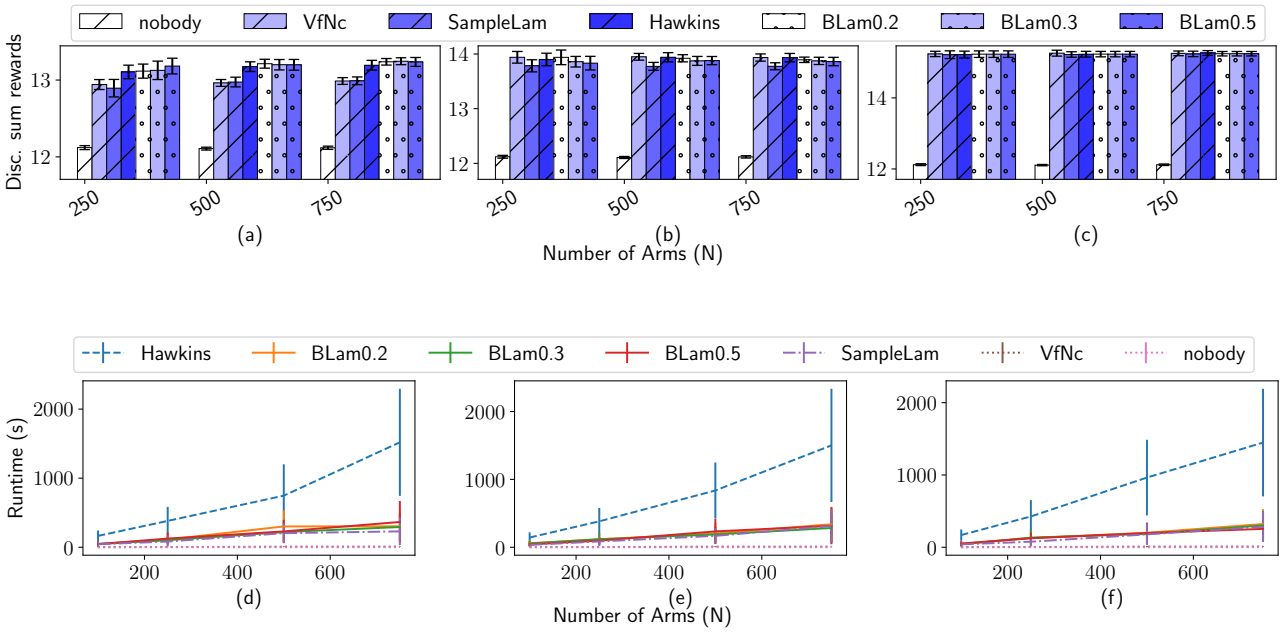


Figure 7: Rewards (top row) and runtimes (bottom row) on the health care dataset with $d = 4$ adherence levels. Columns represent a budget of $0.1N$,$0.2N$, and $0.5N$, respectively. At all values of ϵ, BLam significantly outperforms VfNc when the budget is small and the tradeoff between individual actions is important. BLam also scales much better than Hawkins, achieving a 5 times speedup in the 0.1 budget case and 6 times speedup in the 0.2 and 0.5 budget cases.