

# Beyond "To Act or Not to Act": Fast Lagrangian Approaches to General Multi-Action Restless Bandits

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

Andrew Perrault  
Harvard University  
Cambridge, MA, USA  
aperrault@g.harvard.edu

Milind Tambe  
Harvard University  
Cambridge, MA, USA  
tambe@g.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 many problems, they are restricted to binary actions, i.e., "to act or not to act". This renders them unable to capture critical complexities faced by planners in real domains, such as a system manager balancing maintenance, repair, and job scheduling, or a health worker deciding among treatments for a given patient. Limited previous work on Multi-action MARBs has only been specialized to 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 community health intervention task. Our approach achieves significant, up to ten-fold speedups over more general methods without sacrificing performance and is widely applicable across general Multi-action MARBs. Code is available at <https://github.com/killian-34/MAMARB-Lagrange-Policies>.

## KEYWORDS

Multi-action Restless Bandits; Lagrangian Relaxation; Bound Optimization; Markov Decision Process

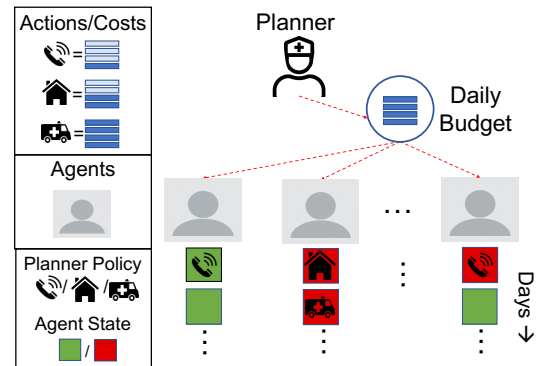
### ACM Reference Format:

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 *Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), Online, May 3-7, 2021, IFAA-MAS*, 9 pages.

## 1 INTRODUCTION

MARBs, the state-based generalization of classic Multi-Armed Bandits [30], have been studied extensively for solving a diverse set of problems including machine replacement [10, 29], sensing and wireless network scheduling [2, 3, 7, 22, 37], job scheduling [13, 34, 35]

*Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), U. Endriss, A. Nowé, F. Dignum, A. Lomuscio (eds.), May 3-7, 2021, Online. © 2021 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.*



**Figure 1: Multi-action MARB: one planner, many stateful agents, daily budget, many actions with varied costs/effects.**

anti-poaching patrol scheduling [28], and healthcare [5, 17, 19]. In classic Multi-Armed Bandits, 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. In MARBs, additional complexity is introduced in that 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 [26], but highly effective heuristics are known to exist [4, 33].

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 actions at their disposal with varying degrees of cost and effect. For example, a system manager may need to balance preventative maintenance, full repair, and job scheduling each with different costs and effects on throughput [6]. 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 [23]. In public health, a community health worker could have several options for intervening with a patient, such as calling, visiting in person, or escalating patients to a more intense treatment [25]. 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 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)<sup>2</sup>RBs). Previous work has considered extending the classical MARB notion of *indexability* and

corresponding *index policies* to  $(MA)^2RBs$  [9]. In both traditional 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 [14, 32]. However, both deriving index policies and verifying indexability is notoriously difficult, and largely requires special problem structure [9, 10]. Our goal is thus to develop fast, well-performing policies for a broader class of  $(MA)^2RBs$  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.<sup>1</sup>

The Weakly Coupled MDP (WCMDP) literature offers a method to compute the Lagrange policy for WCMDPs. Here, we recognize WCMDPs as a generalization of  $(MA)^2RBs$  and identify that this approach can be used to compute the Lagrange policy for  $(MA)^2RBs$ . 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  $(MA)^2RBs$  to create scalable algorithms for computing the Lagrange policy on any  $(MA)^2RB$  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 on various distributions.

**(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. Our exact algorithm, BLam, achieves  $\approx \sqrt{N}$  improvement and SampleLam achieves a factor of  $N$  improvement in the best case;

**(iv) Experimental evaluation:** We compare our algorithms to baselines on synthetic distributions with different underlying structure, including one motivated by a real-world public health challenge. 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  $(MA)^2RBs$  at scale, without the need for the user to first arduously derive a problem-specific index policy.

## 2 RELATED WORK

Previous work extends the traditional MARB notion of indexability to  $(MA)^2RBs$  [9, 14]. However, their analysis is restricted to a subclass of  $(MA)^2RBs$  with special monotonic structure, whereas we build algorithms for general  $(MA)^2RBs$ . “Superprocesses” are an alternative multi-action extension where a primary planner distributes a limited set of sub-planners who act on arms without constraint [18, 31, 36]. This structure does not generally apply to  $(MA)^2RBs$  since they do not constrain the number of agents that can be acted on each round. Very recent work [20] designs a Monte-Carlo rollout approach for estimating traditional 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. [21] derive methods for handling “global” resource constraints over all rounds, whereas we address round-by-round constraints. [12], 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  $(MA)^2RBs$  to greatly speed up the computation of the Lagrange bound compared to [12]. [1] 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. [11] develop a Lagrange approach for solving WCMDPs with MDPs that grow exponentially in problem parameters, restricting them to approximation techniques. In contrast, we develop a method that exactly computes the Lagrange policy.

Finally, our work is related to a large body of work developing Lagrangian methods for solving traditional MARBs [4, 10, 24, 33]. We generalize these settings to allow for multiple actions. Moreover, the methods we develop here reduce in the binary action case to the widely-used, well-performing, Whittle index policy [10, 33].

## 3 PRELIMINARIES

A  $(MA)^2RB$  consists of a set of  $N$  arms, each associated with a Markov Decision Process (MDP) [27]. 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  $r : \mathcal{S} \rightarrow \mathbb{R}$ , 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  $\pi : \mathcal{S} \rightarrow \mathcal{A}$  maps states to actions. The long-term *discounted reward* starting from state  $s_0 = s$  is defined as

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

Each arm  $i$  in a  $(MA)^2RB$  is an MDP with an action set  $\mathcal{A}^i$  of size  $M^i$  and corresponding action cost vector  $C^i \in \mathbb{R}^{M^i}$ . 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  $C$  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

<sup>1</sup>Please see the online appendix for additional discussion [https://teamcore.seas.harvard.edu/files/teamcore/files/multi\\_action\\_bandits\\_appendix.pdf](https://teamcore.seas.harvard.edu/files/teamcore/files/multi_action_bandits_appendix.pdf)

that the sum cost of all actions do not exceed a budget  $B$ . Formally, the planner must choose a decision matrix  $X \in \{0, 1\}^{N \times M}$  with elements denoted  $x_{i,j}$  such that

$$\sum_{j=0}^{M-1} x_{i,j} = 1 \quad \forall i \in 0 \dots N-1 \quad \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x_{i,j} c_j \leq B \quad (2)$$

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  $\mathbf{s} = [s^0, s^1, \dots, s^{N-1}]$  represent the vector of all arm states. The planner's goal can be represented by the following constrained Bellman equation

$$J(\mathbf{s}) = \max_X \left\{ \sum_{i=0}^{N-1} r^i(s^i) + \beta E[J(\mathbf{s}') | \mathbf{s}, X] \mid \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x_{i,j} c_j \leq B \right\} \quad (3)$$

While Eq. 3 could be solved directly via value iteration,  $J(\mathbf{s}) \in \mathcal{S}^N$ , and the number of feasible actions over which to take the max for each  $J(\mathbf{s})$  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  $\lambda$  as follows:

$$J(\mathbf{s}, \lambda) = \max_X \left\{ \sum_{i=0}^{N-1} r^i(s^i) + \lambda(B - \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x_{i,j} c_j) + \beta E[J(\mathbf{s}') | \mathbf{s}, X] \right\} \quad (4)$$

The value functions then decouple as desired, giving:

$$J(\mathbf{s}, \lambda) = \frac{\lambda B}{1 - \beta} + \sum_{i=0}^{N-1} V^i(s^i, \lambda), \quad \text{where } \forall i, \quad (5)$$

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

See [1] for a complete proof. Notice that for a given value of  $\lambda$ , Eq. 5 can be solved using a fast method like value iteration to solve for the individual  $V^i$ s, where  $V^i$  and its corresponding actions are now in  $\mathcal{S}$  and  $\mathcal{A}$ , respectively. However, the choice of  $\lambda$  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  $\lambda \rightarrow \infty$ , 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  $\lambda$  we recast the problem as an LP, rewriting Eq. 5 by leveraging the known LP solution to the value function [27]:

$$\begin{aligned} J(\mathbf{s}, \lambda) &= \min_{V^i(s^i, \lambda), \lambda} \frac{\lambda B}{1 - \beta} + \sum_{i=0}^{N-1} \mu^i(s^i) V^i(s^i, \lambda) \\ \text{s.t. } V^i(s^i, \lambda) &\geq r^i(s^i) - \lambda c_j + \beta \sum_{s^{i'}} T(s^i, a_j^i, s^{i'}) V^i(s^{i'}, \lambda) \\ \forall i \in \{0, \dots, N-1\}, \quad \forall s^i \in \mathcal{S}, \quad \forall a_j \in \mathcal{A}, \quad \text{and } \lambda &\geq 0 \end{aligned} \quad (7)$$

Where  $\mu^i(s^i) = 1$  if  $s^i$  is the start state for arm  $i$  and is 0 otherwise. That we minimize over  $\lambda$  and that  $\lambda \geq 0$  is a classic Lagrangian

result, motivated by making  $J(\mathbf{s}, \lambda)$  a tight-as-possible upper bound on  $J(\mathbf{s})$ . Intuitively, and matching how problem-specific index policies have been derived in previous work [9, 33], we want to derive a policy from the  $V^i$ s that provide the tightest bound on  $J(\mathbf{s})$ . So that our algorithms can generally apply to any (MA)<sup>2</sup>RB, our approaches will solve Eq. 7 in its general form.

The above derivation was first given by [12] for WCMDPs, and as suggested therein, clearly one can directly solve Eq. 7 using any LP solver. However, Eq. 7 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  $O(n^{2+\frac{1}{18}})$ , where  $n$  is the number of variables [15], implying that directly solving Eq. 7 has computational complexity  $\approx O(N^2|\mathcal{S}|^2)$  (derived in section 4). The key to our approach will be separating the computation of the  $\lambda$  that minimizes Eq. 7, henceforth  $\lambda_{min}$ , and the corresponding  $V^i$ s that solve Eq. 7 in a way that provides vast speedups. Herein we derive exact and heuristic methods for computing  $\lambda_{min}$ , 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. 7, visualized in blue in Fig. 2. To exactly compute Eq. 7 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  $\lambda_{test}$  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. 7, are indeed convex decreasing functions of  $\lambda$  (Prop. 4.1), implying that Eq. 7 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,  $\lambda_{min}$  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  $\lambda$  at which the negative sum of slopes equals  $\frac{B}{1 - \beta}$  could only increase, giving an upper bound on  $\lambda_{min}$ . The converse also holds, i.e., replacing with upper bound convex functions gives a lower bound on  $\lambda_{min}$  (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  $\lambda_{min}$  rapidly. We then iteratively add back  $V^i(s^i, \lambda)$ s to the LP until the bounds on  $\lambda_{min}$  are with a pre-specified  $\epsilon$ . With minimal tuning, the test points can

**Algorithm 1:** BLamPrecompute

---

**Data:**  $T, R, C, N, G, \beta$

- 1  $\mathcal{D} = []$ ; // hold slopes at each arm test point
- 2  $\epsilon_0 = 1e-3$ ;
- 3 **for**  $i = 1, \dots, N$  **do**
- 4     **for**  $j = 1, \dots, |G[i]|$  **do**
- 5          $\lambda_{test} = G[i, j]$ ;
- 6          $R_{\lambda}, R_{\lambda+\epsilon_0} = R[i]$ ;
- 7         **for**  $x \in 1, \dots, |C|$  **do** subtract action costs
- 8              $R_{\lambda}[x] -= \lambda_{test} * C[x]$ ;
- 9              $R_{\lambda+\epsilon_0}[x] -= (\lambda_{test} + \epsilon_0) * C[x]$ ;
- 10          $\mathcal{D}[i, j] = (VI(T[i], R_{\lambda+\epsilon_0}, \beta) - VI(T[i], R_{\lambda}, \beta)) / \epsilon_0$
- 11  $\mathcal{U}, \mathcal{L} = \text{BuildBounds}(\mathcal{D})$ ;
- 12 **return**  $\mathcal{U}, \mathcal{L}$

---

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 the arms, passing in the MDP parameters of the arms, along with a list  $G$  of points  $\lambda_{test}$  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)<sup>2</sup>RB to compute  $\lambda_{min}$  for the current set of arm states  $\mathbf{s}$  (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, BLAML P, given in Eq.8 below, that produces upper and lower bounds on  $\lambda_{min}$  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  $\lambda_{min}$  are within  $\epsilon$ . BLAM terminates by running one final value iteration with the appropriate  $\lambda_{min}$ , the result of which solves Eq. 7 *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.

#### 4.1 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  $\lambda$ , and as  $\lambda \rightarrow \infty$ ,  $\frac{dV^i(s^i, \lambda)}{d\lambda} \rightarrow 0$

**PROOF.** This follows directly from Eq. 6, but can be shown via induction that since  $V^i(s^i, \lambda)$  is a max over piecewise linear convex functions of  $\lambda$ , it is also piecewise linear convex, and since  $\lambda c_j \geq 0$ , it must be weakly decreasing in  $\lambda$ . Furthermore, since  $c_0 = 0$  in (MA)<sup>2</sup>RBs, as  $\lambda \rightarrow \infty$ , the one time charge  $\lambda c_j$  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  $\lambda$ .  $\square$

Let  $\lambda_u$  ( $\lambda_\ell$ ) correspond to the  $\lambda$  which solves Eq. 7 when  $V^i(s^i, \lambda)$  are replaced in the objective by  $\mathcal{L}$  ( $\mathcal{U}$ ). Note that the lower bound

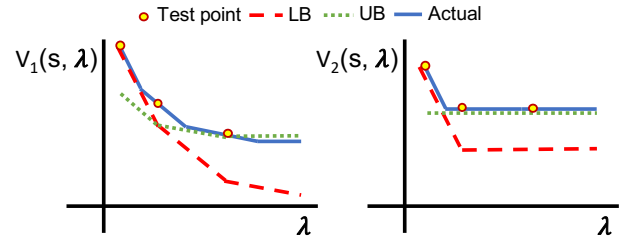
**Algorithm 2:** BLam

---

**Data:**  $T, R, C, N, B, \beta, s, G, \mathcal{U}, \mathcal{L}, \epsilon, kStep$

- 1 /\* Only need bounds for current arm states \*/
- 2 GetCoeffsForState( $\mathcal{U}, \mathcal{L}, s$ );
- 3 Sort( $\mathcal{U}, \mathcal{L}, T, R$ );
- 4  $st = \text{PickStart}(\mathcal{L}, \sqrt{N})$ ;
- 5 **for**  $k \in [st, st+kStep, \dots, N]$  **do**
- 6      $\lambda_u = \text{BLamLP}(T[:k], R[:k], B, C, \beta, s, \mathcal{L})$ ;
- 7      $\lambda_\ell = \text{BLamLP}(T[:k], R[:k], B, C, \beta, s, \mathcal{U})$ ;
- 8     **if**  $\lambda_u - \lambda_\ell \leq \epsilon$  **then** break;
- 9  $V(i, s) = []$  //  $Nx|S|$  array to hold value functions
- 10  $\lambda_{min} = (\lambda_u - \lambda_\ell) / 2$ ;
- 11 **for**  $i = 1, \dots, N$  **do**
- 12      $R_\lambda = R[i]$ ;
- 13     **for**  $x \in 1, \dots, |C|$  **do** subtract action costs
- 14          $R_\lambda[x] -= \lambda_{min} * C[x]$ ;
- 15      $V[i] = VI(T[i], R_\lambda, \beta)$ ;
- 16 **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.**

functions  $\mathcal{L}$  will be used to derive *upper bounds* on the value of  $\lambda_{min}$  and vice versa.

Next, we give a helpful intermediate result.

**PROPOSITION 4.2.** *The optimal solution to Eq. 7 will be found at the value of  $\lambda$  in which the negative sums of the slopes of  $V^i(s^i, \lambda)$  w.r.t.  $\lambda$  become less than or equal to  $\frac{B}{1-\beta}$ .*

**PROOF.** Assume  $\lambda^*$  corresponds to an optimal solution to Eq. 7 and the negative sums of the slopes of convex decreasing  $V^i(s^i, \lambda)$  are greater than  $\frac{B}{1-\beta}$ . Then  $\lambda^*$  can be increased by  $\epsilon$  and the objective value would decrease, i.e.,  $J(s, \lambda^* + \epsilon) < J(s, \lambda^*)$  giving a contradiction.  $\square$

We now can prove our main result:

**THEOREM 4.3.**  $\lambda_\ell \leq \lambda_{min} \leq \lambda_u$

**PROOF.** The proof is best seen by considering  $\lambda_{min}$  which solves  $J(s, \lambda)$ , i.e., Eq. 7. We start with  $\lambda_{min} \leq \lambda_u$ : Let  $\mathcal{V}$  denote the set of  $V^i(s^i, \lambda)$  in the objective of Eq. 7. 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  $J^{\lambda_u}(s, \lambda)$  and name its optimal solution  $\lambda_u$ . By definition, at all values of  $\lambda$ , the slope of

$V^i(s^i, \lambda)$  is greater than the slope of  $\mathcal{L}^b$ . Thus, at  $\lambda_{min}$ , the negative sums of the slopes of  $V^i \in \mathcal{V} \setminus \mathcal{V}^b$  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  $J^{\lambda_u}(s, \lambda) \leq J(s, \lambda)$ , and respectively  $\lambda_u \geq \lambda_{min}$ .  $\lambda_{min} \geq \lambda_l$ : The proof follows similarly.  $\square$

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*  $\lambda_{test}$  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  $V^i$ s. 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 1. Let  $\mathcal{U}^i(G_k^i, m)$  and  $\mathcal{U}^i(G_k^i, b)$  be the slopes and intercepts, respectively, for each piece  $k$  of the upper bounding function  $\mathcal{U}^i$  for arm  $i$ . Define  $\mathcal{L}^i(G_k^i, *)$  similarly.

Now, we can compute  $\lambda_u$  and  $\lambda_l$ . To start, we choose  $K$  arms to include in Eq. 7 in their  $V^i(s^i, \lambda)$  form, while the other  $N - K$  arms will be replaced by their bounded counterparts. To compute  $\lambda_u$ , we replace the  $N - K$  arms with  $\mathcal{L}$  to get the following LP:

$$\begin{aligned} J^{\lambda_u}(s, \lambda) &= \min_{V^i, \lambda, z^j} \frac{\lambda B}{1 - \beta} + \sum_{i=0}^K \mu^i(s^i) V^i(s^i, \lambda) + \sum_j^{N-K} z^j \\ \text{s.t. } V^i(s^i, \lambda) &\geq r^i(s^i) - \lambda c_j + \beta \sum_{s^{i'}} T(s^i, a_j^i, s^{i'}) V^i(s^{i'}, \lambda) \\ &\quad \forall i \in \{0, \dots, K\}, \quad \forall s^i \in \mathcal{S}, \quad \forall a_j \in \mathcal{A} \quad (8) \\ z^j &\geq \mathcal{L}^j(G_k^j, m) * \lambda + \mathcal{L}^j(G_k^j, b) \\ &\quad \forall k \in \{0, \dots, |G^j|\}, \quad \forall j \in \{0, \dots, N - K\} \\ \lambda &\geq 0 \end{aligned}$$

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  $\lambda_l$  we construct a similar LP using  $\mathcal{U}^i(G_k^i, *)$ .

One important choice is in selecting the first  $K$  arms. Intuitively, the best  $V^i(s^i, \lambda)$  to include in Eq. 8 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  $\lambda$  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  $\lambda_u$  and  $\lambda_l$  are computed once, we iterate to include  $K_{step}$  more arms in the LP such that  $K += K_{step}$  then repeat until the algorithm converges to within a difference  $\epsilon$ . 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  $\lambda_{min}$  is determined, we use value iteration to rapidly solve Eq. 7, the result of which we will use to derive feasible policies in Section 6.

## 4.2 BLam: Computational Complexity

In BLAMPRECOMPUTE, BLam computes  $\mathcal{U}^i(G_k^i, *)$  and  $\mathcal{L}^i(G_k^i, *)$  for all  $V^i(s^i, \lambda)$ , which requires two runs of value iteration for each arm for each test point  $G_k^i$ . Assuming all arms use the same number of test points, states and actions, this scales as  $O(NG^i VI(|\mathcal{S}|, |\mathcal{A}|))$  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 [27]. 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. 8 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 [15]. In the best case, BLam would provide tight bounds on  $\lambda_{min}$  after just one iteration. So setting  $K = \sqrt{N}$  and assuming  $G^i \ll N$ , the per-round complexity is

$$\Omega(\sqrt{N}|S|^2|A| + N|S|^2 + N^{\frac{3}{2}}|S| + N^2) \quad (9)$$

Where the first term is the LP setup time to add constraints (which 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

$$O(N|S|^2|A| + N^2|S|^2) \quad (10)$$

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  $\lambda_{min}$ . 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

$$O(N|S|^2|A| + N^{\frac{5}{2}}|S|^2) \quad (11)$$

Which, handily, is only  $\sqrt{N}$  worse than the naive approach. However, we will 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

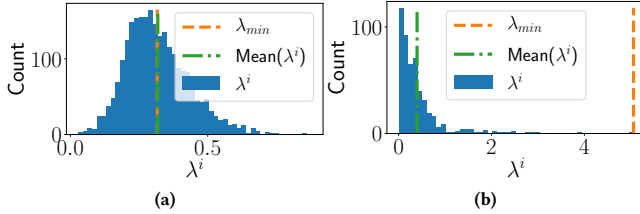
**Algorithm 3: SampleLam**


---

**Data:**  $T, R, C, N, B, \beta, r_{max}, c_{min}$

- 1  $N_{samples} = \frac{\log(N)r_{max}}{c_{min}}$ ;
- 2  $inds = \text{RandomChoice}([1, \dots, N], N_{samples})$ ;
- 3  $T, R = T[inds], R[inds]$ ;
- 4  $\lambda_{list}^i = []$ ;
- 5 **for**  $i = 1, \dots, N_{samples}$  **do**
- 6      $\lambda^i = \text{QuickLP}(T[i], R[i], \frac{B}{N}, C, \beta)$ ;
- 7      $\lambda_{list}^i.append(\lambda^i)$
- 8  $V(i, s) = []$  //  $N \times |S|$  array to hold value functions
- 9  $\lambda_{min} = \text{Mean}(\lambda_{list}^i)$ ;
- 10 **for**  $i = 1, \dots, N$  **do**
- 11      $R_\lambda = R[i]$ ;
- 12     **for**  $x \in 1, \dots, |C|$  **do** subtract action costs
- 13          $R_\lambda[x] -= \lambda_{min} * C[x]$ ;
- 14      $V[i] = \text{VI}(T[i], R_\lambda, \beta)$ ;
- 15 **return**  $V$

---



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

quickly computing a heuristic estimate of  $\lambda_{min}$  based on sampling. The approach is grounded in the mathematical interpretation of  $\lambda_{min}$ , i.e., that  $\lambda_{min}$  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  $\lambda_{min}$ .

The algorithm is given in Alg. 3. It first chooses  $K$  processes at random to run through `QuickLP`, which solves Eq. 7 with a single arm and modified budget  $\frac{B}{N}$  giving a value  $\lambda^i$  that estimates the value of playing that arm. It then creates an estimate of  $\lambda_{min}$  by taking the sample mean of  $\lambda^i$ . Finally, it uses this estimate plus value iteration to solve for Eq. 7, which again we will use to derive feasible policies in section 6.

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

## 5.1 SampleLam: Concentration Bound and Complexity

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

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

The proof involves showing  $0 \leq \lambda^i \leq \frac{r_{max}}{c_{min}(1-\beta)}$  and is given in Appendix 2. We can now use sub-Gaussianity to derive a concentration bound relating the number of samples to a confidence parameter  $1 - \delta$  on the estimate of the sample mean of  $\lambda^i$ .

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

$$n \geq \frac{1}{2\epsilon^2} \left( \frac{r_{max}}{c_{min}(1-\beta)} \right)^2 \log \left( \frac{1}{\delta} \right) \quad (12)$$

The proof, given in Appendix 2, uses the Hoeffding bound and 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  $\lambda_{min}$  is not guaranteed to be close to the sample mean of  $\lambda^i$  in general. An adversarial case is shown in Fig. 3b in which SampleLam would compute an arbitrarily bad estimate for  $\lambda_{min}$ . Setting  $K = \log(N) \frac{r_{max}}{c_{min}}$ , the best and worst case complexity for SampleLam is:

$$O \left( \log(N) \frac{r_{max}}{c_{min}} |S|^2 |A| + N * \text{VI}(|S|, |\mathcal{A}|) \right) \quad (13)$$

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

## 6 COMPUTING A POLICY

Finally, once  $\lambda_{min}$  is finalized, and the resulting value functions from Eq. 7 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 3, 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. 7 with  $\lambda = 0$ , effectively ignoring all future constraints, then follows Section 6; **Hawkins**: Solves Eq. 7 directly

using an LP solver, then follows Section 6. We also include several versions of BLam using various stopping criterion  $\epsilon$ , noted in each plot as BLam( $\epsilon$ ). Larger  $\epsilon$  will lead to faster running times but looser bounds on  $\lambda_{min}$ , 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 [8].

First, we explore an example distribution where VfNc and Hawkins 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 exchange for running approximately twice as fast as BLam.

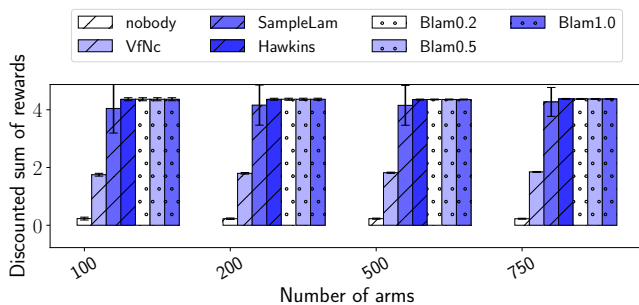


Figure 4: Ignoring future constraints leads to bad policies.

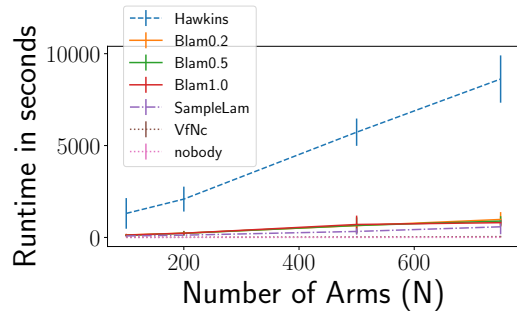


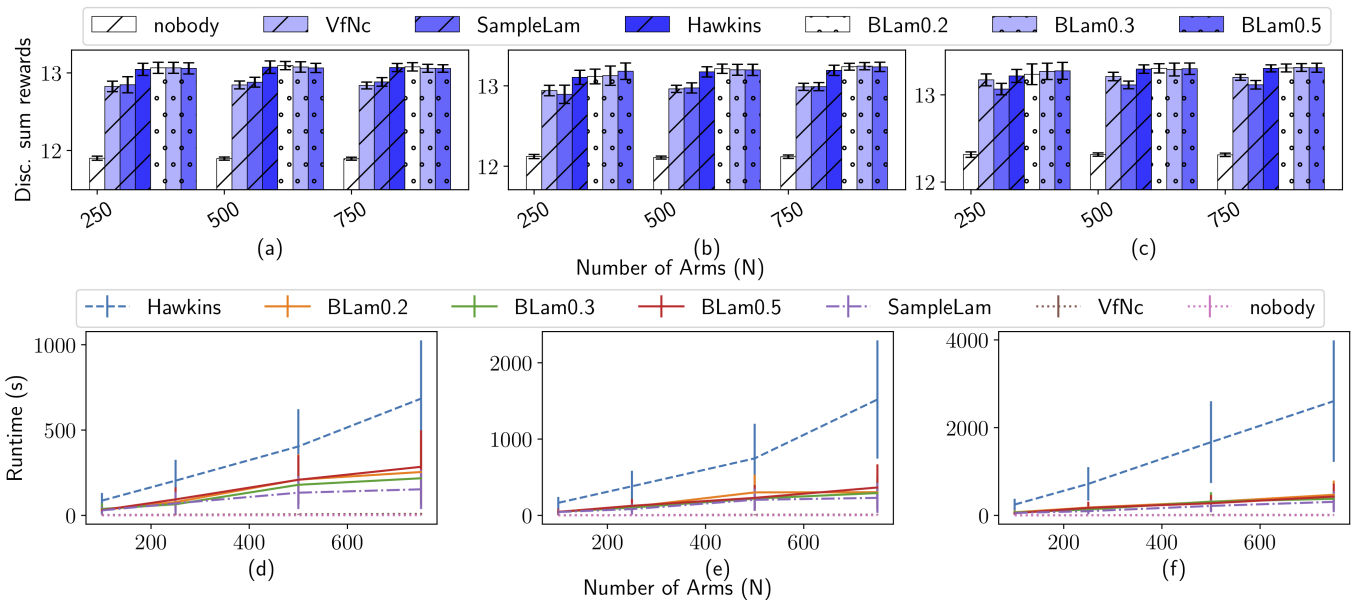
Figure 5: BLam and SampleLam scale better than Hawkins.

Finally, we test our algorithms on a more rigorous simulation motivated by a real-world public health care challenge, 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. 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 (semi-expensive), escalate the patient (very expensive). Because the worker’s time and resources are 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 [16], 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) 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



**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  $\epsilon$ , 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.**

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. In Fig. 6 we see that all versions of BLam significantly outperform VfNc. Crucially, all versions of BLam also 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 4, 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, again, BLam’s adaptivity allows

it to recognize when the problem is less constrained to automatically converge even more quickly 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 not 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)<sup>2</sup>RBs at scale. We demonstrate that our algorithms offer vast speedups and can be readily adapted to new problems without the need for the user to first arduously derive a problem-specific index policy, as was previously the case. These advances make multi-action MARBs newly accessible, laying the groundwork for wider study of this important framework.

## 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.P. was supported by the Harvard Center for Research on Computation and Society.

## REFERENCES

- [1] Daniel Adelman and Adam J Mersereau. 2008. Relaxations of weakly coupled stochastic dynamic programs. *Operations Research* 56, 3 (2008), 712–727.
- [2] Sahand Haji Ali Ahmad and Mingyan Liu. 2009. Multi-channel opportunistic access: A case of restless bandits with multiple plays. In *2009 47th Annual Allerton*



- Conference on Communication, Control, and Computing (Allerton)*. IEEE, 1361–1368.
- [3] Saeed Bagheri and Anna Scaglione. 2015. The restless multi-armed bandit formulation of the cognitive compressive sensing problem. *IEEE Transactions on Signal Processing* 63, 5 (2015), 1183–1198.
  - [4] Dimitris Bertsimas and José Niño-Mora. 2000. Restless bandits, linear programming relaxations, and a primal-dual index heuristic. *Operations Research* 48, 1 (2000), 80–90.
  - [5] 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.
  - [6] C Richard Cassady and Erhan Kutanoglu. 2005. Integrating preventive maintenance planning and production scheduling for a single machine. *IEEE Transactions on reliability* 54, 2 (2005), 304–309.
  - [7] Kobi Cohen, Qing Zhao, and Anna Scaglione. 2014. Restless multi-armed bandits under time-varying activation constraints for dynamic spectrum access. In *2014 48th Asilomar Conference on Signals, Systems and Computers*. IEEE, 1575–1578.
  - [8] Steven A.W. Cordwell. 2015. Python Markov Decision Process (MDP) Toolbox. PyPi, <https://pypi.org/project/pymdptoolbox/>.
  - [9] Kevin D. Glazebrook, David J. Hodge, and Chris Kirkbride. 2011. GENERAL NOTIONS OF INDEXABILITY FOR QUEUEING CONTROL AND ASSET MANAGEMENT. *The Annals of Applied Probability* 21, 3 (2011), 876–907. <http://www.jstor.org/stable/23033358>
  - [10] K. D. Glazebrook, D. Ruiz-Hernandez, and C. Kirkbride. 2006. Some indexable families of restless bandit problems. *Adv. Appl. Probab.* 38, 3 (2006), 643–672.
  - [11] Yasin Gocgun and Archis Ghate. 2012. Lagrangian relaxation and constraint generation for allocation and advanced scheduling. *Computers & Operations Research* 39, 10 (2012), 2323–2336.
  - [12] Jeffrey Thomas Hawkins. 2003. *A Lagrangian decomposition approach to weakly coupled dynamic optimization problems and its applications*. Ph.D. Dissertation. Massachusetts Institute of Technology.
  - [13] Ting He, Shiyao Chen, Hyoil Kim, Lang Tong, and Kang-Won Lee. 2012. Scheduling parallel tasks onto opportunistically available cloud resources. In *2012 IEEE Fifth International Conference on Cloud Computing*. IEEE, 180–187.
  - [14] David J Hodge and Kevin D Glazebrook. 2015. On the asymptotic optimality of greedy index heuristics for multi-action restless bandits. *Advances in Applied Probability* 47, 3 (2015), 652–667.
  - [15] Shunhua Jiang, Zhao Song, Omri Weinstein, and Hengjie Zhang. 2020. Faster dynamic matrix inverse for faster lps. *arXiv preprint arXiv:2004.07470* (2020).
  - [16] 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.
  - [17] Elliot Lee, Mariel S Lavieri, and Michael Volk. 2019. Optimal screening for hepatocellular carcinoma: A restless bandit model. *Manufacturing & Service Operations Management* 21, 1 (2019), 198–212.
  - [18] Aditya Mahajan and Demosthenis Teneketzis. 2008. Multi-armed bandit problems. In *Foundations and applications of sensor management*. Springer, 121–151.
  - [19] Aditya Mate, Jackson A Killian, Haifeng Xu, Andrew Perrault, and Milind Tambe. 2020. Collapsing Bandits and Their Application to Public Health Interventions. In *Advances in Neural Information Processing Systems*.
  - [20] Rahul Meshram and Kesav Kaza. 2020. Simulation Based Algorithms for Markov Decision Processes and Multi-Action Restless Bandits. *arXiv preprint arXiv:2007.12933* (2020).
  - [21] Nicolas Meuleau, Milos Hauskrecht, Kee-Eung Kim, Leonid Peshkin, Leslie Pack Kaelbling, Thomas L Dean, and Craig Boutilier. 1998. Solving very large weakly coupled Markov decision processes. In *AAAI/IAAI*. 165–172.
  - [22] Navikkumar Modi, Philippe Mary, and Christophe Moy. 2019. Transfer restless multi-armed bandit policy for energy-efficient heterogeneous cellular network. *EURASIP Journal on Advances in Signal Processing* 2019, 1 (2019), 46.
  - [23] Thanh H Nguyen, Arunesh Sinha, Shahrzad Gholami, Andrew Plumptre, Lucas Joppa, Milind Tambe, Margaret Driciru, Fred Wanyama, Aggrey Rwetsiba, and Rob Critchlow. 2013. Capture: A new predictive anti-poaching tool for wildlife protection. (2013).
  - [24] Jose Nino-Mora. 2001. Restless bandits, partial conservation laws and indexability. *Advances in Applied Probability* (2001), 76–98.
  - [25] World Health Organization et al. 2018. *WHO guideline on health policy and system support to optimize community health worker programmes*. World Health Organization.
  - [26] C. H. Papadimitriou and J. N. Tsitsiklis. 1999. The complexity of optimal queueing network control. *Math. Oper. Res.* 24, 2 (1999), 293–305.
  - [27] M. L. Puterman. 2014. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons.
  - [28] 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.
  - [29] Diego Ruiz-Hernández, Jesús M Pinar-Pérez, and David Delgado-Gómez. 2020. Multi-machine preventive maintenance scheduling with imperfect interventions: A restless bandit approach. *Computers & Operations Research* 119 (2020), 104927.
  - [30] William R Thompson. 1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika* 25, 3/4 (1933), 285–294.
  - [31] Ina Maria Verloop et al. 2016. Asymptotically optimal priority policies for indexable and nonindexable restless bandits. *The Annals of Applied Probability* 26, 4 (2016), 1947–1995.
  - [32] R. R. Weber and G. Weiss. 1990. On an index policy for restless bandits. *J. Appl. Probab.* 27, 3 (1990), 637–648.
  - [33] P. Whittle. 1988. Restless bandits: Activity allocation in a changing world. *J. Appl. Probab.* 25, A (1988), 287–298.
  - [34] Fuqian Yang, Zhaowei Zhu, Shangshu Zhao, Yang Yang, and Xiliang Luo. 2018. Optimal task offloading in fog-enabled networks via index policies. In *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE, 688–692.
  - [35] Zhe Yu, Yunjian Xu, and Lang Tong. 2018. Deadline scheduling as restless bandits. *IEEE Trans. Automat. Control* 63, 8 (2018), 2343–2358.
  - [36] Gabriel Zayas-Caban, Stefanus Jasin, and Guihua Wang. 2019. An asymptotically optimal heuristic for general nonstationary finite-horizon restless multi-armed, multi-action bandits. *Advances in Applied Probability* 51, 3 (2019), 745–772.
  - [37] Qing Zhao, Bhaskar Krishnamachari, and Keqin Liu. 2008. On myopic sensing for multi-channel opportunistic access: structure, optimality, and performance. *IEEE Transactions on Wireless Communications* 7, 12 (2008), 5431–5440.