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

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

Abstract

We present a new algorithm and theoretical results for solving Multi-action Multi-armed Restless Bandits, an important but insufficiently studied generalization of traditional Multi-armed Restless Bandits (MARBs). Multi-action MARBs are capable of handling critical problem complexities often present in AI4SG domains like anti-poaching and healthcare, but that traditional MARBs fail to capture. Limited previous work on Multi-action MARBs has only been specialized to sub-problems. Here we derive BLam, an algorithm for general Multi-action MARBs using Lagrangian relaxation techniques and convexity to quickly converge to good policies via bound optimization. We also provide experimental results comparing BLam to baselines on a simulated distributions motivated by a real-world community health intervention task, achieving up to five-times 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 *et al.*, 2006; Ruiz-Hernández *et al.*, 2020], anti-poaching patrol scheduling [Qian *et al.*, 2016], and healthcare [Lee *et al.*, 2019; Mate *et al.*, 2020]. The 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. However, 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), making the problem PSPACE-hard [Papadimitriou and Tsitsiklis, 1999].

Despite their difficulty, 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 degrees of cost and effect. E.g., 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 could have several options for intervening with a patient, such as calling, visiting in person, or escalating patients to a more intense treatment [WHO and others, 2018]. Traditional MARBs cannot model these complexities, restricting planners to a world where their only choices are to, e.g., call or not call. Rather, the planner must optimize over all the tools in their toolbelt simultaneously (Fig. 1).

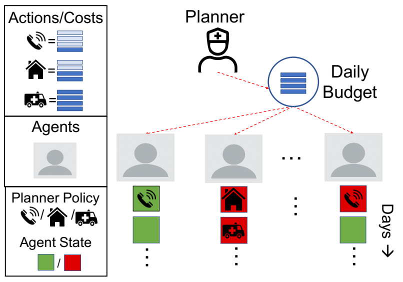

Figure 1: Multi-action MARB: one planner, many stateful agents, daily budget, many actions with varied costs/effects.

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 classical MARB notion of *indexability* and corresponding *index policies* to (MA)^{2}RBs [Glazebrook *et al.*, 2011]. 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 [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 *et al.*, 2006; Glazebrook *et al.*, 2011]. 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. Our contributions:

(i) Bound optimization algorithm: We develop BLam, an iterative bound optimization method for computing the Lagrange policy. BLam uses 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.

(ii) Experimental evaluation: We compare our algorithms to baselines on a synthetic distribution motivated by a real-world public health challenge. Our algorithms scale up to five 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, 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)^{2}RBs [Glazebrook *et al.*, 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 [Zayas-Caban *et al.*, 2019; Verloop and others, 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. Also related are Weakly Coupled MDPs (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]. Finally, our work is related to a large body of work developing Lagrangian methods for solving traditional MARBs [Whittle, 1988; Glazebrook *et al.*, 2006; Nino-Mora, 2001; Bertsimas and Niño-Mora, 2000]. 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 [Whittle, 1988; Glazebrook *et al.*, 2006].

3 Preliminaries

A (MA)^{2}RB consists of N arms, each associated with an MDP $\{\mathcal{S},\mathcal{A},r,T,\beta\}$ which consists of a set of states $\mathcal{S}$, actions $\mathcal{A}$, a bounded reward function $r:\mathcal{S}\rightarrow\mathbb{R}$, transition function T, where $T(s,a,s')$ is 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. Each arm also has an action cost vector $\boldsymbol{C}\in\mathbb{R}^{|\mathcal{A}|}$. W.l.o.g., we assume that the elements c_j of $\boldsymbol{C}$ are ordered ascending. Also, we set $c_0=0$ so an arm can be "not played" at no cost. 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. Let $\boldsymbol{s}=[s^0,s^1,...,s^{N-1}]$ represent the vector of all arm states and let $\boldsymbol{X}\in\{0,1\}^{N\times|\mathcal{A}|}$ with elements denoted $x_{i,j}$ be the action decision matrix. The planner's goal can be represented by the following constrained Bellman equation

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

While Eq. 1 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. This gives a relaxed but tractable value function $J(\boldsymbol{s},\lambda)$ which upper bounds $J(\boldsymbol{s})$ and can be solved with the LP (proof in [Hawkins, 2003]):

$$\begin{aligned}
&J(\boldsymbol{s},\lambda)=\min_{V^i(s^i,\lambda),\lambda}\frac{\lambda B}{1-\beta}+\sum_{i=0}^{N-1}\mu^i(s^i)V^i(s^i,\lambda)\\
&\text{s.t. } V^i(s^i,\lambda)\geq r^i(s^i)-\lambda c_j+\beta\sum_{s^{i\prime}}T(s^i,a^i_j,s^{i\prime})V^i(s^{i\prime},\lambda)\\
&\forall i\in\{0,...,N-1\},\ \forall s^i\in\mathcal{S},\ \forall a_j\in\mathcal{A},\text{ and }\lambda\geq 0
\end{aligned} \tag{2}$$

Where $\mu^i(s^i)=1$ if s^i is the start state for arm i and is 0 otherwise. Intuitively, we want to derive a policy from the V^is that provide the tightest bound on $J(\boldsymbol{s})$. Clearly one can directly solve Eq. 2 using any LP solver, but this scales poorly. The key to our approach will be separating the computation of the λ that minimizes Eq. 2, henceforth λ_{min}, and the corresponding V^is that solve Eq. 2 in a way that provides vast speedups.

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. 2, visualized in blue in Fig. 2. To exactly compute Eq. 2 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. 2, are indeed convex decreasing functions of λ (Prop. 4.1), implying that Eq. 2 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 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. 3 (appendix), 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. 3 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. 1, 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. 1 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.3 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. 2 *without constructing or solving the full LP, leading to vast speed ups*. The resulting value functions will be used to construct a final policy via a modified knapsack (Appendix 10).

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

4.1 BLam: Derivation

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

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

This follows directly from Eq. 2. Let λ_u (λ_ℓ) correspond to the λ which solves Eq. 2 when $V^i(s^i,\lambda)$ are replaced in the objective

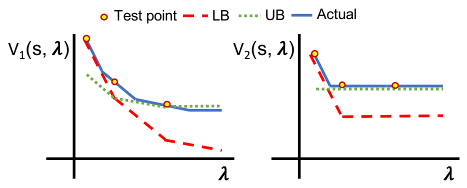

Figure 2: Constructing bounds on the slope of $V^i(s^i,\lambda)$ for two different arms with three test points. Note: bounds are with respect to the *slope*, not the value of the function.

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 (proof in appendix).

Proposition 4.2. *The optimal solution to Eq. 2 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 $\frac{B}{1-\beta}$.*

We now give our main result (proof in appendix):

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

We can build $\mathcal{U}$ and $\mathcal{L}$ by testing for the slope of $V^i(s^i,\lambda)$ at *test points* 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. At minimum, G^i must include $\lambda_{test}=0$ since proposition 4.1 implies that the minimum slope of any V^i occurs at $\lambda = 0$. After computing slopes at test points, $\mathcal{U}$ and $\mathcal{L}$ are constructed via standard linear equations (Alg 2 in the Appendix). 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. 2 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:

$$
\begin{aligned}
&J^{\lambda_u}(\boldsymbol{s},\lambda) = \min_{V^i,\lambda,z^j} \frac{\lambda B}{1-\beta} + \sum_{i=0}^{K} \mu^i(s^i)V^i(s^i,\lambda) + \sum_{j}^{N-K} z^j \\
&\text{s.t. } V^i(s^i,\lambda) \ge r^i(s^i) - \lambda c_j + \beta \sum_{s^{i\prime}} T(s^i,a^i_j,s^{i\prime})V^i(s^{i\prime},\lambda) \\
&\qquad\qquad \forall i \in \{0,...,K\},\ \forall s^i \in \mathcal{S},\ \forall a_j \in \mathcal{A} \qquad (3)\\
&z^j \ge \mathcal{L}^j(G^j_k,m)*\lambda + \mathcal{L}^j(G^j_k,b) \\
&\qquad\qquad \forall k \in \{0,...,|G^j|\},\ \forall j \in \{0,...,N-K\} \\
&\lambda \ge 0
\end{aligned}
$$

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 $\mathcal{U}^i(G^i_k,*)$. To choose the initial K arms, we sort arms by the slope of their last line segment, then set K such that the negative sum of slopes of the last line segments of the remaining $N-K$ arms is less than $B/(1-\beta)$ to ensure the LP is feasible. More details are given in appendix 9. Once λ_{min} is determined, we use value iteration to rapidly solve Eq. 2, the result of which we use to derive feasible policies via a modified knapsack (Appendix 10).

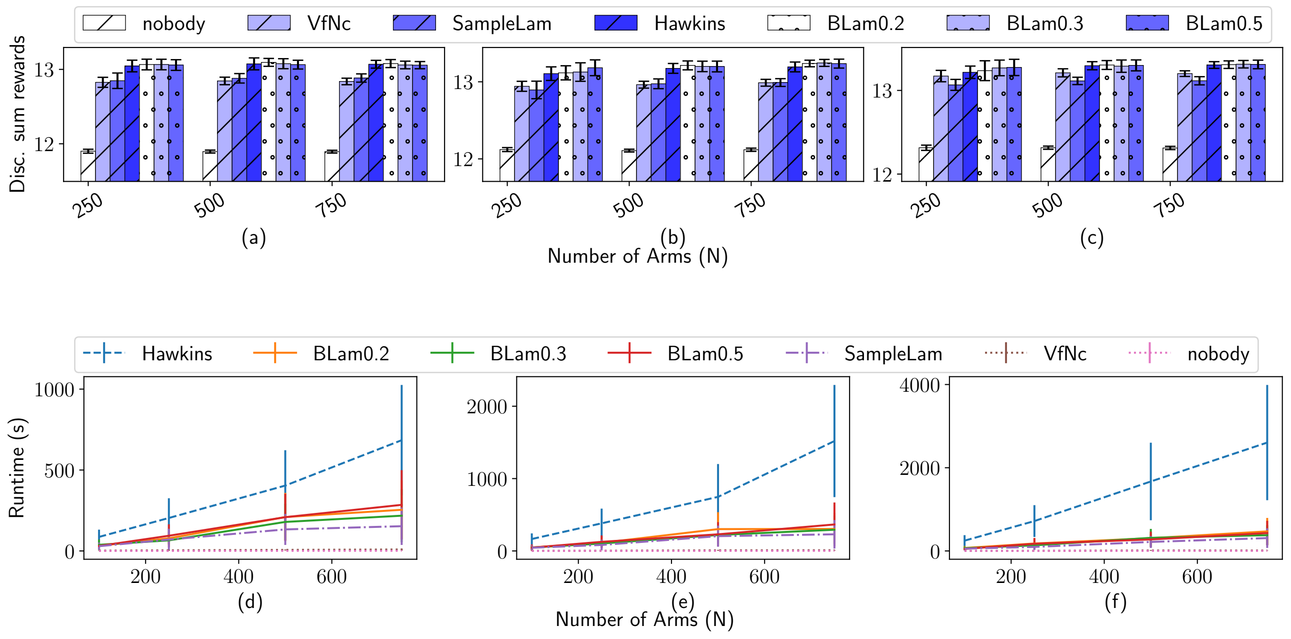

Figure 3: Rewards (top row) and runtimes (bottom row) on the health care dataset with budget $0.1N$. Columns represent $d = 3,4,5$ adherence levels, respectively. At all values of ϵ, BLam significantly outperforms VfNc. Further, the Hawkins LP scales quadratically in the number of states on each arm,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

5 Experiments

We compare our methods against the following baselines: **Nobody**: Take a_0 which has no cost on every arm; **VfNc**: Solves Eq. 2 with $\lambda = 0$, effectively ignoring all future constraints, then follows Section 10; **Hawkins**: Solves Eq. 2 directly using an LP solver, then follows Section 10. We also include several versions of BLam using various stopping criterion ϵ (i.e., BLam$\{\epsilon\}$ in plots). 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 test our algorithm on a simulation motivated by 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 via calls, home visits or escalations, subject to a daily time budget. We model patient adherence via a state tuple: (adherence level, treatment phase, day of treatment), which captures the patient's previous d days of adherence, the "phase" of treatment (patients adhere less in later phases), and time. A dataset of daily TB adherence in Mumbai [Killian *et al.*, 2019] showed patients followed four distinct modes: **(1) High adherence:** adhere daily regardless of health worker action. **(2) Low adherence:** Very low adherence regardless of health worker action. **(3) Receptive patients:** Irregular adherence but can benefit from intervention. **(4) Dropout patients:** Like receptive patients but have probability of dropping out.

We implement these patient types in our simulation in the following mix respectively: **0.64, 0.01, 0.175, 0.175**, matching the number of High and Low adherence patients observed in the data, and splitting the remaining types. We run experiments with $d = 3,4,5$, with worker actions as follows: **(1) Call:** mild boost to adherence (c=1). **(2) Visit:** good boost to adherence (c=2). **(3) Escalate:** Patient returns to the maximum adherence state. Rewards are defined as (adherence level)/d.

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. 3 shows the performance and runtime for the dataset with budget of $0.1N$ for $d = 3,4,5$. With such a small budget, the tradeoff between individual actions is important. We see 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 (complexity for Hawkins and BLam given in Appendix 11), 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. These results demonstrate the exemplary ability for our approach to scale well without sacrificing performance on a dataset whose technical structure has not been established a priori.

6 Conclusion

Our work makes available well-performing policies for (MA)^{2}RBs at scale. Our algorithm offers vast speedups and can be readily adapted to new problems without first arduously deriving a problem-specific index policy, both of which are key benefits when approaching understudied AI4SG domains.

References

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

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

[Glazebrook *et al.*, 2011] Kevin D. Glazebrook, David J. Hodge, and Chris Kirkbride. General notions of indexability for queueing control and asset management. *The Annals of Applied Probability*, 21(3):876–907, 2011.

[Hawkins, 2003] Jeffrey Thomas Hawkins. *A Langrangian decomposition approach to weakly coupled dynamic optimization problems and its applications*. PhD thesis, Massachusetts Institute of Technology, 2003.

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

[Jiang *et al.*, 2020] Shunhua Jiang, Zhao Song, Omri Weinstein, and Hengjie Zhang. Faster dynamic matrix inverse for faster lps. *arXiv preprint arXiv:2004.07470*, 2020.

[Killian *et al.*, 2019] Jackson A Killian, Bryan Wilder, Amit Sharma, Vinod Choudhary, Bistra Dilkina, and Milind Tambe. 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*, pages 2430–2438, 2019.

[Lee *et al.*, 2019] Elliot Lee, Mariel S Lavieri, and Michael Volk. Optimal screening for hepatocellular carcinoma: A restless bandit model. *Manufacturing & Service Operations Management*, 21(1):198–212, 2019.

[Mate *et al.*, 2020] Aditya Mate, Jackson A Killian, Haifeng Xu, Andrew Perrault, and Milind Tambe. Collapsing bandits and their application to public health interventions. In *Advances in Neural Information Processing Systems*, 2020.

[Meuleau *et al.*, 1998] Nicolas Meuleau, Milos Hauskrecht, Kee-Eung Kim, Leonid Peshkin, Leslie Pack Kaelbling, Thomas L Dean, and Craig Boutilier. Solving very large weakly coupled markov decision processes. In *AAAI/IAAI*, pages 165–172, 1998.

[Nguyen *et al.*, 2013] Thanh H Nguyen, Arunesh Sinha, Shahrzad Gholami, Andrew Plumptre, Lucas Joppa, Milind Tambe, Margaret Driciru, Fred Wanyama, Aggrey Rwetsiba, and Rob Critchlow. Capture: A new predictive anti-poaching tool for wildlife protection. 2013.

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

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

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

[Qian *et al.*, 2016] Yundi Qian, Chao Zhang, Bhaskar Krishnamachari, and Milind Tambe. Restless poachers: Handling exploration-exploitation tradeoffs in security domains. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 123–131, 2016.

[Ruiz-Hernández *et al.*, 2020] Diego Ruiz-Hernández, Jesús M Pinar-Pérez, and David Delgado-Gómez. Multi-machine preventive maintenance scheduling with imperfect interventions: A restless bandit approach. *Computers & Operations Research*, 119:104927, 2020.

[Verloop and others, 2016] Ina Maria Verloop et al. Asymptotically optimal priority policies for indexable and nonindexable restless bandits. *The Annals of Applied Probability*, 26(4):1947–1995, 2016.

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

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

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

[Zayas-Caban *et al.*, 2019] Gabriel Zayas-Caban, Stefanus Jasin, and Guihua Wang. An asymptotically optimal heuristic for general nonstationary finite-horizon restless multi-armed, multi-action bandits. *Advances in Applied Probability*, 51(3):745–772, 2019.

Appendix

Algorithm 2: BuildBounds

```
  Data: $D,G,N$
  $\mathcal{U}$ = [] ; // list of dicts for upper bound pieces
  $\mathcal{L}$ = [] ; // list of dicts for lower bound pieces
  for $i \in 0,...,N-1$ do
      /* Computing upper bounds, 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 lower bounds, 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 $\mathcal{U},\mathcal{L}$
```

7 Proof of Prop 4.2

Proposition 7.1. *The optimal solution to Eq. 2 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. 2 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. □

8 Proof of Thm. 4.3

We now can prove our main result:

Theorem 8.1. $\lambda_\ell \le \lambda_{min} \le \lambda_u$

Proof. The proof is best seen by considering λ_{min} which solves $J(\boldsymbol{s},\lambda)$, i.e., Eq. 2. 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. 2. 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. □

Algorithm 3: BLamPrecompute

```
  Data: $T,R,C,N,G,\beta$
  $\mathcal{D}$ = [] ; // hold slopes at each arm test point
  $\epsilon_0 = 1e\text{-}3$;
  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 $\mathcal{U}$, $\mathcal{L}$
```

9 BLam: Choosing first K arms

One important choice is in selecting the first K arms. Intuitively, the best $V^i(s^i,\lambda)$ to include in Eq. 3 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. 1). 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. 1).

Once λ_u and λ_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 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. 2, the result of which we will use to derive feasible policies in Section 10.

10 Computing a Policy

Once λ_{min} is finalized, and the resulting value functions from Eq. 2 have been computed, we use them to compute the one-step greedy policy implied by the bound. To do this, we 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 next, followed by an algorithm for computing $Q^i(s^i,a,\lambda_{min})$ from value functions.

$$\max_X \sum_{i=0}^{N-1}\sum_{j=0}^{M-1} x_{i,j} Q^i(s^i,a_j,\lambda_{min}) \tag{4}$$

$$\text{s.t.} \sum_{i=0}^{N-1}\sum_{j=0}^{M-1} x_{i,j} c_j \le B \tag{5}$$

$$\sum_{j=0}^{M-1} x_{i,j} = 1 \quad \forall i \in 0...N-1 \tag{6}$$

$$\tag{7}$$

$Q^i(s^i,a_j,\lambda_{min})$ 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 4: Compute Action Value Function

```
Data: $V,T,R,C,\lambda,\beta$
1 Q = [] ;    // hold the action value function
2 for $s \in \mathcal{S}$ do
3     for $a \in \mathcal{A}$ do
4         $Q[s,a] = R[s] - \lambda * C[a] + \beta * \sum_{s' \in \mathcal{S}} V[s'] * T[s,a,s']$
5 return Q
```

11 BLam/Hawkins 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 $\mathcal{O}(NG^i VI(|\mathcal{S}|,|\mathcal{A}|))$ 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 [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. 3 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

$$\Omega(\sqrt{N}|S|^2|A| + N|S|^2 + N^{\frac{3}{2}}|S| + N^2) \tag{8}$$

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

$$\mathcal{O}(N|S|^2|A| + N^2|S|^2) \tag{9}$$

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

$$\mathcal{O}(N|S|^2|A| + N^{\frac{5}{2}}|S|^2) \tag{10}$$

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.