SEQUENTIAL RESOURCE ALLOCATION UNDER UNCERTAINTY: AN INDEX POLICY APPROACH

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Weici Hu

August 2017

SEQUENTIAL RESOURCE ALLOCATION UNDER UNCERTAINTY: AN INDEX POLICY APPROACH

Weici Hu, Ph.D.

Cornell University 2017

We consider a class of stochastic sequential allocation problems - restless multi-armed bandits (RMAB) with a finite horizon and multiple pulls per period. Leveraging the Lagrangian relaxation of the problem, we propose an index-based policy that uses the optimal Lagrange multipliers to index individual arms, and prove that the policy is asymptotically optimal as the number of arms tends to infinity. We also demonstrate numerically that this index-based policy outperforms state-of-the-art heuristics in several instances of RMAB. In addition, we study two other applications of sequential resource allocation problems which are extensions of the RMAB problem, and demonstrate how our index policy can be adapted to these settings.

BIOGRAPHICAL SKETCH

Born as a mainlander, Weici Hu spent most of her childhood in a southern city of China. She then went further south to Singapore to complete her secondary school and high school education. Tired of the tropical climate, she spent the past 8 years acquiring a college degree and trying to acquire a PhD degree in the Northeast of the United States.

This document is dedicated to my mother.

ACKNOWLEDGEMENTS

I would like to first thank my advisor for his guidance and extreme patience and for not kicking me out of the program. I would also like to thank my parents for their non-interference policy which has allowed me to experience life in its truest states. Lastly I would like to thank two of my best pals in graduate school: Wei Qian and Kenneth Chong, for handing me water and energy bars during my Marathon race.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

We consider a general class of dynamic resource allocation problems with average-case criteria. Such problems enjoy a variety of applications in a wide range of industries. Examples include:

- Facebook displays ads in the *suggested posts* section every time its users browse their personal pages. Among the ads that have been shown, some are known to attract more clicks than others. But there are also many ads which have yet to be shown and they may attract even more clicks. Given that the slots for display are limited, a policy is required to select ads to maximize total clicks.
- In a multi-stage clinical trial, a medical group starts with a number of new treatments and an existing treatment with reliable performance. In each stage, a few treatments are selected from the pool to test, with the goal to identifying the new treatments that perform better than the existing one with high confidence. A strategy is required to select which treatments to test at every stage to most effectively support their judgment at the end of the trial.
- A data analyst wishes to label a large number of images using crowdsourced effort from low-cost but potentially inaccurate workers. Each label given by the crowdworkers comes with a cost and the analyst has limited budget. Hence she needs to carefully assign tasks to workers so as to maximize the likelihood of correct labeling.

We formulate such problems as instances of the restless multiarmed bandit (RMAB) problem [48] with a finite horizon and multiple pulls per period, which, in turn, is closely related to a broader class of problem called *Weakly Coupled Dynamic Programs*

(WCDP) [25]. In the RMAB, we have a collection of "arms", each of which is endowed with a state that evolves independently. If the arm is "pulled" or "engaged' in a time period then it advances stochastically according to one transition kernel, and if not then it advances according to a different kernel. Rewards are generated with each transition, and our goal is to maximize the expected total reward over a finite horizon, subject to a constraint on the number of arms pulled in each time period. The RMAB forms a generalization of the more famous multi-armed bandit (MAB) [41] by allowing arms that are not engaged to change state and multiple pulls per period.

Theoretically an optimal solution of a RMAB can be obtained by leveraging the *Bellman equation* and solving it as a dynamic program (DP) [40]. However, this approach becomes computationally infeasible when the state space grows large. In particular, the state space grows exponentially with the number of arms in a RMAB, and the number of arms are often large in practice. This approach therefore suffers the so-called "curse of dimensionality" [39]. Much research has been dedicated to efficiently finding "good" solutions. Gittin in 1979 proposed a tractable-to-compute optimal policy for the infinite horizon MAB with one pull per time period, which is famously known as the Gittins index policy [21]. This policy is appealing because it can be computed by considering the state space for only a single arm, making it computationally tractable for problems with many arms. This policy loses its optimality properties, however, when modifying the problem in any problem dimension: when allowing arms that are not engaged to change state; when moving to a finite horizon [6]; or when allowing multiple pulls per period. Thus, the Gittins index does not apply to our problem setting.

While the RMAB is not known to have a computable optimal policy, [48] proposed a heuristic called the Whittle index for the infinite-horizon RMAB with multiple pulls per period, which is well-defined when arms satisfy an indexability condition. This policy

is derived by considering a Lagrangian relaxtion of the RMAB in which the constraint on the number of arms pulled is replaced by a penalty paid for pulling an arm. An arm's Whittle index is then the penalty that makes a rational player indifferent between pulling and not pulling that arm. The Whittle index policy then pulls those arms with the highest Whittle indices. Appealingly, the Whittle index and the Gittins index are identical when applied to the MAB problem with a single pull per period. [48] further conjectured that if the number of arms and the number of pulls in each time period go to infinity at the same rate in an infinite-horizon RMAB, then the Whittle index policy is asymptotically optimal when arms are indexable. [46, 47] gave a proof to Whittle's conjecture with a difficult-to-verify condition: that the fluid approximation has a globally asymptotically stable equilibrium point. This condition was shown to hold when each arm's state space has at most 3 states, but this condition does not hold in general and [46] provides a counterexample with 4 states.

Our contribution in this dissertation is to (1) create an index policy for finite horizon RMABs with multiple pulls per period, and (2) show that it is asymptotically optimal in the same limit considered by Whittle. Like the Whittle index, our approach is computationally appealing because it requires considering the state space for only a single arm, and its computational complexity does not grow with the number of arms. Unlike the Whitle index, our index policy does not require an indexability condition to be well-defined, and in contrast with [46, 47] our proof of asymptotic optimality holds regardless of the number of states, and does not depend on hard-to-verify conditions. We further demonstrate our index policy numerically on problems from the literature that can be formulated as finite-horizon RMABs, and show that it provides finite-sample performance that improves over the state-of-the-art. We also use our framework to develop policies for two major RMAB-like applied problems: Multiple comparison with a standard (MCS) in the field of simulation and crowdsourcing in the field of artificial

intelligence, and demonstrate numerically that our index-based policies out-perform the state-of-art using both real and synthetic data.

In addition to building on [48, 46, 47], our work builds on the literature in weakly coupled dynamic programs (WCDP), that itself builds on RMABs. Indeed, at the end of his paper, Whittle pointed out that his relaxation technique can be applied to a more general class of problems in which sub-problems are linked by constraints on actions, but are otherwise independent. Hawkins in his thesis [25] formally termed these problems (but with a more general type of constraints) as WCDPs and proposed a general decoupling technique. Moreover, he proposed a minimal-lambda policy for infinite horizon WCDPs which, like the Whittle index policy, is derived by considering a Lagrangian relaxation of the WCDP. The minimal-lambda policy finds the smallest Lagrange multiplier so that the current optimal decision for the Lagrangian relaxation is also feasible for the original WCDP. The minimal-lambda policy then pulls arms according to this optimal decision. In the case of RMAB, the minimal-lambda policy is equivalent to the Whittle index with the smallest of the indices of all pulled arms for the former being the same as the smallest Lagrange multiplier to attain a feasible solution for the latter.

Another major work in WCDP is [1] which shows that the ADP relaxation is tighter than the Lagrangian relaxation but is also computationally more expensive. It gives necessary and sufficient conditions for the Lagrangian relaxation to be tight and proves that the optimality gap is bounded by a constant when the Lagrange multipliers are allowed to be state dependent. The last result that the optimality gap is bounded by a constant implies that the per arm gap goes to zero as the number of arms grows. We achieve a similar result in the dissertation by showing the per arm reward of our index-based heuristic policy goes to the per-arm reward of the Lagrangian bound, despite that our Lagrange multipliers not being state-dependent. While there are similarities between

the two works, the focus differs: while our work focuses on offering an asymptotically optimal heuristic policy, [1] examines the ordering and tightness of different bounds. The heuristic proposed in [1] is based on an ADP technique, and is different from our index-based policy.

Other work on WCDP also includes [52] who proposes an even tighter bound by incorporating an information relaxation on the non-anticipative constraints in addition to the existing relaxation methods. [24] considers two classes of large-scale WCDPs in which the state and action space in each sub-problem also grows exponentially and uses an ADP technique to approximate the value functions of individual sub-MDPs in addition to employing a Lagrangian relaxation for the overall problem.

In this thesis, we use Chapter 2 to present the main theoretical work on RMAB and propose an index-based policy. Chapter 3 and 4 present applied problems that are similar to RMAB but with some variations, and index-based policies for these problems. Below is a brief outline of the chapters in the dissertation:

- In Chapter 2, we consider the RMAB with a finite horizon and multiple pulls per period. Leveraging a Lagrangian relaxation, we approximate the RMAB with a problem that can be decomposed into a collection of single arm problems. We then propose an index-based policy that uses optimal solutions of the single arm problems to index individual arms, and offer a proof that it is asymptotically optimal as the number of arms tends to infinity. We also use simulation to show that this index-based policy performs better than the state-of-art heuristics in various problem settings.
- In Chapter 3, we consider the problem of multiple comparisons with a known standard, in which we wish to allocate simulation effort efficiently across a finite number of simulated systems, to determine which systems have mean performance

exceeding a known threshold. We suppose that parallel computing resources are available, and that we are given a fixed simulation budget. We consider this problem in a Bayesian setting, and formulate it as a stochastic dynamic program. The set-up of the problem is the same as a RMAB except that every simulated system (the arms) is allowed to have multiple computing resources (the pulls) in a time period. For simplicity, we focus on Bernoulli sampling, with a linear loss function. Using links to restless multi-armed bandits, we provide a computationally tractable upper bound on the value of the Bayes-optimal policy, and an index policy motivated by these upper bounds. This chapter has been published in the proceedings of the 2014 *Winter Simulation Conference*.

- In Chapter 4, we consider effort allocation in crowdsourcing, where we wish to assign labeling tasks to imperfect homogeneous crowd workers to maximize overall accuracy in a continuous-time Bayesian setting, subject to budget and time constraints. The Bayes-optimal policy for this problem is the solution to a partially observable Markov decision process, but the curse of dimensionality renders the computation infeasible. Based on the Lagrangian Relaxation technique in [1], we provide a computationally tractable instance-specific upper bound on the value of this Bayes-optimal policy, which can in turn be used to bound the optimality gap of any other sub-optimal policy. In an approach similar in spirit to the Whittle index for restless multi-armed bandits, we provide an index policy for effort allocation in crowdsourcing and demonstrate numerically that it outperforms the state of the art and is near-optimal. This chapter has been published in the proceedings of the 2016 *Artificial Intelligence and Statistics Conference*.

CHAPTER 2

AN ASYMPTOTICALLY OPTIMAL INDEX POLICY FOR RMAB

Chapter 2 presents the major theoretical work of this dissertation. In this chapter, Section 2.1 formulates the problem, Section 2.2 discusses the Lagrangian relaxation of the problem, Section 2.3 states our index-based policy and provides computational methods, Section 2.4 gives a proof of asymptotic optimality, Section 2.5 numerically evaluates our index policy, and Section 7 concludes.

2.1 Problem Description and Notation

We consider an MDP $(\mathbb{S}^K, \mathbb{A}^K, \mathbb{P}, R)$ which is created by a collection of K sub-processes $(\mathbb{S}, \mathbb{A}, P, r)$. The sub-processes are independent of each other except that the actions taken by each sub-process have to jointly satisfy some constraints at each time step. These sub-processes are also referred to as *arms* in the bandit literature and shall be indexed by $x \in \{1, ..., K\}$. Following a standard construction for MDPs, both the larger joint MDP and the sub-processes will be constructed on the same measurable space $(\Omega, \mathbf{F})$. Random variables on this measurable space will correspond to states, actions, rewards, and each policy will induce a probability measure over this space.

We describe the MDP to consider formally as comprising:

- The **time horizon** $T < \infty$.
- The **state space** $\mathbb{S}^K$ is the cross product of K sub-processes' state space $\mathbb{S}$, which is assumed to be finite. We use $\mathbf{s} = (s_1, ..., s_K)$ to denote an element in $\mathbb{S}^K$ and $\mathbf{S}$ when the state is random. We also use $\mathbf{S}_t$ to emphasize that the state is at time t. Likewise, we use s to denote an element in $\mathbb{S}$, and S or S_t when it is random.

- The **action space** $\mathbb{A}^K$ is the cross product of K sub-processes' action space $\mathbb{A} = \{0, 1\}$. We use a to denote a generic element of $\mathbb{A}$, and A when it is random. We use $\mathbf{a} = (a_1, a_2, ..., a_K)$ to denote a generic element in $\mathbb{A}^K$ and $\mathbf{A}$ when it is random. In the context of bandit problems, $a = 1$ is called "pulling" an arm (sub-process).
- The **reward function** $R_t : \mathbb{S}^K \times \mathbb{A}^K \mapsto \mathbb{R}$ for each $1 \leq t \leq T$. $R_t(\mathbf{s}, \mathbf{a}) = \sum_{x=1}^K r_t(s_x, a_x)$, where $r_t(s_x, a_x)$ is the reward obtained by a sub-process when action a_x is taken in state s_x at time t. We assume rewards are non-negative and finite.
- The **transition kernel** $\mathbb{P}^{\mathbf{a}}(\mathbf{s}', \mathbf{s}) = \prod_{x=1}^K P^{a_x}(s'_x, s_x)$, where $P^a(s', s)$ is the probability of a sub-process transitioning from s' to s if action a is taken, i.e., $P(s|s', a)$. The product implies that the K sub-processes evolve independently. RMAB differ from MAB in that MABs require $P^0(s, s) = 1$ while RMABs allows $P^0(s, s) < 1$. Since we are considering both cases, we do not restrict the value of $P^0(s, s)$.

Next we describe the set of policies for our MDP problem. Since the state and action space defined above are finite, it is sufficient to consider the set of Markov policies $\mathbf{\Pi}$ [40]. Define a policy $\boldsymbol{\pi} \in \mathbf{\Pi}$ as a function $\mathbb{S}^K \times \mathbb{A}^K \times \{1, ..., T\} \to [0, 1]$ that determines the probability of choosing action $\mathbf{a}$ in state $\mathbf{s}$ at time t. Subsequently we have $\sum_{\mathbf{a} \in \mathbb{A}^K} \boldsymbol{\pi}(\mathbf{s}, \mathbf{a}, t) = 1$, $\forall \mathbf{s} \in \mathbb{S}^K, \forall 1 \leq t \leq T$. A policy $\boldsymbol{\pi}$ and the transition kernel $\mathbb{P}^{\cdot}(\cdot, \cdot)$ together defines a probability distribution $P^{\boldsymbol{\pi}}$ on all possible paths of the process $\{\mathbf{s}_1\mathbf{a}_1...\mathbf{s}_T : \mathbf{s}_t \in \mathbb{S}^K, \mathbf{a}_t \in \mathbb{A}^K\}$. Starting at a fixed state $\mathbf{s}_1$, i.e., $P^{\boldsymbol{\pi}}(\mathbf{S}_1 = \mathbf{s}_1) = 1$, we have the conditional distributions of $\mathbf{S}_t$ and $\mathbf{A}_t$ defined recursively by $P^{\boldsymbol{\pi}}(\mathbf{S}_{t+1} = \mathbf{s}'|\mathbf{S}_t = \mathbf{s}, \mathbf{A}_t = \mathbf{a}) = \mathbb{P}^{\mathbf{a}}(\mathbf{s}, \mathbf{s}')$ and $P^{\boldsymbol{\pi}}(\mathbf{A}_t = \mathbf{a}|\mathbf{S}_t = \mathbf{s}) = \boldsymbol{\pi}(\mathbf{s}, \mathbf{a}, t)$.

The MDP we are considering allows exactly m sub-processes to be set active at each time step. Hence a feasible policy, $\boldsymbol{\pi} \in \mathbf{\Pi}$, has to satisfy $P^{\boldsymbol{\pi}}(|\mathbf{A}_t| = m) = 1$, $\forall t \in \{1, ..., T\}$. Here we use $|\cdot|$ as an operator that sums all the elements in a vector.

The objective of our MDP is:

$$\begin{aligned} \underset{\boldsymbol{\pi}\in\boldsymbol{\Pi}}{\text{maximize}} \quad & \mathbb{E}^{\boldsymbol{\pi}}\left[\sum_{t=1}^{T} R_t(\mathbf{S}_t, \mathbf{A}_t)\right] \\ \text{subject to} \quad & P^{\boldsymbol{\pi}}(|\mathbf{A}_t| = m_t) = 1, \quad \forall 1 \le t \le T. \end{aligned} \tag{2.1}$$

Since we will discuss other MDPs in the process of solving this one, (2.1) will be referred to as the *original* MDP in the rest of the chapter to avoid confusion. For convenience, we summarize our notation in Appendix A.1.

The original MDP (2.1) suffers from the "curse of dimensionality", and hence solving it is computationally intractable. In the remainder of the chapter we build a computationally feasible index-based heuristics with a performance guarantee.

2.2 Lagrangian Relaxation and Upper Bounds

In this section we discuss the Lagrangian relaxation of the original MDP and the corresponding single process problem. These single process problems together with the Lagrange multipliers form the building blocks of our index-based policy, which will be formally introduced in Section 2.3. The Lagrangian relaxation considers an unconstrained problem whose objective is obtained by augmenting the objective of (2.1):

$$P(\boldsymbol{\lambda}) = \max_{\boldsymbol{\pi}\in\boldsymbol{\Pi}} \mathbb{E}^{\boldsymbol{\pi}}\left[\sum_{t=1}^{T} R_t(\mathbf{S}_t, \mathbf{A}_t)\right] - \mathbb{E}^{\boldsymbol{\pi}}\left[\sum_{t=1}^{T} \lambda_t(|\mathbf{A}_t| - m_t)\right], \tag{2.2}$$

for any $\boldsymbol{\lambda} = \{\lambda_1, ..., \lambda_T\} \in \mathbb{R}^T$. This unconstrained problem has the following property:

Lemma 1. *For any $\boldsymbol{\lambda} \in \mathbb{R}^T$, $P(\boldsymbol{\lambda})$ is an upper bound to the optimal value of the original MDP.*

[1] gave a proof to Lemma 1 using the Bellman equation. We provide a more

straightforward proof by viewing $P(\boldsymbol{\lambda})$ as the Lagrange dual function of a relaxed problem of the original MDP; see Appendix A.2.

This Lagrangian relaxation then decomposes into K smaller MDPs, which we can easily solve to optimality. To elaborate on this idea of decomposition, we construct a *sub-MDP* problem based on tuple $(\mathbb{S}, \mathbb{A}, P(\cdot,\cdot), r(\cdot,\cdot))$. Again we consider only the set of Markov policies, Π, for this problem. Similarly a policy $\pi \in \Pi$ is a function that determines the probability of choosing action a in state s at time t, i.e., $\pi : \mathbb{S} \times \mathbb{A} \times \{1, ..., T\} \to [0, 1]$. The sub-MDP starts at a fixed state s_1. Subsequently we can define distributions of S_t and A_t under P^π in a similar manner as we did for $\mathbf{S}_t$ and $\mathbf{A}_t$ in the previous section. The objective of the sub-MDP is:

$$Q(\boldsymbol{\lambda}) = \max_{\pi \in \Pi} \mathbb{E}^\pi \left[\sum_{t=1}^{T} r_t(S_t, A_t) - \lambda_t A_t \right]. \tag{2.3}$$

We are now ready to present the decomposition of the Lagrangian relaxation.

Lemma 2. *The optimal value of the relaxed problem satisfies*

$$P(\boldsymbol{\lambda}) = KQ(\boldsymbol{\lambda}) + \sum_t m_t \lambda_t, \tag{2.4}$$

[1] also gave a proof to Lemma 2, and we again provide a different proof in Appendix A.3. Since the state space of the sub-MDP is much smaller, we can solve it directly by using backward induction and the optimality equation. The existence of such an optimal Markov deterministic policy follows from that the state and action spaces of the sub-MDP being finite [40]. Let $\Pi^*(\boldsymbol{\lambda})$ be the set of optimal Markov deterministic policies of the sub-MDP for a given $\boldsymbol{\lambda}$. The relaxed problem can be solved by combining the solutions of individual sub-MDPs, that is, we can construct an optimal policy of the relaxed problem $\boldsymbol{\pi}^{\boldsymbol{\lambda}}$ by setting $\boldsymbol{\pi}^{\boldsymbol{\lambda}}(\mathbf{s}, \mathbf{a}, t) = \prod_{x=1}^{K} \pi^{\boldsymbol{\lambda}}(s_x, a_x, t)$, where $\pi^{\boldsymbol{\lambda}}$ is an element in $\Pi^*(\boldsymbol{\lambda})$. Moreover, $\mathbf{P}(\boldsymbol{\lambda})$ is convex and piecewise linear in $\boldsymbol{\lambda}$ [1].

2.3 An Index Based Heuristic Policy

Our index based heuristic policy assigns an index to each sub-process, based upon its state and current time. At each time step, we set active the m sub-processes with the highest indices. Before carrying out the process of sequential decision-making, our index policy calls for pre-computation of 1) $\boldsymbol{\lambda}^* \in \arg\inf_{\boldsymbol{\lambda}} P(\boldsymbol{\lambda})$, as defined in Section 2.2; 2) a set of indices, $\boldsymbol{\beta}$, that will later be used for decision-making at every time step; 3) an optimal policy π^{**} for the sub-MDP problem in (2.3). In the first part of this section we discuss how we carry out such computations.

2.3.1 Pre-computations

Dual optimal $\boldsymbol{\lambda}^*$

We use subgradient descent to solve $\inf_{\boldsymbol{\lambda}} P(\boldsymbol{\lambda})$, which converges to its solution $\boldsymbol{\lambda}^*$ by convexity of $\boldsymbol{\lambda} \mapsto P(\boldsymbol{\lambda})$ (Theorem 7.4 in [42]). By (2.3) and (2.4), a sub-gradient of $P(\boldsymbol{\lambda})$ with respect to $\boldsymbol{\lambda}$ is given by $(-K\mathbb{E}^{\pi^{\boldsymbol{\lambda}}}[A_t] + m : 1 \leq t \leq T)$, where $\pi^{\boldsymbol{\lambda}}$ is any policy in $\Pi^*(\boldsymbol{\lambda})$.

To compute this sub-gradient, we compute a policy $\pi^{\boldsymbol{\lambda}}$ in $\Pi^*(\boldsymbol{\lambda})$ and then use exact computation or simulation with a large number of replications to compute $\mathbb{E}^{\pi^{\boldsymbol{\lambda}}}[A_t]$. To compute a policy in $\Pi^*(\boldsymbol{\lambda})$, we first compute the value function $V^{\boldsymbol{\lambda}} : \mathbb{S} \times \{1, ..., T\} \mapsto \mathbb{R}$ of sub-MDP $Q(\boldsymbol{\lambda})$. We accomplish this using backward induction [40]:

$$V^{\boldsymbol{\lambda}}(s,t) = \begin{cases} \max_{a \in \mathbb{A}} \{r_T(s,a) - a\lambda_T\} & \text{if } t = T, \\ \max_{a \in \mathbb{A}} \{r_t(s,a) - a\lambda_t + \sum_{s' \in \mathbb{S}} P^a(s,s') V^{\boldsymbol{\lambda}}(s', t+1)\} & \text{otherwise.} \end{cases} \tag{2.5}$$

Recalling that $\Pi^*(\boldsymbol{\lambda})$ includes only deterministic policies, a policy $\pi^{\boldsymbol{\lambda}}$ in $\Pi^*(\boldsymbol{\lambda})$ are constructed by determining for each s and t the action a whose one-step lookahead value

$r_t(s,a) - a\lambda_t + \sum_{s' \in \mathbb{S}} P^a(s,s') V^{\boldsymbol{\lambda}}(s', t+1)$ is equal to $V^{\boldsymbol{\lambda}}(s,t)$, and then setting $\pi^{\boldsymbol{\lambda}}(s,a,t) = 1$ for this a. For those s and t for which both actions a have one-step lookahead values equal to $V^{\boldsymbol{\lambda}}(s,t)$, one may set $\pi^{\boldsymbol{\lambda}}(s,a,t) = 1$ for either such action. Thus, the cardinality of $\Pi^*(\boldsymbol{\lambda})$ is 2 raised to the power of the number of s,t for which the one-step lookahead values for playing and not playing are tied.

When we construct a policy in $\Pi^*(\boldsymbol{\lambda})$ for the purpose of computing a sub-gradient of $P(\boldsymbol{\lambda})$, we choose to play in those s,t with tied one-step lookahead values. While our subgradient descent algorithm would converge for other choices, making this choice better supports computation of indices in section 2.3.1.

Indices $\beta_t(s)$

Define the vector $\mathbf{v}[a,t]$ to be $\mathbf{v} + (a - v_t) * \mathbf{e}_t$, that is, the vector $\mathbf{v}$ with the t^{th} element replaced by $a \in \mathbb{R}$. We define the *index* of state $s \in \mathbb{S}$ at time t as

$$\beta_t(s) = \sup\{\beta : \exists\, \pi \in \Pi^*(\boldsymbol{\lambda}^*[\beta,t])\ s.t.\ \pi(s,1,t) = 1\}. \tag{2.6}$$

Instead of computing the entire set $\Pi^*(\boldsymbol{\lambda}^*[\beta,t])$, we only need to compute a policy in $\Pi^*(\boldsymbol{\lambda}^*[\beta,t])$ using the method discussed in section 2.3.1, i.e., always choose the active action when there are ties. Intuitively, this index is the maximum price we are willing to pay to set a sub-process active in state s at t. By leveraging the monotonicity of optimal actions with respect to rewards, as shown in Lemma 13 in Appendix A.7, we compute $\beta_t(s)$ via bisection search in the interval $[0, U]$, where U upper bounds the largest possible value of $\beta_t(s)$. For example, we can set U as $T * \max_{s,a,t} r_t(s,a)$ when $\boldsymbol{\lambda}^* \geq 0$ (we show in Appendix A.6 that $\beta_t(s)$ cannot be greater than this value in this case). We pre-compute the set $\boldsymbol{\beta} = \{\beta_t(s) : s \in \mathbb{S}, 1 \leq t \leq T\}$ before running the actual algorithm.

Occupation measure ρ^* and its corresponding optimal policy π^{**}

Our tie-breaking policy involves constructing an optimal Markov policy π^{**} for the sub-MDP $Q(\boldsymbol{\lambda}^*)$ such that $\mathbb{E}^{\pi^{**}}[A_t] = \frac{m}{K}$, $\forall 1 \leq t \leq T$. The existence of π^{**} is shown in Appendix A.5. To compute π^{**}, we borrow the idea of the occupation measure [18]. Define the occupation measure, $\rho(s, a, t)$, induced by a policy π to be the probability of being in state s and taking action a at time t under π. Subsequently π^{**} can be computed by solving the following linear program (LP):

$$\begin{aligned}
\max_{\{\rho(s,a,t):y\in\mathbb{S},a\in\mathbb{A},t\in\{1,\ldots,T\}\}} \quad & \sum_{t=1}^{T}\sum_{a\in\mathbb{A}}\sum_{s\in\mathbb{S}}\rho(s,a,t)r_t(s,a) \\
\text{subject to} \quad & \sum_{s\in\mathbb{S}}\rho(s,1,t) = \frac{m_t}{K}, \forall t = 1\ldots,T \\
& \sum_{a\in\mathbb{A}}\rho(s,a,t) - \sum_{a\in\mathbb{A}}\sum_{s'\in\mathbb{S}}\rho(s',a,t-1)P^a(s',s) = 0, \\
& \qquad\qquad \forall s\in\mathbb{S}, 2\leq t\leq T \\
& \sum_{a\in\mathbb{A}}\rho(s,a,1) = \mathbb{1}(s=s_1) \;\; \forall s\in\mathbb{S} \\
& \rho(s,a,t)\geq 0, \;\; \forall s\in\mathbb{S}, a\in\mathbb{A}, t=1,\ldots,T,
\end{aligned} \tag{2.7}$$

The first constraint ensures that $\mathbb{E}^{\pi^{**}}[A_t] = \frac{m}{K}$. The second constraint ensures flow balance. The third constraint shows that we start at state s_1. The second and third constraint together imply $\sum_{a\in\mathbb{A},s\in\mathbb{S}}\rho(s,a,t) = 1$, i.e., that $(\rho(s,a,t) : a\in\mathbb{A}, s\in\mathbb{S})$ is a probability distribution for each t. The fourth and fifth constraints ensure that ρ is a valid probability measure.

Let ρ^* be an optimal solution to (2.7). π^{**} can then be constructed by

$$\pi^{**}(s,a,t) = \begin{cases} \frac{\rho^*(s,a,t)}{\sum_{a\in\mathbb{A}}\rho^*(s,a,t)}, & \text{if } \sum_{a\in\mathbb{A}}\rho^*(s,a,t) > 0 \\ \mathbb{1}(a=1), & \text{if } \sum_{a\in\mathbb{A}}\rho^*(s,a,t) = 0 \text{ and } \beta_t(s)\geq\lambda_t^* \\ \mathbb{1}(a=0), & \text{if } \sum_{a\in\mathbb{A}}\rho^*(s,a,t) = 0 \text{ and } \beta_t(s)<\lambda_t^*, \end{cases} \tag{2.8}$$

for all $s \in \mathbb{S}, a \in \mathbb{A}, 1 \leq t \leq T$.

Here we also make an observation that $\boldsymbol{\lambda}^* \in \arg\min P(\boldsymbol{\lambda}^*)$ is the optimal dual variable corresponding to the first constraint in 2.7.

2.3.2 Index policy

Let $\{\beta_t(S_{t,x}) : x \in \{1, ..., K\}\}$ be the indices associated with the K sub-processes at time t. We define $\bar{\beta}_t(\mathbf{S}_t)$ to be the largest value β in $\{\beta_t(S_{t,x}) : x \in \{1, ..., K\}\}$ such that at least m sub-processes have indices of at least β. Our index policy then sets the actions of sub-processes with indices strictly greater than $\bar{\beta}_t$ to 1 (active), and those with indices strictly less than $\bar{\beta}_t$ to 0 (inactive). When more than m sub-processes have indices greater than or equal to $\bar{\beta}_t$, a tie-breaking a rule is needed. For simplicity, in the following discussion we use the term *remaining resources* to refer to the remaining number of the arms to be set active after we activate all the arms with indices greater than $\bar{\beta}_t(\mathbf{S}_t)$, and use $I_t = \{S_{t,x} : 1 \leq x \leq K, \beta_t(S_{t,x}) = \bar{\beta}_t(\mathbf{S}_t)\}$ to denote the set of states occupied by the sub-processes with tied indices. Our tie-breaking rule allocates the remaining resources across I_t according to the probability distribution induced by π^{**} over $\mathbb{S}$ at time t. More specifically, we allocate

$$q_t(S_{t,x}) = \begin{cases} \frac{\rho^*(S_{t,x},1,t)}{\sum_{s' \in I_t} \rho^*(s',1,t)}, \text{ if } \sum_{s' \in I_t} \rho^*(s',1,t) > 0 \\ \frac{N_t(S_{t,x})}{\sum_{s' \in I_t} N_t(s')}, \text{ otherwise} \end{cases} \tag{2.9}$$

fraction of the remaining resources to each of the state in I_t, where $N_t(s)$ denote the number of sub-processes in state s at time t.

We then use the function Rounding(total, frac, avail) in Algorithm 2 to deal with situations where the products between the desired fractions and the remaining resources

are not integers. Here *total* represents the number of remaining resources, $frac$ is a vector of the fractions of the remaining resources to be approximated allocated to each tied state, and *avail* is a vector of the number of sub-processes in each tied state. The function also allows the number of sub-processes in a tied state s to be less than the number of resources we would like to assign to s according to the fraction in (2.9). We note the following property of this function Rounding, which we will rely on in our proof in Section 6.

Remark 1. *When total, avail, frac satisfy* $avail_i \geq total * frac_i$*, the output vector* $b = Rounding(total, frac, avail)$ *satisfies* $|b_i - total * frac_i| < 1$ *for all i.*

This tie-breaking ensures asymptotic optimality of the index policy as it enforces that the fraction of sub-processes in each state s is equal to the distribution induced by π^{**} in the limit. This idea shall become clear in Section 2.4 where the proof of asymptotic optimality is presented.

We formally present our index policy in Algorithms 1 and 2.

2.4 Proof of Asymptotic Optimality

Our index policy $\hat{\boldsymbol{\pi}}$ achieves asymptotic optimality when we let the number of sub-processes K go to infinity, while holding $\alpha_t = \frac{m_t}{K}$ constant for all t. Let $Z(\boldsymbol{\pi}, \mathbf{m}, K)$ to denote the expected reward of the original MDP obtained by policy $\boldsymbol{\pi}$ with K sub-processes and $\mathbf{m} = (m_1, ..., m_T)$ constraints at each time. We use $\boldsymbol{\Pi}_{\mathbf{m},K}$ to denote the set of all feasible Markov policies for the such an MDP. Lastly, it should be understood that whenever we use $\hat{\boldsymbol{\pi}}$ to denote our index policy there is a dependency of $\hat{\boldsymbol{\pi}}$ on $\mathbf{m}$ and K that is not explicitly stated. We are now ready to state the main result of this chapter,

Algorithm 1: Index Policy $\hat{\boldsymbol{\pi}}$

Pre-compute: $\boldsymbol{\lambda}^*; \boldsymbol{\beta}; \rho^*$. (Refer to section 2.3.1 for computational details)

for $t = 1, ..., T$ **do**

Let $\beta_{t,[i]}$ be the i^{th} largest element in the list $\beta_t(S_{t,1}), ..., \beta_t(S_{t,K})$, so $\beta_{t,[1]} \geq \ldots \geq \beta_{t,[K]}$.

Let $\bar{\beta}_t = \beta_{t,[m]}$

Let $I_t = \{s : \beta_t(s) = \bar{\beta}_t \text{ and } s = S_{t,x} \text{ for some x}\}$

Let $N_t(s) = |\{x : S_{t,x} = s\}|$, for all s.

For $s \in I_t$, let

$$q_t(s) = \begin{cases} \frac{\rho^*(s,1,t)}{\sum_{s' \in I_t} \rho^*(s',1,t)}, \text{ if } \sum_{s' \in I_t} \rho^*(s',1,t) > 0 \\ \frac{N_t(s)}{\sum_{s' \in I_t} N_t(s')}, \text{ otherwise} \end{cases}$$

Let $b = \text{Rounding}(m - \sum_{s':\beta_t(s') > \bar{\beta}_t} N_t(s'), (q_t(s) : s \in I_t), (N_t(s) : s \in I_t))$

for all s **do**

If $\beta_t(s) > \bar{\beta}_t$, set all $N_t(s)$ sub-processes in s active.

If $\beta_t(s) = \bar{\beta}_t$, set $b(s)$ sub-processes in s active.

If $\beta_t(s) < \bar{\beta}_t$, set 0 sub-processes in s active.

end for

end for

which shows that the per arm gap between the upper bound and the index policy goes to zero under the limit assumption.:

Theorem 1. *For any* $\boldsymbol{\alpha} \in (0,1)^T$,

$$\lim_{K \to \infty} \frac{1}{K} \left(Z(\hat{\boldsymbol{\pi}}, \lfloor \boldsymbol{\alpha} K \rfloor, K) - \max_{\boldsymbol{\pi} \in \boldsymbol{\Pi}_{\lfloor \alpha K \rfloor, K}} Z(\boldsymbol{\pi}, \lfloor \boldsymbol{\alpha} K \rfloor, K) \right) = 0, \tag{2.10}$$

where $\lfloor \boldsymbol{\alpha} K \rfloor = (\lfloor \alpha_1 K \rfloor, ..., \lfloor \alpha_T K \rfloor)$

Algorithm 2: Rounding(total, frac, avail)

Inputs: total (a scalar), frac (a vector satisfying $\sum_i \text{frac}_i = 1$), avail (a vector of the same length as frac satisfying total $\leq \sum_i \text{avail}_i$)

Output: b (a vector of the same length as the inputs satisfying $\sum_i b_i = \text{total}, b_i \leq \text{avail}_i$)

Let n = length(frac)

Let $\text{b}_i = \min\{\text{avail}_i, \lfloor \text{total} * \text{frac}_i \rfloor\}$, for $i = 1, ..., n$.

Let $j = 1$

while total $> \sum_{i=1}^{n} \text{b}_i$ **do**

  Let $\text{b}_j = \text{b}_j + \mathbb{1}(\text{avail}_j > b_j)$

  Let $j = (j \mod n) + 1$

end while

return b

We first point out that the optimal solutions of $\max_{\boldsymbol{\pi} \in \boldsymbol{\Pi}_{\lfloor \alpha K \rfloor, K}} Z(\boldsymbol{\pi}, \lfloor \boldsymbol{\alpha} K \rfloor, K)$ are trivial when $\alpha = \mathbf{0}$ or $\mathbf{1}$. So we do not include these two cases when we consider convergence of the index policy $\hat{\boldsymbol{\pi}}$. To formalize the notations that will be used throughout the proofs, we augment $P(\boldsymbol{\lambda})$ to $P(\boldsymbol{\lambda}, \mathbf{m}, K)$ to indicate the values of $\mathbf{m}$ and K assumed in the Lagrangian relaxation. We use $\boldsymbol{\lambda}^*$ to denote one and any element in $\arg\inf_{\boldsymbol{\lambda}} P(K, \boldsymbol{\alpha} K, \boldsymbol{\lambda})$ and let π^{**} be the optimal policy constructed in (2.8) using $m_t = \alpha_t K$, which satisfies $\mathbb{E}^{\pi^{**}}(A_t) = \alpha_t$ for all t. Note $\boldsymbol{\lambda}^*$ and π^{**} depend on only $\boldsymbol{\alpha}$ (not on K).

As before, we let $N_t(s)$ be the number of sub-processes in state s at time t under $\hat{\boldsymbol{\pi}}$. We additionally define $M_t(s)$ to be the number of sub-processes in state s at time t that are set active by $\hat{\boldsymbol{\pi}}$. These quantities depend on K and $\mathbf{m}$, but for simplicity we do not include this dependence in the notation. We always assume $\mathbf{m} = \lfloor \boldsymbol{\alpha} K \rfloor$ and we rely on context to make clear the value of K assumed. We also define $V_t(s)$ to be the set of states with the same index value as s, including s, and $U_t(s)$ to be the set of states with index

value greater than that of s, for each time t. These quantities depend on $\boldsymbol{\alpha}$ but not on K or $\mathbf{m}$.

We prove Theorem 1 by first demonstrating below in Theorem 2 that for each time t, the proportion of the sub-processes that are in state s under our index policy $\hat{\boldsymbol{\pi}}$, $\frac{N_t(s)}{K}$, approaches $P_t(s)$ as $K \to \infty$. In other words, our index policy $\hat{\boldsymbol{\pi}}$ recreates the behavior of π^{**} in the large K limit.

Theorem 2. *For every $s \in \mathbb{S}$ and $1 \leq t \leq T$,*

$$\lim_{K\to\infty} \frac{N_t(s)}{K} = P_t(s), \quad P^{\hat{\boldsymbol{\pi}}} - a.s., \tag{2.11}$$

and

$$\lim_{K\to\infty} \frac{M_t(s)}{K} = P_t(s) * \pi^{**}(s,1,t), \quad P^{\hat{\boldsymbol{\pi}}} - a.s., \tag{2.12}$$

Before proving Theorem 2, we first present two intermediate results, whose proofs are given in Appendix A.8 and A.9.

Lemma 3. *At time $1 \leq t \leq T$, for all $s \in \mathbb{S}$, we have*

(1) If $\beta_t(s) > \lambda_t^*$, then $\pi^{**}(s,1,t) = 1$.

(2) If $\beta_t(s) < \lambda_t^*$, then $\pi^{**}(s,1,t) = 0$.

Lemma 4. *For any state $s \in \mathbb{S}$ and time $1 \leq t \leq T$,*

(1) If $\alpha_t - \sum_{s' \in U_t(s) \cup V_t(s)} P_t(s') \geq 0$, then $\pi^{**}(s,1,t) = 1$.

(2) If $\alpha_t - \sum_{s' \in U_t(s)} P_t(s') \leq 0$, then $\pi^{**}(s,1,t) = 0$.

Now we are ready to prove Theorem 2.

Proof. We prove (2.11) and (2.12) simultaneously via induction over the time periods.

When $t = 1$, all sub-processes starts in state s_1, and we have

$$\lim_{K\to\infty} \frac{N_1(s)}{K} = \lim_{K\to\infty} \frac{K}{K} = 1 = P_1(s) \text{ if } s = s_1,$$
$$\lim_{K\to\infty} \frac{N_1(s)}{K} = \lim_{K\to\infty} \frac{0}{K} = 0 = P_1(s) \text{ otherwise.}$$

By the set-up of the original MDP, $M_1(s) = \lfloor \alpha * K \rfloor$, and we have

$$\lim_{K\to\infty} \frac{M_1(s)}{K} = \lim_{K\to\infty} \frac{\lfloor \alpha * K \rfloor}{K} = \alpha = \pi^{**}(s, 1, t) = P_1(s) * \pi^{**}(s, 1, t), \text{ if } s = s_1,$$
$$\lim_{K\to\infty} \frac{M_1(s)}{K} = \frac{0}{K} = 0 = P_1(s) * \pi^{**}(s, 1, t), \text{ otherwise,}$$

so we have proved the base case of the induction.

Now assume (2.11) and (2.12) hold up until time t. Fix a state $s \in \mathbb{S}$ and time $1 \le t \le T$, define $Y_t(s', s)$ to be the number of sub-processes set active by $\hat{\boldsymbol{\pi}}$ in s' at time t which transition to state s at time $t + 1$, and $X_t(s', s)$ to be the number of sub-processes set inactive by $\hat{\boldsymbol{\pi}}$ in s' at time t which transition to s at time $t + 1$. Note that $Y_t(s', s)$ and $X_t(s', s)$ also depend on K. We can subsequently express $N_{t+1}(s)$ as

$$N_{t+1}(s) = \sum_{s'\in\mathbb{S}} Y_t(s', s) + X_t(s', s).$$

Dividing both sides by K, and taking K to a limit, we get

$$\lim_{K\to\infty} \frac{N_{t+1}(s)}{K} = \lim_{K\to\infty} \sum_{s'\in\mathbb{S}} \frac{1}{K} Y_t(s', s) + \lim_{K\to\infty} \sum_{s'\in\mathbb{S}} \frac{1}{K} X_t(s', s). \tag{2.13}$$

Note $Y_t(s', s)$ is a binomial random variable with $M_t(s')$ trials and success probability $P^1(s', s)$. Similarly, $X_t(s', s)$ is a binomial random variable with $N_t(s') - M_t(s')$ trials and success probability $P^0(s', s)$. We can rewrite the RHS of (2.13) by applying Lemma 14,

which is stated in Appendix A.10:

$$\lim_{K\to\infty}\frac{N_{t+1}(s)}{K} = \sum_{s'\in\mathbb{S}}\lim_{K\to\infty}\frac{M_t(s')}{K} * P^1(s',s) \tag{2.14}$$

$$+\sum_{s'\in\mathbb{S}}\lim_{K\to\infty}\frac{N_t(s')-M_t(s')}{K} * \mathbb{P}^0_x(s',s)$$

$$= \sum_{s'\in\mathbb{S}} P_t(s') * \pi^{**}(s',1,t+1) * P^1(s',s) \tag{2.15}$$

$$+\sum_{s'\in\mathbb{S}} P_t(s')(1-\pi^{**}(s',1,t+1)) * \mathbb{P}^0_x(s',s)\ \ a.s.$$

$$= P_{t+1}(s).\ \ a.s. \tag{2.16}$$

The last equality follows as we have exhausted all the ways of getting to s at time $t+1$. Hence we have shown (2.11) holds for time $t+1$.

Next we show (2.12) holds for time $t+1$. We define sets $\mathbf{P}_t = \{P_t(s) : s \in \mathbb{S}\}$, and $\mathbf{N}_t = \{N_t(s) : s \in \mathbb{S}\}$. Recall $V_t(s)$ is the set of states with the same index value as s, including s, and $U_t(s)$ is the set of states with index value greater than that of s, we let $N_t^+(s) = \sum_{s'\in U_t(s)} N_t(s')$ and $N_t^=(s) = \sum_{s'\in V_t(s)} N_t(s')$. We use notation $\frac{\mathbf{N}_t}{K}$ for the set which consists of all elements in $\mathbf{N}_t$ divided by K. Lastly we use the function $f_s(\mathbf{N}_t, \lfloor\alpha_t K\rfloor)$ to represent the number of sub-processes set active at time t in state s:

$$f_s(\mathbf{N}_t, \lfloor\alpha_t K\rfloor) = \mathbb{1}_{([\lfloor\alpha_t K\rfloor - N_t^+(s)]^+ \geq N_t^=(s))} * N_t(s)$$

$$+\ \mathbb{1}_{([\lfloor\alpha_t K\rfloor - N_t^+(s)]^+ < N_t^=(s))} * b_s(\mathbf{N}_t, \lfloor\alpha_t K\rfloor), \tag{2.17}$$

where $b_s(\mathbf{N}_t, \lfloor\alpha_t K\rfloor)$ is number of sub-processes to set active as output by *Rounding* in Algorithm 2. The first indicator represents situations where tie-breaking is not needed and all the sub-processes are set active. The second indicator represents situations where tie-breaking is needed and is determined by the function Rounding, and situations where no tie-breaking is needed and no sub-process is set active.

To facilitate our proof, we consider a continuous version of the function Rounding

and denote it as *Rounding-c*. Rounding-c first distributes $\min\{\text{total} * \text{frac}_i, \text{avail}_i\}$, instead of $\min\{\lfloor \text{total} * \text{frac}_i \rfloor, \text{avail}_i\}$ in Rounding, to each state, and uses a fluid way to distribute the difference between *total* and the amount distributed initially. Rounding-c is given with full detail in Algorithm 3. We use $\bar{b}_s(\mathbf{N}_t, \lfloor \alpha_t K \rfloor)$ to denote the output of Rounding-c.

Algorithm 3: Rounding-c(total, frac, avail)

Inputs: total (a scalar), frac (a vector satisfying $\sum_i \text{frac}_i = 1$), avail (a vector of the same length as frac satisfying $\text{total} \leq \sum_i \text{avail}_i$)

Output: b (a vector of the same length as the inputs satisfying $\sum_i b_i = \text{total}, b_i \leq \text{avail}_i$)

Let $n = \text{length(frac)}$

Let $\text{b}_i = \min\{\text{avail}_i, \text{total} * \text{frac}_i\}$, for $i = 1, ..., n$.

Let $L = \{i : 1 \leq i \leq n, \text{b}_i < \text{avail}_i\}$

while $\text{total} > \sum_{i=1}^{n} \text{b}_i$ **do**

Let $t = \max\{t \geq 0 : \text{b}_i + \frac{t}{|L|} \leq \text{avail}_i, \ \forall i \in L \text{ and } \sum_{i=1}^{n}(\text{b}_i + \frac{t}{|L|}) \leq \text{total}\}$

Let $\text{b} = \{b_i + \mathbb{1}_{(i \in L)} \frac{t}{|L|} : 1 \leq i \leq n\}$

Let $L = \{i : 1 \leq i \leq n, \text{b}_i < \text{avail}_i\}$

end while

return b

Moreover, we use

$$\bar{f}_s(\mathbf{N}_t, \lfloor \alpha_t K \rfloor) = \mathbb{1}_{([\lfloor \alpha_t K \rfloor - N_t^+(s)]^+ \geq N_t^=(s))} * N_t(s) + \mathbb{1}_{([\lfloor \alpha_t K \rfloor - N_t^+(s)]^+ < N_t^=(s))} * \bar{b}_s(\mathbf{N}_t, \lfloor \alpha_t K \rfloor), \tag{2.18}$$

to denote the number of sub-processes set active in state s at time t according to this continuous tie-breaking rule Rounding-c.

This proof will be accomplished by the following three lemmas, whose proofs are

given in Appendices A.11, A.12, A.13

Lemma 5.

$$\left| f_s(\mathbf{N}_t, \lfloor \alpha_t K \rfloor) - K \bar{f}_s \left(\frac{\mathbf{N}_t}{K}, \frac{\lfloor \alpha_t K \rfloor)}{K} \right) \right| \leq 2, \quad \forall 1 \leq t \leq T$$

.

Lemma 6.

$$\lim_{K \to \infty} \bar{f}_s \left(\frac{\mathbf{N}_t}{K}, \frac{\lfloor \alpha_t K \rfloor}{K} \right) = \bar{f}_s(\mathbf{P}_t, \alpha_t), a.s, \quad \forall 1 \leq t \leq T.$$

Lemma 7.

$$\bar{f}_s(\mathbf{P}_t, \alpha_t) = P_t(s) \pi^{**}(s, 1, t), \quad \forall 1 \leq t \leq T.$$

Combining the three lemmas above we have

$$\lim_{K \to \infty} \frac{M_{t+1}(s)}{K} = \lim_{K \to \infty} \frac{f_s(\mathbf{N}_{t+1}, \lfloor \alpha_{t+1} K \rfloor)}{K} = \lim_{K \to \infty} \bar{f}_s \left(\frac{\mathbf{N}_{t+1}}{K}, \frac{\lfloor \alpha_{t+1} K \rfloor}{K} \right) = P_{t+1}(s) \pi^{**}(s, 1, t+1),$$

□

Finally, we prove Theorem 1 by leveraging the results from Theorem 2.

Proof of Theorem 1. $\hat{\boldsymbol{\pi}} \in \boldsymbol{\Pi}_{\lfloor \boldsymbol{\alpha} K \rfloor, K}$ implies $Z(\hat{\boldsymbol{\pi}}, \lfloor \boldsymbol{\alpha} K \rfloor, K) \leq \max_{\boldsymbol{\pi} \in \boldsymbol{\Pi}_{\lfloor \boldsymbol{\alpha} K \rfloor, K}} Z(\boldsymbol{\pi}, \lfloor \boldsymbol{\alpha} K \rfloor, K)$. Thus,

$$\lim_{K \to \infty} \frac{1}{K} Z(\hat{\boldsymbol{\pi}}, \lfloor \boldsymbol{\alpha} K \rfloor, K) \leq \lim_{K \to \infty} \frac{1}{K} \sup_{\boldsymbol{\pi} \in \boldsymbol{\Pi}_{\lfloor \boldsymbol{\alpha} K \rfloor, K}} Z(\boldsymbol{\pi}, \lfloor \boldsymbol{\alpha} K \rfloor, K).$$

On the other hand,

$$\begin{aligned}
\lim_{K\to\infty}\frac{1}{K}Z(\hat{\boldsymbol{\pi}},\lfloor\boldsymbol{\alpha}K\rfloor,K) &= \lim_{K\to\infty}\frac{1}{K}\mathbb{E}^{\hat{\boldsymbol{\pi}}}\left[\sum_{t=1}^{T}\sum_{s\in\mathbb{S}} r_t(s,1)M_t(s) + r_t(s,0)(N_t(s)-M_t(s))\right] \\
&= \sum_{t=1}^{T}\sum_{s\in\mathbb{S}} r_t(s,1)\lim_{K\to\infty}\frac{1}{K}\mathbb{E}^{\hat{\boldsymbol{\pi}}}\left[M_t(s)\right] + \\
&\qquad r_t(s,0)\lim_{K\to\infty}\frac{1}{K}\mathbb{E}^{\hat{\boldsymbol{\pi}}}\left[N_t(s)-M_t(s)\right] \\
&= \sum_{t=1}^{T}\sum_{s\in\mathbb{S}}\left[r_t(s,1)\rho(s,1,t) + r_t(s,0)\rho(s,0,t)\right] \\
&= \sum_{t=1}^{T}\sum_{s\in\mathbb{S}}\left[r_t(s,1)\rho(s,1,t) + r_t(s,0)\rho(s,0,t)\right] \\
&\qquad - \mathbb{E}^{\pi^{**}}\left[\sum_{t}\lambda_t\left(A_t-\alpha_t\right)\right] \\
&= Q(\boldsymbol{\lambda}^*) + \sum\lambda_t^*\alpha_t \\
&= \lim_{K\to\infty}\frac{1}{K}(KQ(\boldsymbol{\lambda}^*) + \lfloor\boldsymbol{\alpha}K\rfloor\sum\lambda_t^*) \\
&= \lim_{K\to\infty}\frac{1}{K}P(\boldsymbol{\lambda}^*,\lfloor\boldsymbol{\alpha}K\rfloor,K) \\
&\geq \lim_{K\to\infty}\frac{1}{K}\sup_{\boldsymbol{\pi}\in\boldsymbol{\Pi}_{\lfloor\boldsymbol{\alpha}K\rfloor,K}} Z(\boldsymbol{\pi},\lfloor\boldsymbol{\alpha}K\rfloor,K).
\end{aligned}$$

Here, the third line follows by Theorem 2 and the fact that both $N_t(s)$ and $M_t(s)$ are bounded and hence uniformly integrable random variables (for uniformly integrable random variables, convergence almost surely implies convergence in expectation). The fourth line holds because π^{**} takes the active action at each time with probability α. The fifth line follows from Lemma 2, where we have augmented the notation for P to include the values of m and K assumed. The sixth line follows from Lemma 1.

Finally, sandwiching the two inequalities gives the desired result. □ □

2.5 Numerical Experiments

In this section we present numerical experiments for two problems: the finite-horizon multi-arm bandit with multiple pulls per period,and subset selection [9, 33]. These experiments demonstrate numerically that our index policy is indeed asymptotically optimal. We also compare the finite-time performance of our policy to other policies from the literature. Although our previously provided theoretical results do not apply to finite K, we see that our index policy performs strictly better than all benchmarks considered in both of the problems.

2.5.1 Multi-armed bandit

In our first experiment, we consider a Bernoulli multi-armed bandit problem with a finite time horizon $T = 6$, and multiple pulls per time period. A player is presented with K arms and may select $m = \lfloor K/3 \rfloor$ of them to pull at every time st. Each arm pulled returns a reward of 0 or 1. The player's goal is to maximize her total expected reward. We assume that each arm returns i.i.d rewards according to a Bernoulli distribution with an unknown rate of success θ_x. We take a Bayesian approach and impose a Beta(1,1) prior on each of the θ_x. Note that the posterior distributions of θ_x are still going to be beta-distributed. The values of the state then correspond to the posterior parameters of the K arms.

For comparison, we include results from an upper confidence bound (UCB) algorithm with pre-trained confidence width. At every time step, we compute $\mu_i + \alpha * \delta_i$ for each arm i, where μ_i and δ_i are the sample mean and standard deviation of arm i. We pre-train α by running the UCB algorithm on a different set of data (but simulated with

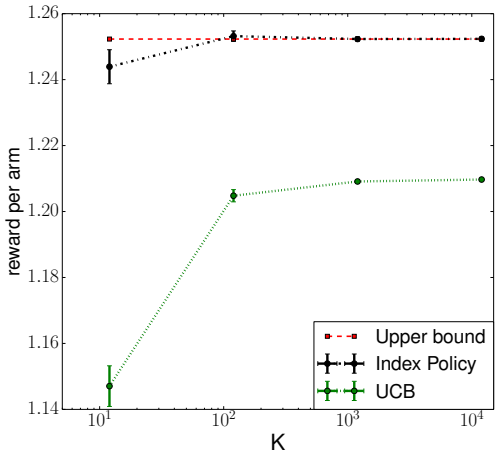

Figure 2.1: Upper bound and simulation results of MAB

the same distribution) with values of α ranging from 0 to 5 with a step size of 0.1 and then set α to the value that gives the best performance.

Figure 2.5.1 plots the reward per arm (expected total reward divided by K) against K, for $K = 12, 120, 1200, 12000$. The red dashed line represents the upper bound computed using $P(\boldsymbol{\lambda}^*)$. For each policy, circles show the sample mean of the total reward per arm, and vertical bars indicate a 95% confidence interval for the expected total reward per arm. The UCB policy's sample means are connected by a dashed green line, and the index policy's are connected by a dashed black line. Values are calculated using 5000 replications.

The index policy consistently outperforms the UCB policy. As K grows large, the confidence interval for the index policy's total performance per arm overlaps with the upper bound, which numerically attests to the accuracy of Theorem 1 and illustrates the rate of convergence.

2.5.2 Project assignment problem

In our second experiment, we consider the following finite-horizon restless bandit problem: suppose a team has K ongoing projects and $m = \lfloor K/5 \rfloor$ engineers who can work on any single project at a time. The manager of the team decides which projects to prioritize on a weekly basis. The state space of each project is the set of positive integers. Every project starts at the state of 1. The state of a project can either remain unchanged or can increment by 1 as we move to the next week. When an engineer works on a project in state s for a week, it moves to the next state with probability $p_1(s)$; otherwise, it transitions to the next state with probability $p_0(s)$. Each project can be worked on by at most 1 engineer. When a transition happens, the team collects a reward which is a function of the current state. All the states have the same reward 1 except the final state which has a reward of 50. The manager's goal is to maximize the total expected reward over a horizon of $T = 5$ weeks. Here we set $p_1(s) = 0.9 - 0.6s/T$ for $s < T$, and $p_1(s) = 0.05$ for $s = T$. We set $p_2 = 0.16p_1$.

For comparison we use Whittle's index policy [48] and a randomized policy that selects m projects randomly at every time step. To apply Whittle's index policy which is for infinite horizon setting, we convert the problem to a infinite horizon problem by adding an absorbing state for time T onwards, and make all the state at time T transit to this absorbing state with probability 1.

Figure 2.5.2 presents the outcome of the experiment at $K = 10, 100, 1000, 10000$. Again we plot the number of projects against the per arm reward. We use red dashed line to represent the upper bound. The green dashed line shows the result of the randomized policy, which perform significantly worse than the other two. The index policy is represented by the black dashed line and the Whittle's index policy is represented by the blue dashed line. The index policy out-performs the Whittle's policy and converges

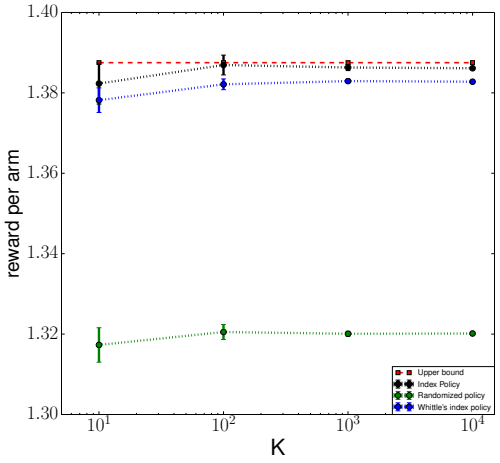

Figure 2.2: Upper bound and simulation result of project assignment problem

to the upper bound. Whittle's policy is sub-optimal as it considers average case over time. Hence it tends to allocate more resources to projects which are in state 5 which has a very low probability of getting a large reward, while the optimal thing to do is to allocate resource to arms that are that a state with lower value but has a much higher probability of obtaining some reward.

2.5.3 Subset selection problem

In the third experiment, we consider a subset selection problem in ranking and selection whose goal is to identify m best designs out of K designs, each with some underlying distribution θ_x. This problem is considered in [9] as well as [33]. We assume $\bar{m}$ parallel computing resources are available; at each time step we select $\bar{m}$ out of K design to evaluate. After T rounds of evaluation, we select m best designs. In this numerical study, we set $T = 4$, $\frac{m}{K} = 0.3$ and $\frac{\bar{m}}{K} = 0.5$. We consider the situation when the outcomes of evaluation are binary, but note that our model can handle any real-valued outcomes.

Below is how we formulate this problem as an RMAB:

$$\begin{aligned} \underset{\boldsymbol{\pi}\in\boldsymbol{\Pi}}{\text{maximize}} \quad & \mathbb{E}^{\boldsymbol{\pi}}\left[\sum_{t=1}^{T+1} R_t(\mathbf{S}_t, \mathbf{A}_t)\right] \\ \text{subject to} \quad & P^{\boldsymbol{\pi}}(|\mathbf{A}_t| = m) = 1, \quad \text{for } t = T+1, \\ & P^{\boldsymbol{\pi}}(|\mathbf{A}_t| = \bar{m}) = 1, \quad \text{for } 1 \le t \le T, \end{aligned} \tag{2.19}$$

where $R_t(\mathbf{S}_t, \mathbf{A}_t) = 0$ when $t \le T$, and $R_t(\mathbf{S}_t, \mathbf{A}_t) = \sum_{x=1}^{K} \mathbb{E}[\theta_x | \mathbf{S}_{t,x}]$ when $t = T+1$. We start with a uniform prior for each design.

We compare the performance of our policy against the *OCBA-m* selection procedure proposed in [9]. Since [9] considers a slightly different setting in which a policy maker can evaluate a design more than once in a time step, we modify the procedure slightly to fit our setup: instead of sampling according to the number of times dictated by the algorithm, we rank the designs by their desired number of samples, and simulate the first $\bar{m}$ of them. Moreover, we assign a positive sample and a negative sample to each of the design at the beginning of the simulation so that it starts with the same amount of information as our Bayesian setup. We also use the UCB policy as a standard of comparison. The implementation of the UCB policy is similar to the one in section 2.5.1.

The simulation results show that all the three policies perform similarly when K is small, with the OCBA-m policy having a slight edge for $K = 10$. From $K = 100$ onwards, the index policy consistently outperforms the other two. In addition, the gap between the upper bound and the index policy vanishes as K becomes large, while the gaps between the upper bound and the other two policies remain constant.

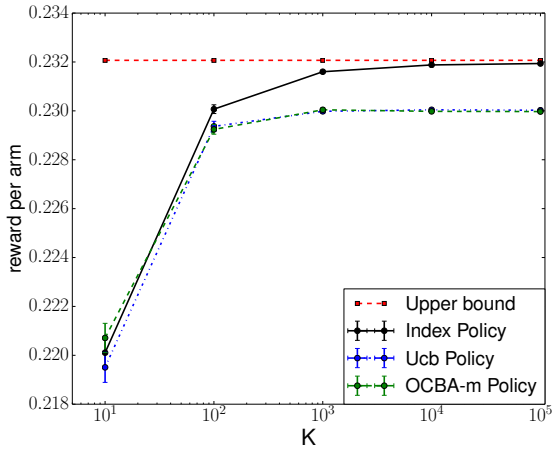

Figure 2.3: Upper bound and simulation result of subset selection

2.6 Conclusion

In this chapter we propose an index-based policy for finite horizon RMAB problems that is computational tractable, and prove that it is asymptotically optimal in the same limit as considered by Whittle. We also show that the numerical performance of this index-based policy beats the state-of-art. For future work, we conjecture that our results, including the formulation of the policy and the asymptotic optimality, can be extended to the following situations:

1. Multiple actions associated with a state, instead of an active and a passive action in the current formulation;
2. A total budget constraint over the entire time horizon, in addition to a budget constraint at every time step.
3. Infinite state space.

CHAPTER 3

PARALLEL BAYESIAN POLICIES FOR FINITE MULTIPLE COMPARISONS WITH A KNOWN STANDARD

In this chapter, we consider an applied problem in simulation optimization: the multiple comparisons with a known standard (MCS). The problem is similar to an RMAB problem considered in Chapter 2 except that it allows multiple pulls per arm at a time. In this problem, one wishes to use simulation to determine, for each among a finite pool of simulating systems, which ones have an expected output measure that exceeds a known threshold. In this chapter, we use Bayesian statistics and dynamic programming to study how one should allocate simulate effort in the MCS problem, so as to best support this final determination. This chapter is organized as follows: In Section 3.1, we do a light motivation and literature review; In Section 3.2 we formally state the problem; In Section 3.3, we provide a computationally tractable upper bound on the value of a Bayes-optimal procedure, and a method for computing this upper bound; In Section 3.4 we present a heuristic motivated by this upper bound, which is similar to the index-based policy proposed in Chapter 2 for RMAB, with the exception that it allows assignment of multiple resources to a system at a time; In Section 3.5 we present numerical results in which we demonstrate that the index policy performs close to an optimal policy; Lastly we present our numerical results and conclusion in Sections 3.5 and 3.6.

3.1 Introduction

The MCS problem arises in at least two distinct ways in simulation applications. First, it arises when determining which options perform better than some standard option whose performance is so well-estimated that it can be treated as known [36]. Second, it arises

when determining which options have a secondary performance measure that satisfies a constraint [2]. The MCS problem also arises outside of simulation, in crowdsourcing service centers like Amazon's Mechanical Turk, when allocating a budget across workers who label items (e.g., images, documents), to best support accurate classification of these items [12].

We consider a variant of the MCS problem in which parallel computing resources are available. The growing availability of parallel computing resources presents new opportunities to perform simulation analysis at larger scales, but also imposes constraints on the way simulation effort is allocated. In our model, simulation effort is allocated batch-sequentially: simulations are performed in batches, and we decide how many additional replications to perform for each system at the start of each batch based on the results of previous batches. We are given a fixed budget, specified as a number of parallel computing resources and a number of batches, and our goal is to allocate these batches of simulation efficiently, so as to best allow correct classification of the systems once our simulation budget is exhausted.

We formulate the MCS problem in a Bayesian framework, and we measure the performance of a batch-sequential procedure by its average case performance, averaging across problem instances drawn from the prior and across simulation noise. While the Bayes-optimal procedure is characterized by the dynamic programming equations [19], the curse of dimensionality makes solving this dynamic program computationally intractable for problems with many systems.

Rather than solving this dynamic program exactly, we provide a computationally tractable upper bound on its value. This allows us to evaluate the quality of sub-optimal heuristic policies relative to this upper bound. This provides guidance to the development and improvement of heuristic policies, in the form of information about the opti-

mality gap. The analysis technique used in this upper bound is a Lagrangian relaxation on the total number of simulations performed in any given batch. Using the Lagrange multipliers obtained from this relaxation, we also develop a heuristic policy, and use numerical experiments to demonstrate that it performs close to the upper bound on optimal in the problem setting studied.

This chapter builds on the previous work [49], which also considered the Bayesian MCS problem. That chapter considered the sequential setting, without parallel resources, and provided a computationally efficient method for computing the Bayes-optimal policy under two assumptions about limitations of sampling: that there is a time horizon that is random and exponentially distributed; or there is no time horizon, and we pay a fixed cost for each sample. Our current work differs from that work by considering parallelism, and by considering a fixed budget. While the infinite-horizon fixed-cost-per-sample model in [49] is quite natural for cloud computing settings, and the random exponentially distributed horizon is attractive for its computational tractability, using a fixed horizon is more natural than either model in [49] when allocating computing resources that are owned rather than rented. While we focus on the parallel setting and [49] focused on the sequential setting, our work can also provide an upper bound for the sequential setting with fixed horizon by setting the number of parallel nodes to 1.

For simplicity in this chapter, we consider only Bernoulli samples, with a linear loss function. However, the techniques developed in this chapter should also be adaptable to other parametric sampling distributions with conjugate priors, and other loss functions.

Our model assumes synchronous computations, in which we wait for all simulations in a batch to complete before starting the next batch. This approach is reasonable when the variability in the time to simulate a system is small enough to allow waiting until simulations finish before starting the next batch. Such assumptions are more commonly

met in controlled high-performance computing environments, and are less common in cloud computing environments. If computation time is highly variable, it may be more appropriate to model computation as asynchronous.

Our Lagrangian relaxation of a budget constraint in the MCS problem is related to Lagrangian relaxations in restless multi-armed bandit problems [48, 23]. Indeed, if one added an additional constraint that each system can be simulated at most once in any batch, then our MCS problem can be reformulated as a restless multi-armed bandit problem where we can pull multiple arms in each round: in this restless multi-armed bandit problem, each system is an arm, pulling the arm corresponds to simulating that system; the reward from this pull is the improvement in our ability to classify this system; and the number of arms we can pull in a round is the number of parallel resources. The restless multi-armed bandit framework of [48] is most well-known for allowing bandit arms to evolve when they are not pulled, but also allows multiple pulls per round. Our analysis can be seen as examining a generalization of restless bandits in which each arm can be pulled multiple times per round, and our heuristic policy can be seen as a generalization of the Whittle index policy to this setting.

Our use of a Lagrangian relaxation to study the Bayesian formulation of the MCS problem is also similar to [50], which used a Lagrangian relaxation to bound the value of the Bayes-optimal procedure for the ranking and selection problem.

While we consider the MCS problem in the Bayesian setting, much of the previous work on the MCS problem has considered non-Bayesian settings. This previous work includes the one-stage procedures by [37, 16], the two-stage procedures by [15, 7, 13], and work on indifference-zone ranking and selection (see the survey [32]). The more recent frequentist work includes [5] which provides a fully sequential procedure under stochastic constraint, and [26] which further allows correlation across systems.

In the version of the MCS problem that we consider, we emphasize that our standard has known value, and we seek to determine only whether each system is better or worse than this standard. We do not consider standards with unknown value, produce joint confidence intervals, nor select the best among those systems performing better than standard. This is in contrast with much previous work on multiple comparisons [36, 31]. The variant of the MCS problem that we consider has also been called *feasibility determination* [44].

3.2 Problem Formulation

We have k systems, each of which can be simulated using a stochastic simulation. When we simulate system $x \in \{1, \ldots, k\}$, we observe a $Bernoulli(\theta_x)$ random variable, indicating the system's performance in that simulation. We will think of an outcome of 1 as indicating the system succeeded in that simulation, and 0 as indicating failure. Although we expect that the methods we develop in this paper can be extended to simulations that generate non-Bernoulli samples, we focus on the Bernoulli setting here for simplicity. The sampling means θ_x are initially unknown, and we wish to determine through simulation, for each system x, whether θ_x is greater than some known threshold d_x. This threshold may differ across systems.

We adopt a Bayesian formulation, in which we seek to do well on average with respect to a prior probability distribution over the unknown sampling means $\theta_1, \ldots, \theta_k$. We consider Bayesian prior probability distributions under which

$$\theta_x \sim \text{Beta}(\alpha_{0,x}, \beta_{0,x}),$$

with independence across x, for some given values $\alpha_{0,x}, \beta_{0,x}$. We assume a Beta prior for tractability: the Beta prior is conjugate to the Bernoulli likelihood [14], and so this

assumption allows our posterior distribution to remain Beta-distributed.

We perform N batches of simulations, performing at most m simulations in parallel in each batch. At the start of each batch $n = 1, \ldots, N$, we choose the number of samples $z_{n,x}$ to take from each system x, making sure to satisfy the constraint $\sum_x z_{n,x} \leq m$. This choice of $z_{n,x}$ may depend upon the results observed from all previous batches. We then observe the number of successes from each system x, which are conditionally binomial,

$$Y_{n,x}|\theta_x, z_{n,x} \sim \text{Binomial}(z_{n,x}, \theta_x).$$

We assume conditional independence of $Y_{n,x}$ across x and from all previous samples, given θ_x and $z_{n,x}$. This precludes the use of common random numbers, but is satisfied when using independent sampling. After observing this $Y_{n,x}$, we update our posterior distribution on θ_x to obtain [14]

$$\theta_x|z_{1,x}, Y_{1,x}, \ldots, z_{n,x}, Y_{n,x} \sim \text{Beta}(\alpha_{n,x}, \beta_{n,x}),$$

where $\alpha_{n,x} = \alpha_{0,x} + \sum_{n' \leq n} Y_{n',x}$ can be interpreted as the effective number of successes from system x, and $\beta_{n,x} = \beta_{0,x} + \sum_{n' \leq n}(z_{n',x} - Y_{n',x})$ as the effective number of failures. This posterior is independent across x.

We use dynamic programming to analyze this problem. To support this, for each x and n, we define the state variable $S_{n,x} = (\alpha_{n,x}, \beta_{n,x})$. To streamline the discussion, we define $\boldsymbol{\alpha}_n = (\alpha_{n,1}, ..., \alpha_{n,k})$, $\boldsymbol{\beta}_n = (\beta_{n,1}, .., \beta_{n,k})$, $\mathbf{S}_n = (S_{n,1}, ..., S_{n,k})$, and $\mathbf{z}_n = (z_{n,1}, \ldots, z_{n,k})$. Let Λ_n be the space in which $\mathbf{S}_n$ takes values, $\Lambda_n = \left\{(\alpha_{0,1} + s_1, \beta_{0,1} + f_1, \ldots, \alpha_{0,k} + s_k, \beta_{0,k} + f_k) : s_x, f_x \in \mathbb{Z}_+ \forall x, \ \sum_{x=1}^{k} s_x + f_x \leq mn\right\}$.

We stop sampling after batch N and decide, for each system x, whether to label θ_x as above or below the threshold d_x. If we label it as above, we receive a reward of $\theta_x - d_x$. Otherwise, we receive $d_x - \theta_x$. The total terminal reward is the sum of these rewards across the systems. As shown in [49], to maximize the conditional expected value of this

reward given what we know after our simulations are complete (which is summarized in $S_{N,x}$), we should choose to receive $\theta_x - d$ whenever the conditional expected value of this reward $E\left[\theta_x - d_x | S_{N,x}\right]$ is positive, and to receive $d_x - \theta_x$ when it is negative. When making decisions in this way, the conditional expected terminal reward received is

$$\max\left\{E\left[\theta_x - d_x | S_{N,x}\right], E\left[d_x - \theta_x | S_{N,x}\right]\right\} = \left|E\left[\theta_x - d_x | S_{N,x}\right]\right| = \left|\frac{\alpha_{N,x}}{\alpha_{N,x} + \beta_{N,x}} - d_x\right|.$$

Summing this reward across systems x, our conditional expected terminal reward is

$$r(\mathbf{S}_N) = \sum_{x=1}^{k} \left|\frac{\alpha_{N,x}}{\alpha_{N,x} + \beta_{N,x}} - d_x\right|.$$

We optionally allow a cost $c_x \geq 0$ for each sample generated for system x, in the same units as the objective function. For example, if the reward is monetary, then the cost might be the payment made to a cloud computing service such as Amazon Ec2 or Microsoft Azure for the computer time required to run a simulation. If the computers are owned, rather than rented, then it may be appropriate to set $c_x = 0$. Combining this optional sampling cost with the reward $r(\mathbf{S}_N)$ gives the conditional expected overall reward

$$r(\mathbf{S}_N) - \sum_{x=1}^{k} \sum_{n=1}^{N} c_x z_{n,x}.$$

Our goal is to find an algorithm or *policy* for choosing the samples to take, $z_{n,x}$, so as to maximize the expected value of this reward. A policy is a rule for choosing how to allocate the next batch of samples, based on the results of the previous samples as summarized by $\mathbf{S}_n$. Formally, a policy π is a sequence of mappings $\pi = (\pi_0, \ldots, \pi_{N-1})$, where $\pi_n : \Lambda_n \mapsto \mathbb{Z}_+^k$ maps the state $\mathbf{S}_n$ to the action $\mathbf{z}_n = \pi_n(\mathbf{S}_n)$, while satisfying the constraint on the number of samples in each batch. The set of all policies is $\Pi = \left\{\pi = (\pi_0, \ldots, \pi_{N-1}) : \sum_{x=1}^{k} \pi_{n,x}(\mathbf{S}) \leq m \; \forall \, n = 0, \ldots, N-1, \mathbf{S} \in \Lambda_n\right\}$, where $\pi_{nx}(\mathbf{S})$ indicate the xth component of $\pi_n(\mathbf{S})$.

Each policy π induces a probability distribution over $(\mathbf{S}_0, \mathbf{z}_0, \ldots, \mathbf{S}_{N-1}, \mathbf{z}_{N-1}, \mathbf{S}_N)$, which we call P^π. We let E^π indicate the expectation taken with respect to this probability distribution. We define,

$$V_n^\pi(\mathbf{S}) = E^\pi\left[r(\mathbf{S}_N) - \sum_{x=1}^{k}\sum_{n'=n+1}^{N} c_x z_{n',x} \mid \mathbf{S}_n = \mathbf{S}\right],$$

for $0 \le n \le N$, which is the conditional expectation of the future reward under policy π, starting from state $\mathbf{S}$ at time n. We also define the value function, used below in our dynamic programming approach, as

$$V_n(\mathbf{S}) = \sup_{\pi\in\Pi} V_n^\pi(\mathbf{S}).$$

The (unconditional) expected reward obtained under π is $V_0^\pi(\mathbf{S}_0)$, and an optimal policy π^* is any for which $\pi^* \in \arg\max_{\pi\in\Pi} V_0^\pi(\mathbf{S}_0)$.

The problem $\sup_{\pi\in\Pi} V_0^\pi(\mathbf{S}_0)$ is a Markov decision process with finite horizon and finite state space, and its solution is characterized by the dynamic programming equations. To apply dynamic programming, we first write down the dynamic programming equation, which is a recursive relation for V_n:

$$V_n(\mathbf{S}_n) = \max_{\mathbf{z}_{n+1}:\sum_x z_{n+1,x}\le m}\left\{-\sum_x c_x z_{n+1,x} + \mathbb{E}[V_{n+1}(\mathbf{S}_{n+1})|\mathbf{S}_n, \mathbf{z}_{n+1}]\right\}, \quad 0 \le n \le N-1. \tag{3.1a}$$

$$V_N(\mathbf{S}_N) = \sum_{x=1}^{k}\left|\frac{\alpha_{N,x}}{\alpha_{N,x}+\beta_{N,x}} - d_x\right|. \tag{3.1b}$$

Any policy whose actions achieve the maximum in (3.1a) is optimal [17].

We can use these recursive equations to compute V_n directly, first calculating $V_N(\mathbf{S}_N)$ for all possible $\mathbf{S}_N \in \Lambda_N$, and then proceeding in a backward recursion, using previously computed values of $V_{n+1}(\mathbf{S}_{n+1})$ to calculate $V_n(\mathbf{S}_n)$ for all $\mathbf{S}_n \in \Lambda_n$. Then, given these computed value functions, we can compute an optimal policy.

While this direct dynamic programming approach is theoretically well understand, it quickly becomes computationally intractable as k grows. This is because the state space

at time n, Λ_n, has $O((mn)^{2k})$ elements, and so storing the value function at all possible states has a memory requirement that scales exponentially in k. Computation also scales exponentially in k. This computational infeasibility due to the large dimension of the problem is generally referred to as the "curse of dimensionality" [38].

The computationally intractability of computing an optimal policy when k is large leads us to consider other characterizations that can be computed more easily. In the next section, we show how to compute an upper bound on the value of the optimal policy that scales linearly in k, rather than exponentially.

3.3 Upper Bound

In this section, we provide a computationally tractable upper bound on the value of an optimal policy. This bound can be used to calculate an optimality gap for any desired heuristic policy π, by comparing the upper bound to an estimate of the heuristic policy's value $V^{\pi}(\mathbf{S}_0)$ obtained from direct simulation. This in turn can be used to judge whether a particular heuristic policy is good enough to be used in practice, or if more development (either of better heuristics or tighter upper bounds) would be worthwhile.

The main idea in our upper bound is to relax the constraints $\sum_{x=1}^{k} z_{n,x} \leq m$ with a Lagrange multiplier. As the first step, we introduce values $\boldsymbol{\lambda} = (\lambda_1, ..., \lambda_N) \in \mathbb{R}_+^k$. The value λ_n will be the Lagrange multiplier for the constraint $\sum_{x=1}^{k} z_{n,x} \leq m$. For each $\boldsymbol{\lambda}$, define a modified value function $V_n^{\boldsymbol{\lambda}}(\mathbf{S}_n)$ via the following recursion:

$$V_n^{\boldsymbol{\lambda}}(\mathbf{S}_n) = \max_{\mathbf{z}_{n+1} \in \{0,1,...,m\}^k} \Big\{ -\sum_x c_x z_{n+1,x} + \mathbb{E}\Big[V_{n+1}^{\boldsymbol{\lambda}}(\mathbf{S}_{n+1}) | \mathbf{S}_n, \mathbf{z}_{n+1} \Big] - \lambda_{n+1} \Big(\sum_{x=1}^{k} z_{n+1,x} - m \Big) \Big\}, n \leq N-1, \tag{3.2a}$$

$$V_N^{\boldsymbol{\lambda}}(\mathbf{S}_N) = \sum_{x=1}^{k} \Big| \frac{\alpha_{N,x}}{\alpha_{N,x} + \beta_{N,x}} - d_x \Big|. \tag{3.2b}$$

We will see below in Lemma 9 that $V_0^{\boldsymbol{\lambda}}(\mathbf{S}_0)$ provides an upper bound on the value function $V_0(\mathbf{S}_0)$, and hence on the value of an optimal policy.

While computing $V_0^{\boldsymbol{\lambda}}(\mathbf{S}_0)$ directly using the recursion (3.2) would also seem to require storing a value for every state in the state space, just as the recursion (3.1), we will see below in Lemma 8 that it can be computed instead as the sum of other modified value functions, each of which corresponds to a single system, and which can be computed by considering a much smaller state space whose size does not grow with k. This will allow efficient computation.

Toward this end, we define the function $V_{n,x}^{\boldsymbol{\lambda}}$ for each x via the recursive relation

$$V_{n,x}^{\boldsymbol{\lambda}}(S_{n,x}) = \max_{z_{n+1,x}\in\{0,\dots,m\}} \Big\{ -z_{n+1,x}(c_x + \lambda_{n+1}) + \mathbb{E}\Big[V_{n+1,x}^{\lambda}(S_{n+1,x})\Big|S_{n,x}, z_{n+1,x}\Big]\Big\},\ n \le N-1, \tag{3.3a}$$

$$V_{N,x}^{\boldsymbol{\lambda}}(S_{N,x}) = \Big|\frac{\alpha_{N,x}}{\alpha_{N,x}+\beta_{N,x}} - d_x\Big|. \tag{3.3b}$$

$V_{n,x}^{\boldsymbol{\lambda}}(S_{n,x})$ is the value function for a dynamic program corresponding to an MCS problem with a single system x, with a new sampling cost $c_x + \lambda_{n+1}$ for samples in batch n. The cost of sampling depends on the batch n. The additional cost λ_{n+1} beyond the cost c_x in our original model arises from the way in which we relaxed the constraint on the number of samples in each batch in the multi-system problem. $V_{n,x}^{\boldsymbol{\lambda}}(S_{n,x})$ can be computed directly using this recursive relation, because the space $\Lambda_{n,x} = \{(\alpha_{0,x}+s_x, \beta_{0,x}+f_x) : s_x, f_x \in \mathbb{Z}_+, s_x+f_x \le mn\}$ in which $S_{n,x}$ takes values has only $O(nm)$ elements, and does not grow exponentially in the problem parameters.

The following lemma shows that $V_n^{\boldsymbol{\lambda}}(\mathbf{S}_n)$ can be computed directly from $V_{n,x}^{\boldsymbol{\lambda}}(S_{n,x})$, allowing its efficient computation.

Lemma 8. *For any* $\boldsymbol{\lambda} \geq \mathbf{0}$*,*

$$V_n^{\boldsymbol{\lambda}}(\mathbf{S}_n) = \sum_{x=1}^{k} V_{n,x}^{\boldsymbol{\lambda}}(S_{n,x}) + m \sum_{n'=n+1}^{N} \lambda_{n'}, \tag{3.4}$$

Proof. This proof uses an inductive argument. At time N, (4.14) holds from (3.3b) and (3.2b). Assume (4.14) holds for time $n+1$. Then,

$$\begin{aligned}
V_n^{\boldsymbol{\lambda}}(\mathbf{S}_n) = \max_{\mathbf{z}_{n+1} \in \{1,..,m\}^k} \Big\{ & - \sum_{x=1}^{k} c_x z_{n+1,x} + \mathbb{E}\Big[\sum_{x=1}^{k} V_{n+1,x}^{\boldsymbol{\lambda}}(S_{n+1,x}) + m \sum_{n'=n+2}^{N} \lambda_{n'} \Big| \mathbf{S}_n, \mathbf{z}_{n+1} \Big] \\
& - \lambda_{n+1}\Big(\sum_{x=1}^{k} z_{n+1,x} - m\Big) \Big\} && (3.5) \\
= \max_{\mathbf{z}_{n+1} \in \{1,...,m\}^k} \Big\{ & - \sum_{x=1}^{k} \Big(c_x z_{n+1,x} + \mathbb{E}\Big[V_{n+1,x}^{\boldsymbol{\lambda}}(S_{n+1,x}) \Big| S_{n,x}, z_{n+1,x} \Big] \Big) - \lambda_{n+1} \sum_{x=1}^{k} z_{n+1,x} \Big\} && (3.6) \\
& + m \sum_{n'=n+1}^{N} \lambda_{n'} && (3.7) \\
= \sum_{x=1}^{k} \max_{z_{n+1,x} \in \{1,...,m\}} \Big\{ & - c_x z_{n+1,x} + \mathbb{E}\Big[V_{n+1,x}^{\boldsymbol{\lambda}}(S_{n,x}) \Big| S_{n+1,x}, z_{n+1,x} \Big] - \lambda_{n+1,x} z_{n+1,x} \Big\} \\
& + m \sum_{n'=n+1}^{N} \lambda_{n'} && (3.8) \\
= \sum_{x=1}^{k} V_{n,x}^{\boldsymbol{\lambda}}(S_{n,x}) & + m \sum_{n'=n+1}^{N} \lambda_{n'}. && (3.9)
\end{aligned}$$

Equation (3.5) follows from the inductive hypothesis. Equality (3.7) holds due to the fact that each system is independent from the current action and states of the other systems. Equality (3.8) holds because the x^{th} summand in the summation of (3.7) only depends on $z_{n+1,x}$; to maximize the sum is to maximize each of the summands by choosing the right $z_{n+1,x}$. □

Setting $n = 0$ in the above lemma, we obtain a readily computed expression for $V_0^{\boldsymbol{\lambda}}(\mathbf{S}_0)$,

$$V_0^{\boldsymbol{\lambda}}(\mathbf{S}_0) = \sum_{x=1}^{k} V_{0,x}^{\boldsymbol{\lambda}}(S_{0,x}) + m \sum_{n=1}^{N} \lambda_n. \tag{3.10}$$

The following lemma shows that this is an upper bound on the value function for our original MCS problem.

Lemma 9. *For any* $\boldsymbol{\lambda} \geq \mathbf{0}$*,*

$$V_0^{\boldsymbol{\lambda}}(\mathbf{S}_0) \geq V_0(\mathbf{S}_0). \tag{3.11}$$

Proof. We first prove

$$V_n^{\boldsymbol{\lambda}}(\mathbf{S}_n) \geq V_n(\mathbf{S}_n), \tag{3.12}$$

for all n such that $0 \leq n \leq N$ using an inductive argument. Then (3.11) holds automatically. At time N, inequality (3.12) follows from equation (3.1b) and (3.2b). Assume (3.12) holds at time $n+1$. Noticing $\sum_{x=1}^{k} z_{n+1,x} - m \leq 0$ as stated in the problem formulation, we have

$$\begin{aligned}
V_n^{\boldsymbol{\lambda}}(\mathbf{S}_n) &= \max_{\mathbf{z}_{n+1}\in\{1,..,m\}^k} \{-\sum_{x=1}^{k} c_x z_{n+1,x} + \mathbb{E}[V_{n+1}^{\boldsymbol{\lambda}}(\mathbf{S}_{n+1})|\mathbf{S}_n, \mathbf{z}_{n+1}] - \lambda_{n+1}(\sum_{x=1}^{k} z_{n+1,x} - m)\} \\
&\geq \max_{\mathbf{z}_{n+1}\in\{1,..,m\}^k} \{-\sum_{x=1}^{k} c_x z_{n+1,x} + \mathbb{E}[V_{n+1}^{\boldsymbol{\lambda}}(\mathbf{S}_{n+1})|\mathbf{S}_n, \mathbf{z}_{n+1}]\} \\
&\geq \max_{\mathbf{z}_{n+1}\in\{1,..,m\}^k} \{-\sum_{x=1}^{k} c_x z_{n+1,x} + \mathbb{E}[V_{n+1}(\mathbf{S}_{n+1})|\mathbf{S}_n, \mathbf{z}_{n+1}]\} \\
&\geq \max_{\mathbf{z}_{n+1}\in\{1,..,m\}^k, \text{s.t. } \sum_{x=1}^{k} z_{n+1,x}\leq m} \{-\sum_{x=1}^{k} c_x z_{n+1,x} + \mathbb{E}[V_{n+1}(\mathbf{S}_{n+1})|\mathbf{S}_n, \mathbf{z}_{n+1}]\} = V_n(\mathbf{S}_n). \square
\end{aligned}$$

Now we have that for any $\boldsymbol{\lambda} \geq \mathbf{0}$, $V_0^{\boldsymbol{\lambda}}(\mathbf{S}_0)$ forms an upper bound on the optimal total expected reward of the original problem. We then obtain the tightest upper bound of this form by selecting the infimum.

Theorem 3.

$$\mathrm{UB}(\mathbf{S}_0) = \inf_{\boldsymbol{\lambda}\geq\mathbf{0}} \Big[\sum_{x=1}^{k} V_{0,x}^{\boldsymbol{\lambda}}(S_{0,x}) + m \sum_{n=1}^{N} \lambda_n \Big] \tag{3.13}$$

gives an upper bound on $V_0(\mathbf{S}_0)$*.*

Proof. This result follows directly from (3.10) and Lemma 9. $\square$

While we have argued that $V_0^{\boldsymbol{\lambda}}(\mathbf{S}_0)$ can be computed efficiently as the sum of values from small dynamic programs, each corresponding to a single system. We now show how the infimum in $\mathrm{UB}(\mathbf{S}_0)$ can be computed. Define $B(\mathbf{S}_0, \boldsymbol{\lambda}) = \sum_{x=1}^{k} V_{0,x}^{\boldsymbol{\lambda}}(S_{0,x}) + m \sum_{n=1}^{N} \lambda_n$. To compute the upper bound in equation (3.13), we first show that $B(\mathbf{S}_0, \boldsymbol{\lambda})$ is convex in $\boldsymbol{\lambda}$ and subsequently (3.13) is a convex optimization problem. Moreover, it is possible to compute the subgradient of $\boldsymbol{\lambda} \mapsto B(\mathbf{S}_0, \boldsymbol{\lambda})$, allowing the use of a first-order convex optimization method.

Lemma 10. *$B(\mathbf{S}_0, \boldsymbol{\lambda})$ is convex in $\boldsymbol{\lambda}$ for any $\boldsymbol{\lambda} \geq \mathbf{0}$.*

Proof. Since $m \sum_{n=1}^{N} \lambda_n$ is convex in $\boldsymbol{\lambda}$, it is sufficient to show $V_{0,x}^{\boldsymbol{\lambda}}(S_{0,x})$ is convex in $\boldsymbol{\lambda}$ for all x. This shall be shown by induction. At time N, $V_{N,x}^{\boldsymbol{\lambda}}(S_{N,x}) = |\frac{\alpha_{N,x}}{\alpha_{N,x}+\beta_{N,x}} - d_x|$ is constant thus convex in $\boldsymbol{\lambda}$. Assume the result holds true for $V_{n+1}^{\boldsymbol{\lambda}}(S_{n+1,x})$, then same is true for $\mathbb{E}[V_{n+1}^{\boldsymbol{\lambda}}(S_{n+1,x})|S_{n,x}, z_{n+1,x}]$ for any fixed $S_{n,x}$ and $z_{n+1,x}$. $-z_{n+1,x}(c_x + \lambda_{n+1})$, which is linear in λ_{n+1}, is also convex in $\boldsymbol{\lambda}$. Since the maximum of convex functions is still convex, $V_{n,x}^{\lambda}(S_{n,x}) = \max_{z_{n+1,x}}\{-z_{n+1,x}(c_x + \lambda_{n+1}) + \mathbb{E}[V_{n+1}^{\lambda}(S_{n+1,x})|S_{n,x}, z_{n+1,x}]\}$ is convex in $\boldsymbol{\lambda}$. Subsequently we have $V_{0,x}^{\boldsymbol{\lambda}}(S_{0,x})$ is also convex through an inductive argument. □

To allow the use of a first-order convex optimization method for solving (3.13), which are generally faster than derivative-free methods, we much provide a method for computing the subgradient of $B(\mathbf{S}_0, \boldsymbol{\lambda})$ with respect to $\boldsymbol{\lambda}$. Since $B(\mathbf{S}_0, \boldsymbol{\lambda})$ is the sum of $V_{0,x}^{\boldsymbol{\lambda}}(S_{0,x})$ and a term that is linear in $\boldsymbol{\lambda}$, it is sufficient to compute the subgradient $V_{0,x}^{\boldsymbol{\lambda}}(S_{0,x})$. The next lemma provides an expression for this subgradient.

Before presenting this lemma, we first introduce some notation. Define $\pi_x^*(\boldsymbol{\lambda})$ to be an optimal policy obtained by solving the single-system MCS problem for system x, that is, the optimal policy for the dynamic program (3.3). Let $r_{n,x}(S_{n-1,x}, z_{n,x}; \lambda_n) = -z_{n,x}(c_x + \lambda_n)$ be the reward collected in state $S_{n-1,x}$ by taking action $z_{n,x}$ for the single-

system MCS problem in (3.3). Let $r_{N+1,x}(S_{N,x})$ be the terminal reward. Then $V^{\boldsymbol{\lambda}}_{0,x}(S_{0,x}) = \mathbb{E}^{\pi^*_x(\boldsymbol{\lambda})}[\sum_{n=1}^{N} r_{n,x}(S_{n-1,x}, z_{n,x}; \lambda_n) + r_{N+1,x}(S_{N,x})|S_{0,x}]$.

Lemma 11. *Fix any $\boldsymbol{\lambda}$, and any corresponding optimal policy $\pi^*_x(\boldsymbol{\lambda})$. Then, the vector*

$$g(S_{0,x}; \boldsymbol{\lambda}) = \left(-\mathbb{E}^{\pi^*_x(\boldsymbol{\lambda})}[z_{n,x}|S_{0,x}] : n = 1, \ldots, N\right) \tag{3.14}$$

is a subgradient of $\boldsymbol{\lambda} \mapsto V^{\boldsymbol{\lambda}}_{0,x}(S_{0,x})$ at $\boldsymbol{\lambda}$.

Proof. Let $\boldsymbol{\lambda}' \in \mathbb{R}^N_+$. Consider the dynamic program (3.3) with this value $\boldsymbol{\lambda}'$, which has value $V^{\boldsymbol{\lambda}'}_{0,x}(S_{0,x})$. Since $\pi^*_x(\boldsymbol{\lambda})$ is a feasible policy for this dynamic program, we have

$$\begin{aligned}
V^{\boldsymbol{\lambda}'}_{0,x}(S_{0,x}) &\geq \mathbb{E}^{\pi^*_x(\boldsymbol{\lambda})}\Big[\sum_{n=1}^{N} r_{n,x}(S_{n-1,x}, z_{n,x}; \lambda'_n) + r_{N+1,x}(S_{N,x})|S_{0,x}\Big] \\
&= \mathbb{E}^{\pi^*_x(\boldsymbol{\lambda})}\Big[\sum_{n=1}^{N} r_{n,x}(S_{n-1,x}, z_{n,x}; \lambda_n) + r_{N+1,x}(S_{N,x})|S_{0,x}\Big] - \sum_{n=1}^{N}(\lambda'_n - \lambda_n)\mathbb{E}^{\pi^*_x(\boldsymbol{\lambda})}[z_{n,x}|S_{0,x}] \\
&= V^{\boldsymbol{\lambda}}_{0,x}(S_{0,x}) + (\boldsymbol{\lambda}' - \boldsymbol{\lambda}) \cdot g(S_{0,x}, \boldsymbol{\lambda})
\end{aligned}$$

Thus $g(S_{0,x}; \boldsymbol{\lambda})$ is a subgradient of $V^{\boldsymbol{\lambda}}_{0,x}(S_{0,x})$. □

We can compute (3.14) recursively using the Markov property. Recall $\pi^*(\boldsymbol{\lambda})$ is an optimal policy for the single-system problem (3.3) when given $\boldsymbol{\lambda}$, and let $z^*_{n+1,x}(S_{n,x})$ be the action taken in state $S_{n,x}$ at time n as dictated by $\pi^*(\boldsymbol{\lambda})$. Let $P(s,n) = \mathbb{P}^{\pi^*(\boldsymbol{\lambda})}[S_{n,x} = s|S_{0,x}]$. We can then write the subgradient (3.14) as

$$-\mathbb{E}^{\pi^*_x(\boldsymbol{\lambda})}[z_{n,x}|S_{0,x} = s_x] = -\sum_{s' \in \Lambda_{n-1,x}} z^*_n(s')P(s', n-1). \tag{3.15}$$

$P(s,n)$ can then be computed recursively as

$$P(s,n) = \begin{cases} \mathbb{1}_{(s=S_{0,x})}, & \text{if } n = 0, \\ \mathbb{P}[S_{1,x} = s|S_{0,x}, z^*_1(S_{0,x})], & \text{if } n = 1, \\ \sum_{s' \in \Lambda_{n-1,x}} \mathbb{P}[S_{n,x} = s|S_{n-1,x} = s', z^*_n(s')] \cdot P(s', n-1), & \text{if } n > 1. \end{cases}$$

Hence we can compute (3.14). Finally, since $B(\mathbf{S}_0, \boldsymbol{\lambda})$ is the sum of the single-system values $V^{\boldsymbol{\lambda}}_{0,x}(S_{0,x})$ and $m \sum_{n=1}^{N} \lambda_n$, we have the following subgradient of $B(\mathbf{S}_0, \boldsymbol{\lambda})$:

$$\sum_{x=1}^{k} \frac{\partial V^{\boldsymbol{\lambda}}_{0,x}(S_{0,x})}{\partial \lambda_n} + m \in \partial B(\mathbf{S}_0, \boldsymbol{\lambda}) \tag{3.16}$$

Equation (3.16) allows us to compute the upper bound (3.13) by first-order convex optimization.

3.4 Index Policy

In this section, we describe an index-based policy based on the same decomposition used to develop the upper bound. The intuition behind this policy is based on an unproven conjecture, but the policy is well-defined whether or not this conjecture is true. We demonstrate in numerical experiments in Section 2.5 that this policy performs well.

This index-based policy considers the relaxed problem (3.2) with a Lagrange multiplier $\boldsymbol{\lambda} = \lambda \mathbf{e}$, where $\mathbf{e} = (1, \ldots, 1)$ is the vector of all 1s and λ is a real number. This index-based policy is based on the intuition that, for any state $S_{n,x}$, as we increase λ and thus $\boldsymbol{\lambda}$ we should see that an optimal policy corresponding to $\boldsymbol{\lambda}$ should take fewer samples, because the samples are more expensive. While we conjecture that this is true, and our numerical experiments support it, we have not confirmed it theoretically.

To calculate the number of samples taken in a given state, our index-based policy varies $\boldsymbol{\lambda}$ until we find a value in which an optimal policy for the relaxed problem takes m samples (or, if no such $\boldsymbol{\lambda}$ exists, it should take as many samples as possible without taking more than m). We then sample according to the optimal policy for this $\boldsymbol{\lambda}$. When we get a new state, we repeat this process, finding a new $\boldsymbol{\lambda}$ vector, and a new sampling allocation.

We define this index-based policy more formally as follows. We first introduce some notation. Let $Q^{\boldsymbol{\lambda}}$ be the set of single-arm policies that are optimal for (3.3) at the given value of $\boldsymbol{\lambda}$. Let $z^{\pi}_{n,x}(S_{n-1,x})$ be the number of samples taken under a single-arm policy π at time n in state $S_{n-1,x}$.

At each time step $n = 1, \ldots, N$, this policy computes $\mathbf{z}_n$ based on $\mathbf{S}_{n-1}$ in the following way:

1. Let $z^{\boldsymbol{\lambda}}_{n,x}(S_{n-1,x}) \in \left\{z^{\pi}_{n,x}(S_{n-1,x}) : \pi \in Q^{\boldsymbol{\lambda}}\right\}$ be the number of samples taken under an optimal single-arm policy with the given set of Lagrange multipliers $\boldsymbol{\lambda}$, breaking ties arbitrarily.
2. Let $\lambda^* = \inf\{\lambda : \sum_x z^{\boldsymbol{\lambda}}_{n,x}(S_{n-1,x}) \leq m, \boldsymbol{\lambda} = \lambda \mathbf{e}\}$.
3. Set $\boldsymbol{\lambda}^* = \lambda^* \mathbf{e}$.
4. Let $z_{n,x} \in \{z^{\pi}_{n,x}(S_{n-1,x}) : \pi \in Q^{\boldsymbol{\lambda}^*}\}$, so as to satisfy $\sum_{x=1}^{k} z_{n,x} \leq m$, breaking ties arbitrarily between different allocations $\mathbf{z}_n$ that satisfy this constraint.

This index-based policy leaves free the tie-breaking rule used when choosing $z^{\boldsymbol{\lambda}}_{n,x}(S_{n-1,x})$, and also when choosing $z_{n,x}$ in the final step. We conjecture that the first tie-breaking rule has no impact on the value for λ^*, although we have not confirmed this theoretically. We conjecture that the best tie-breaking rule used to choose $z_{n,x}$ in the final step is one that minimizes the number of unused cores, $m - \sum_{x=1}^{k} z_{n,x}$, and that it is always possible to find one with $m = \sum_{x=1}^{k} z_{n,x}$, but again we have not confirmed this theoretically.

Figure 3.1 illustrates the above procedure with two systems and two parallel resources. Figure 3.1(a) and 3.1(b) show how the optimal number of samples varies with the value of λ for each system, given their current states $(2, 2)$ and $(3, 3)$. Figure 3.1(c)

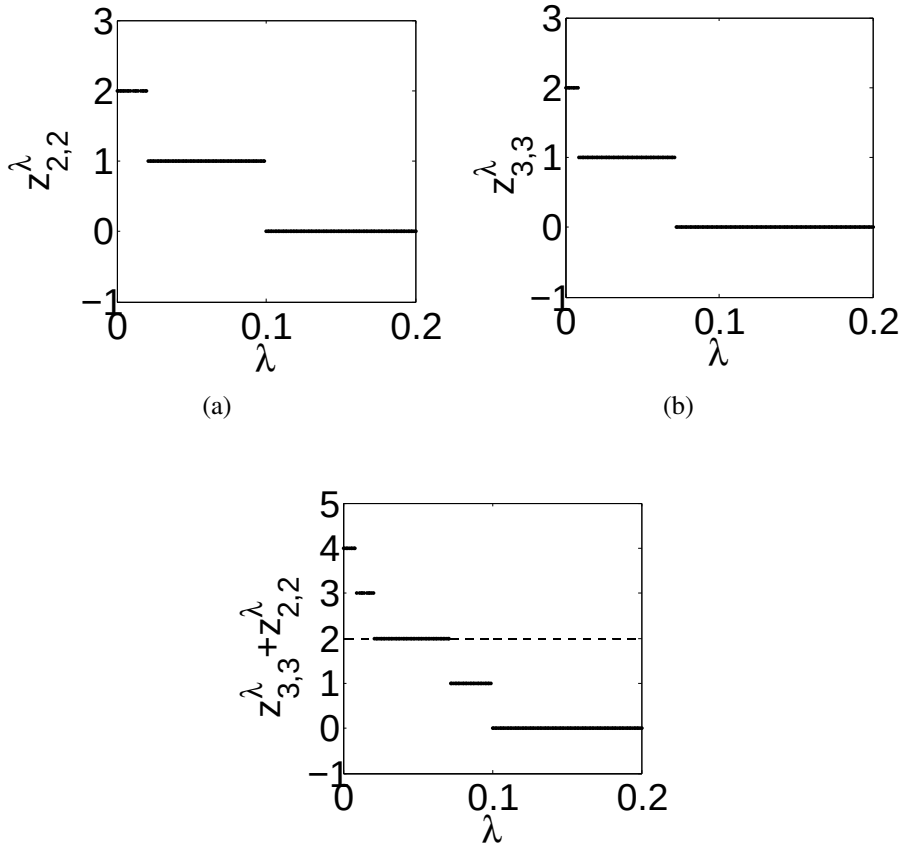

Figure 3.1: This figure illustrates how $\mathbf{z}_n$ is chosen by the index-based policy with $k = 2$ systems, $m = 2$ parallel computing resources, and $N = 20$ simulation batches. Figure (a) plots $z^{\boldsymbol{\lambda}}_{2,2}$, the optimal number of samples to take in batch 3 from system 1 when it is in state (2,2), against the value of λ; Figure (b) plots $z^{\boldsymbol{\lambda}}_{3,3}$, the optimal number of samples to take in batch 3 from system 2 when it is in state (3,3). Figure (c) plots $z^{\boldsymbol{\lambda}}_{2,2} + z^{\boldsymbol{\lambda}}_{3,3}$, the optimal total number of samples to take across both systems. The dashed line in (c) shows the constraint $m = 2$, and λ^* will be the left endpoint of the solid line overlapping this dashed line. The number of samples taken from each system will be $z^{\boldsymbol{\lambda}^*}_{2,2} = 1$ and $z^{\boldsymbol{\lambda}^*}_{3,3} = 1$ respectively.

shows the optimal total number of samples across both systems. Since there are two parallel resources, the constraint is $m = 2$. Hence in this case λ^* is the left end point of the interval at height $z^{\boldsymbol{\lambda}}_{2,2} + z^{\boldsymbol{\lambda}}_{3,3} = 2$.

3.5 Numerical Results

In this section, we present numerical results illustrating the upper bound and the index-based policy, as well as the baseline equal allocation policy.

We consider 4 different value of k: $k = 2, 4, 8, 16$. For each value of k, we set the time horizon $N = 5$, the threshold value $d_x = 0.2$ for all $x \in \{1, ..., k\}$, and the number of parallel computing resources $m = k$. We set the initial state to be the same for every system: $S_{0,x} = (1, 1)$. We first calculated the upper bounds according to Theorem 3 for each value of k. The squared dots in Figure 3.2 represents the values of the upper bounds (shown on a scale in which we divide by k). We then simulated the index-based policy described in Section 4.6 for 10000 iterations respectively for $k = 2, 4, 8, 16$ respectively. The thinner lines in Figure 3.2 show 95% confidence intervals for the mean performance of this policy. As a baseline, we also simulate the equal allocation policy in which the m parallel computing resources are distributed equally to the k systems. The equal allocation policy is simulated for $50,000$ replications for each value of k, and 95% confidence intervals for the mean performance are shown as the thicker lines in Figure 3.2.

Confidence intervals for the index policy are wider than those for the equal allocation policy because fewer samples are taken when estimating the expected value of the index policy. This is because each simulation of the index policy takes a substantial amount of time in our current implementation. For the same reason, the index policy's confidence intervals still overlap the upper bounds. If we took more samples, we expect that the upper limit of the confidence interval would eventually fall below the upper bound, because we do not think that our policy is optimal. Nevertheless, our results show that the index policy performs substantially better than the equal allocation policy in the

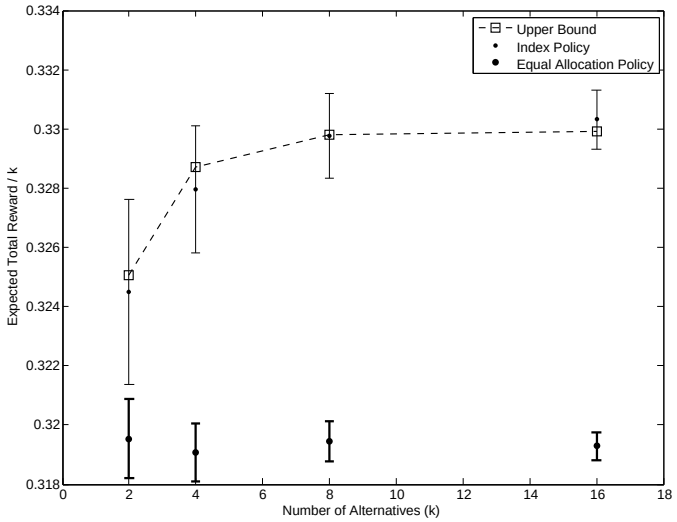

Figure 3.2: This figure shows the upper bound on the performance of the optimal policy for the MCS problem (dashed line with squares) normalized by dividing by k, as well as the estimated performance of two suboptimal policies: the index policy from Section 4.6 (thinner lines and dots); and the equal allocation policy (thicker lines and dots). The setting pictured uses $m = k$, $d_x = 0.2$, $\alpha_{0x} = \beta_{0x} = 1$, $c_x = 0$. We use 10,000 independent replications to estimate the value of the index policy, and 50,000 for the equal allocation policy. The plot shows that the index policy is substantially better than equal allocation, and is statistically indistinguishable from optimal given the number of replications performed.

setting studied.

In the figure, the upper bound, divided by k, initially increases, and then levels out. This can be understood as follows. Because our initial state $S_{0,x}$ is identical for each system, our upper bound 3.13 can be rewritten as $\mathrm{UB}(\mathbf{S}_0) = \inf_{\boldsymbol{\lambda}\geq\mathbf{0}} \left[k V^{\boldsymbol{\lambda}}_{0,x}(S_{0,x}) + k \sum_{n=1}^{N} \lambda_n \right]$. where x is any arbitrary x. Dividing by k provides $\frac{1}{k}\mathrm{UB}(\mathbf{S}_0) = \inf_{\boldsymbol{\lambda}\geq\mathbf{0}} \left[V^{\boldsymbol{\lambda}}_{0,x}(S_{0,x}) + \sum_{n=1}^{N} \lambda_n \right]$.

$V^{\boldsymbol{\lambda}}_{0,x}$ does not depend on k directly, but it does depend on m which is the constraint on the maximum number of samples that can be taken from system x in any batch, and

we set $m = k$ in our experiments. Thus, when we increase k, we loosen this constraint. We believe this is why $\frac{1}{k}\text{UB}(\mathbf{S}_0)$ increases initially with k. Then, as k grows large, this constraint is no longer binding, as the optimal value of $\boldsymbol{\lambda}$ causes us to take less than $m = k$ samples in each batch. We believe this is why $\frac{1}{k}\text{UB}(\mathbf{S}_0)$ levels off as k becomes large.

3.6 Conclusion

We offered a computationally feasible way to obtain the upper bound on the total expected reward of the finite-horizon MCS problem through Lagrangian relaxation. We then proposed an index-based policy using this Lagrangian relaxation. Using the upper bound as a reference, we showed this index policy performs close to optimal by running numerical experiments on a specific set of parameters.

CHAPTER 4

BAYES-OPTIMAL EFFORT ALLOCATION IN CROWDSOURCING: BOUNDS AND INDEX POLICIES

In this chapter, we consider another RMAB-like problem: classifications with crowdsourcing. In this problem, one has a group of classification tasks and wishes to hire crowd-workers to determine, for each among a finite pool of simulatable systems, which ones have an expected output measure that exceeds a known threshold. In this chapter, we use Bayesian statistics and dynamic programming to study how one should allocate simulate effort in the MCS problem, so as to best support this final determination. This chapter is organized as follows: In Section 4.1 and 4.2, we introduce the background of the problem and give a literature review; In Section 4.3 and 4.4, we formally state the problem and formulate the problem as a dynamic program; In Section 4.5, we provide a computationally tractable upper bound on the value of a Bayes-optimal procedure; In Section 4.6 we present an index-based heuristic policy derived from the computation of the upper bound, and which is similar to the general index-based policy proposed in Chapter 2; In Section 4.7, we present numerical results in which we demonstrate that this index-based policy performs close to an optimal policy and outperforms the state-of-art; Lastly we make a conclusion in Sections 4.8.

4.1 Introduction

Crowdsourcing can accomplish large-volume tasks such as image classification or document relevance assessment by using large pool of amateur workers at much less expense than is possible by hiring experts or by developing an automatic machine learning method [29]. Moreover, online platforms such as Amazon Mechanical Turk make crowdsourcing service widely accessible by providing a marketplace in which requesters

may post tasks, which crowd-workers may complete in exchange for money. These factors are making crowdsourcing increasingly important.

Although crowdsourcing is less expensive than hiring experts, the number of images or other tasks that a requester can correctly label or process is nonetheless limited by his or her budget. This fact is compounded by the noise and variability inherent to crowd-workers' responses, which typically requires a single item to be processed independently several times by multiple workers.

In this part of the dissertation, our goal is to find a sequential allocation of workers to tasks that most accurately supports a correct aggregated label for each task, subject to a limited budget (which in turn limits the number of workers that a requester can hire) and a limited time horizon. In this chapter we focus on binary labeling tasks, but our approach can also be extended to multi-class labeling.

Intuitively, much can be accomplished through a sophisticated allocation of worker effort: When budgets are large relative to the overall difficulty of the tasks to be accomplished, a good scheme should allocating more workers to those tasks that are more difficult, so that uniform quality can be ensured. When budgets are small, however, those most difficult tasks should be abandoned so that the bulk of the budget can be used to ensure that at least those easy tasks are done correctly.

We adopt a Bayesian approach, which is natural in crowdsourcing because: 1) It allows us to leverage prior information about the tasks to be accomplished, which may be learned in the crowdsourcing setting from features associated with each task and the typically large collections of historical data collected in previous crowdsourcing campaigns; 2) It seeks to maximize average-case performance with respect to the prior distribution, which is natural in crowdsourcing where requesters typically tolerate some vari-

ability in quality, and are most interested in maximizing aggregate performance across a large volume of tasks, rather than ensuring robustness to some worst-case distribution over task characteristics, or studying asymptotic behaviors that do not become relevant until the number of workers working on each task grows large.

Within this Bayesian framework, we formulate and study sequential effort allocation as a partially observable Markov decision process, using tools from dynamic programming. While the curse of dimensionality [38] prevents solving this dynamic program to optimality, we provide a computationally tractable upper bound on the expected performance under any Bayes-optimal effort allocation policy. Upper bounds are useful because they allow evaluating the optimality gap for any given heuristic on any problem instance, simply by simulating the heuristic and comparing its performance to the bound. The technique we use to obtain such upper bound is the Lagrangian Relaxation on weakly coupled dynamic programs discussed in [1] and [25]. The proofs we present in Section 4.5 are very similar in spirit to [1], but while Adelman based his proof on the value functions of the DP formulation in a infinite horizon setting, we offer a proof based on the initial objective function of the problem in a finite horizon setting. Nonetheless, our crowdsourcing model is a specific application of the more general formulation in [1] and [25]. Then, using Lagrange multipliers that appear in this upper bound, we derive an index-based heuristic policy that is similar in spirit to the Gittins index policy for multi-armed bandits [22] and the Whittle index policy for restless bandits [48]. We then show that this index policy has performance close to the upper bound in numerical experiments, and also outperforms other state-of-art policies for resource allocation .

Although the primary novelty and contribution of our work is that it is the first to characterize the performance of the Bayes-optimal policy for effort allocation in crowdsourcing, and to develop Bayesian bandit-style index policies, our work is also novel

is modeling the *asynchronous* nature of crowd-work in a continuous-time setting, in contrast with previous work on effort allocation in crowdsourcing that assumed instant completion of tasks [51], [11], [28]. This model is inspired by how crowd-workers are employed on Amazon Mechanical Turk; allowing an asynchronous process thus gives a closer proximity to the real situations.

4.2 Related Work

There are two major strands of former works to which our work is related. The first is the work on effort allocation and crowd labeling. Much of this work adopts a frequentist viewpoint and focuses on error bounds for inference [30, 20, 29, 45, 27]. [30] proposed an allocation algorithm based on a random graph, and while its performance asymptotically order-optimal, one needs a very large number of workers to make this relevant. [45] incorporates a limited budget, but lacks the notion of optimality. None of the work above considers a finite time horizon. There is also work with more of a Bayesian flavor([51, 4]). While they focused on the efficiency of allocation, they did not consider an optimal solution. Among the work that adopt a Bayesian framework, our work is similar to [10] in that we both form an optimal policy in the form of a stochastic dynamic program. Although they also provide a well-motivated heuristic policy, our work pushes further by deriving an upper bound based on this formulation of optimal policy.

The second strand resides in the literature of Multi-armed bandit (MAB) and stochastic dynamic programming. The formulation a Bayesian-optimal procedure as a dynamic program is considered in [34, 35]. Our use of Lagrangian relaxation is an application of the relaxation method of weakly coupled dynamic program discussed in

[1]. The setting in this work differs from the previous works by that only one task is to be assigned when a worker enters and the completion of task is not instant. The index-based policy proposed in this work, which uses Lagrangian Multipliers to assign indices, draws inspiration from [48].

4.3 Problem Statement

We consider a requester of crowdsourcing service with K independent binary labeling tasks. Due to a budget constraint, the requester allows a maximum of U workers to work on these tasks, and requires all work to be completed by a time horizon T. We model the arrival of workers to the crowdsourcing system by a Poisson process with rate r. (Our model can be generalized to non-homogeneous Poisson processes with little additional effort.)

As each worker enters the system, the requester selects one of the K tasks for the worker to label. We let $z_\ell \in \{1, \ldots, K\}$ indicate the task assigned to the ℓ^{th} worker. (We use $[K]$ to denote $\{1, \ldots, K\}$ for the rest of the paper.) The worker spends a random Exponential(μ) amount of time on the task x, independent of all else, and then provides a binary label y_ℓ.

Workers do not always give the correct label because the task may be ambiguous and thus hard to categorize, or workers may be careless or lack background information when they conduct the labeling process. We suppose that workers are "homogeneous" (a term used in [11]), and give noisy but unbiased labels. More specifically, each task x has an associated unknown value $\theta_x \in [0, 1]$, which is the underlying probability that it will be labeled as positive by a worker. The distribution of the label generated by the ℓ^{th}

worker given $\theta_1, ...\theta_K$ and z_ℓ is

$$y_\ell|\theta_{1:K}, z_\ell \sim \text{Bernoulli}(\theta_{z_\ell}). \tag{4.1}$$

We set a known threshold value d_x, and consider the label for task x being positive if $\theta_x > d_x$. We let $B = \{x : \theta_x > d_x\}$ be the set of tasks whose correct label is positive. Note B is unknown as θ_x are unknown.

For analytical convenience we use the Beta distribution, which is the conjugate prior of the Bernoulli distribution, as the prior for each θ_x independent across all x.

$$\theta_x \sim \text{Beta}(\alpha_{0,x}, \beta_{0,x}).$$

With the assumption of this independent beta prior on each θ_x, and the conditionally independent Bernoulli responses as in (4.1), the posterior on θ_x after some number of workers have provided responses will remain beta-distributed, with first parameter equal to the sum of $\alpha_{0,x}$ and the number of positive responses, and the second parameter equal to the sum of $\beta_{0,x}$ and the number of negative responses. In practice, one can estimate appropriate values for the parameters $\alpha_{0,x}$ and $\beta_{0,x}$ from historical data on tasks previously labeled by the crowd. We discuss this further in section 4.7 where numerical experiments are performed.

Note the assumption of a Beta distribution can be relaxed without a great deal of difficulty, as the posterior distribution will remain in an exponential family parameterized by the number of positive and negative labels observed for the instance. The assumption of independence cannot be easily generalized, as it is necessary for the decomposition in our Lagrangian relaxation, without which the upper bound in Section 4.5 much more challenging to compute.

Thus, after the worker budget U has been exhausted or the time horizon T has

elapsed, the requester will have a posterior distribution on each θ_x which remains beta-distributed. Let α'_x, β'_x be the posterior parameter for this time. At this time, we model the requester as choosing, for each task x, an estimated label based on the responses of the crowd-workers, and then receiving a reward of 1 for each correctly labeled task, and 0 for the incorrectly labeled tasks. (Our approach can be easily generalized to other reward or loss structures that are additive across tasks, and depend only on θ_x and some task-specific estimate based on the crowd's feedback.)

The expected reward under the posterior that the requester will obtain is $\mathbb{P}(\theta_x > d_x|\alpha'_x,\beta'_x)$, if s/he chooses a positive label, and $\mathbb{P}(\theta_x < d_x|\alpha'_x,\beta'_x)$ if s/he chooses a negative label (θ_x has a density, and so $\theta_x = d_x$ with a posterior probability of 0). Thus, the requester chooses the label giving the larger reward, and achieves a reward whose expected value under the posterior is,

$$R(\alpha'_x,\beta'_x) = \max\left\{\mathbb{P}[\theta_x > d_x|\alpha'_x,\beta'_x], \mathbb{P}[\theta_x < d_x|\alpha'_x,\beta'_x]\right\},$$

Across all tasks, the requester's expected reward under the posterior is

$$R(\boldsymbol{\alpha}',\boldsymbol{\beta}') = \sum_{x=1}^{K} R(\alpha'_x,\beta'_x), \tag{4.2}$$

where $\boldsymbol{\alpha}' = (\alpha'_x : x \in [K])$ and similarly for $\boldsymbol{\beta}'$.

The goal of the requester is to design a policy to dynamically assign tasks to workers entering the system so as to maximize the expected reward received, based on the labels obtained from the crowd-workers.

4.4 Dynamic Programming Formulation

We now formalize the problem statement from Section 4.3 as control of a continuous-time Markov chain, which can be analyzed through a stochastic dynamic program built

on the embedded discrete-time Markov chain. This continuous-time Markov chain tracks the evolution of worker assignments and posterior distributions on θ_x that results from a requester's dynamic assignment policy.

The state of this continuous-time Markov chain contains:

- length-K vectors $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_K)$ and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_K)$ that will describe the posterior distribution on each θ_x given the labels observed thus far (θ_x will be distributed according to Beta(α_x, β_x) under this posterior).
- a length-K vector $\mathbf{w} = (w_1, ..., w_K)$ that tracks the number of workers currently working on each task.
- an integer ℓ that tracks the number of workers that have entered the system and been assigned to tasks (but not necessarily completed them).
- the time t of the most recent *event*, either a worker completing a task, or a worker arriving.

We indicate such a generic state by $s = (\boldsymbol{\alpha}, \boldsymbol{\beta}, t, \mathbf{w}, l)$ and let $\mathbb{S} = \mathbb{R}^K \times \mathbb{R}^K \times \mathbb{R} \times \mathbb{N}^K \times \mathbb{N}$ be the set of possible values this state can take. We let $\boldsymbol{\alpha}(s)$, $\boldsymbol{\beta}(s)$, $t(s)$, $\mathbf{w}(s)$ and $\ell(s)$ all indicate the corresponding components of s.

Transitions occur in this Markov chain when workers complete tasks, and when workers arrive to start work on a task. We use n to count the number transitions (or "events"), we let $S_n \in \mathbb{S}$ indicate the state just after the n^{th} event, for $n \geq 1$. The initial state is $S_0 = (\boldsymbol{\alpha}_0, \boldsymbol{\beta}_0, 0, \mathbf{0}, 0)$, where $\boldsymbol{\alpha}_0 = (\alpha_{0,x} : x \in [K])$ and $\boldsymbol{\beta}_0 = (\beta_{0,x} : x \in [K])$ together describe the prior distribution, and $\mathbf{0}$ is a vector of K zeros.

We let Δ_n denote the time duration between event n and $n+1$, i.e., $\Delta_n = t(S_{n+1}) - t(S_n)$. Then, $\Delta_n | S_n \sim \text{Exp}(\mu \sum_{x=1}^{K} w_x(S_n) + r)$.

We define a policy π that controls how the requester assigns incoming workers to tasks, based on the current state. This policy π will map S_n and Δ_n onto $\{0,1\}^K$, and $\pi(S_n, \Delta_n)$ will give the number of new workers assigned to each of the K tasks, if the transition from S_n to S_{n+1} was caused by an arriving worker. Below we will constrain this to prevent assigning more than one task to a worker, and then later in the Lagrangian relaxation we will relax this constraint.

Formally, let Π be the set of all measurable functions from $\mathbb{S} \times \mathbb{R}_+$ to $\{0,1\}^K$. Then, let $|\cdot|$ return the sum of individual components of a vector, and define

$$\Pi_0 = \{\pi \in \Pi : |\pi(s, \Delta)| \leq 1, \forall s \in \mathbb{S}, t \in \mathbb{R}\}, \tag{4.3}$$

where we have added the additional constraint to Π that at most one task can be assigned to an incoming worker. Only those $\pi \in \Pi_0$ will be feasible policies for the problem of interest, but we will consider the larger set of policies Π to support later theoretical analysis.

The set of policies Π_0 allows not assigning an incoming worker to a task even when budget or time remains, but we will see below that this will still exhaust one unit of budget, and so optimal policies (or reasonable heuristics) will always assign incoming workers to tasks when possible.

Each $\pi \in \Pi$ defines a discrete time Markov chain $(S_n : n \in \{0, 1, \ldots\})$ over the state space $\mathbb{S}$, whose transition kernel we will indicate by $\mathbb{P}^\pi(s'|s)$. This transition kernel can be written as

$$\mathbb{P}^\pi(s'|s) = \int_0^\infty \mathbb{P}^\pi(s'|s, \Delta) \exp(-\Delta q(s))\, d\Delta,$$

where we have defined

$$q(s) = \mu \sum_{x=1}^{K} w_x(s) + r.$$

Thus, to complete the description of this transition kernel, it is sufficient to describe $\mathbb{P}^{\pi}(S_{n+1}|S_n, \Delta)$. For this description we suppose $S_n = (\boldsymbol{\alpha}, \boldsymbol{\beta}, t, \mathbf{w}, \ell)$ and let $q = q(S_n)$.

When $t + \Delta_n \geq T$, the system has exceeded its time horizon, all outstanding tasks on which workers are currently working are canceled, and only the time is updated: $S_{n+1} = (\boldsymbol{\alpha}, \boldsymbol{\beta}, t + \Delta_n, \mathbf{0}, \ell)$.

When $t + \Delta_n < T$, time remains and the next event can be either a worker arrival or a worker completion. A completion either outputs a positive result or a negative result.

A worker arrives with probability r/q. If $\ell < U$, then the requester allocates this worker to a task, and the total number of arrivals is incremented: $S_{n+1} = (\boldsymbol{\alpha}, \boldsymbol{\beta}, t + \Delta_n, \mathbf{w} + \pi(S_n, \Delta_n), \ell + 1)$. If $\ell \geq U$, then the worker budget has been exceeded, and the requester cannot allocate the worker, so $S_{n+1} = (\boldsymbol{\alpha}, \boldsymbol{\beta}, t + \Delta_n, \mathbf{w}, \Delta_n), \ell)$.

For each $x \in [K]$, a worker completes this task x and reports a positive label with probability $\frac{\alpha_x}{\alpha_x+\beta_x}\frac{\mu w_x}{q}$. When this occurs, $S_{n+1} = (\boldsymbol{\alpha} + \mathbf{e}_x, \boldsymbol{\beta}, t, \mathbf{w} - \mathbf{e}_x, \ell)$.

Similarly, a worker completes task x and reports a negative label with probability $\frac{\beta_x}{\alpha_x+\beta_x}\frac{\mu w_x}{q}$. When this occurs, $S_{n+1} = (\boldsymbol{\alpha}, \boldsymbol{\beta} + \mathbf{e}_x, t, \mathbf{w} - \mathbf{e}_x, \ell)$.

This completely specifies the transition kernel for the discrete-time Markov chain that describes the continuous-time dynamics of both worker allocation and the posterior distribution on each θ_x.

To model completion, we define $\mathbb{S}_A = \{s \in \mathbb{S} : t(s) \geq T \text{ or } (\ell(s) \geq U \text{ and } \mathbf{w}(s) = \mathbf{0})\}$ to be the set of states in which our time horizon has elapsed, or our worker budget has been exhausted and all allocated workers have finished their work. We then let $N = \inf\{n \geq 0 : S_n \in \mathbb{S}_A\}$ be the number of events that occur up to and including the time when we reach a state in $\mathbb{S}_A$. The posterior $\boldsymbol{\alpha}(S_N), \boldsymbol{\beta}(S_N)$ is the one with which the

requester must make his/her final determination of the task labels, and so the expected reward under the posterior that s/he receives at time $t(S_N)$ is $R(\boldsymbol{\alpha}(S_N),\boldsymbol{\beta}(S_N))$.

Recall that our goal stated in section 4.3 was to find the dynamic allocation policy π of workers to tasks that maximizes the expected number of correctly classified tasks. With the definition of this Markov chain in place, this overarching goal may be stated formally as solving

$$\sup_{\pi\in\Pi_0} \mathbb{E}^{\pi}\left[R(\boldsymbol{\alpha}(S_N),\boldsymbol{\beta}(S_N))\right]. \tag{4.4}$$

As a stochastic control problem, its solution may be characterized using stochastic dynamic programming. We define the value function as

$$V(s) = \sup_{\pi\in\Pi_0} \mathbb{E}^{\pi}\left[R(\boldsymbol{\alpha}(S_N),\boldsymbol{\beta}(S_N))|S_0 = s\right], \tag{4.5}$$

and observe that the value function satisfies the dynamic programming recursion.

First, for $s \in \mathbb{S}_A$, we have $V(s) = R(\boldsymbol{\alpha}(s),\boldsymbol{\beta}(s))$. Then, for $s = (\boldsymbol{\alpha},\boldsymbol{\beta},t,\mathbf{w},\ell) \notin \mathbb{S}_A$ and $q = q(s)$, we have, If $\ell < U$:

$$\begin{aligned}
V(s) = &\Big(1-\exp(-q(T-t))\Big)\cdot\Big[\\
&r\int_0^{T-t}\max_z V(\boldsymbol{\alpha},\boldsymbol{\beta},t+y,\mathbf{w}+\mathbf{e}_z,\ell+1)e^{-qy}dy+\\
&\sum_{x=1}^{K}\mu w_x\Big(\frac{\alpha_x}{\alpha_x+\beta_x}\int_0^{T-t}V(\boldsymbol{\alpha}+\mathbf{e}_x,\beta,t+y,\mathbf{w}-\mathbf{e}_x,\ell)e^{-qy}dy\\
&+\frac{\beta_x}{\alpha_x+\beta_x}\int_0^{T-t}V(\boldsymbol{\alpha},\beta+\mathbf{e}_x,t+y,\mathbf{w}-\mathbf{e}_x,\ell)e^{-qy}dy\Big)\Big]\\
&+\exp(-q(T-t))\Big\{R(\boldsymbol{\alpha},\boldsymbol{\beta})\Big\}.
\end{aligned} \tag{4.6}$$

If $\ell \geq U$:

$$
\begin{aligned}
V(s) = \Big(1 - \exp(-q(T-t))\Big) \cdot \Big[& \\
& r \int_0^{T-t} V(\boldsymbol{\alpha}, \boldsymbol{\beta}, t+y, \mathbf{w}, \ell) e^{-qy} dy + \\
& \sum_{x=1}^{K} \mu w_x \Big(\frac{\alpha_x}{\alpha_x + \beta_x} \int_0^{T-t} V(\boldsymbol{\alpha} + \mathbf{e}_x, \beta, t+y, \mathbf{w} - \mathbf{e}_x, \ell) e^{-qy} dy \\
& + \frac{\beta_x}{\alpha_x + \beta_x} \int_0^{T-t} V(\boldsymbol{\alpha}, \beta + \mathbf{e}_x, t+y, \mathbf{w} - \mathbf{e}_x, \ell) e^{-qy} dy \Big) \Big] \\
& + \exp(-q(T-t)) \Big\{ R(\boldsymbol{\alpha}, \boldsymbol{\beta}) \Big\}. \qquad (4.7)
\end{aligned}
$$

Moreover, knowing the value function reveals an optimal policy: an optimal policy is given by choosing the task Z_ℓ to assign to the next worker, in response to previous state S_n at time $t(S_n)+\Delta_n$, to achieve the maximum in $\max_z V(\boldsymbol{\alpha}(S_n), \boldsymbol{\beta}(S_n), t(S_n)+\Delta_n, \mathbf{w}(S_n)+\mathbf{e}_z, \ell+1)$.

However, solving this dynamic program is computationally infeasible. For example, if we discretize the continuous time line to just 1000 intervals, when we have $K = 4$ tasks, the number of states to consider after $l = 20$ workers entering the system is $2.26 * 10^{11}$, which is too big to compute. Hence we seek to first provide an upper bound to the optimal value and then use the upper bound as the yardstick to measure how close a heuristic policy performs to an optimal policy.

4.5 Upper Bound on the Bayes-Optimal Policy

Although solving (4.4) directly using the stochastic dynamic program (4.6),(4.7) is computationally intractable, in this section we show how to obtain a computationally feasible upper bound on the value (4.4) using a Lagrangian relaxation.

Recall we use n to count events and S_n is the state corresponding to the n^{th} event. Define n_ℓ as the number of events that have occurred by the time of the ℓ^{th} arrival (in-

clusive), i.e., $n_\ell = \inf\{n : \ell(S_n) = \ell\}$. For $1 \leq \ell \leq U$, define $a_\ell = \pi(S_{n_\ell - 1}, \Delta_{n_\ell - 1})$, so that $a_{\ell,x} = 1$ if the ℓ^{th} worker is assigned to task x. Therefore Π_0 satisfies $\Pi_0 = \{\pi \in \Pi : \mathbb{P}^\pi(\sum_{x=1}^K a_{\ell,x} \leq 1) = 1 \ \forall \ell\}$. We also define a new subset of Π:

$$\Pi_1 = \left\{ \pi \in \Pi : \mathbb{E}^\pi \left[\sum_{x=1}^K a_{\ell,x} \right] \leq 1 \ \forall \ell \right\}. \tag{4.8}$$

Under Π_1, we may assign a worker to more than one task along a particular sample path, as long as the *expected* number of tasks assigned to each worker is no larger than 1. Returning to our Markov chain model, we will observe that when a worker is assigned more than one task, the tasks are completed independently from each other. Observe that Π_1 includes a larger set of policies than Π_0, and that Π includes a set that is larger still, i.e., $\Pi_0 \subseteq \Pi_1 \subseteq \Pi$. Our result will use this relation.

To streamline notation, let $R = R(\boldsymbol{\alpha}(S_N), \boldsymbol{\beta}(S_N))$. The optimal reward for the original crowdsourcing problem (4.4) is then,

$$R_0 = \sup_{\pi \in \Pi_0} \mathbb{E}^\pi [R], \tag{4.9}$$

and the optimal reward under the larger class of policies Π_1 is

$$R_1 = \sup_{\pi \in \Pi_1} \mathbb{E}^\pi [R]. \tag{4.10}$$

Here we introduce a non-negative vector $\boldsymbol{\lambda} = \{\lambda_1, ..., \lambda_U\} \geq \mathbf{0}$, which we use below as a Lagrange multiplier within a Lagrangian relaxation.

More specifically, we will relax the constraint that each worker is assigned to at most one task, but will penalize the number of tasks assigned in a way that ensures that an upper bound holds regardless of what $\boldsymbol{\lambda}$ is (as long as it is componentwise-nonnegative). This will then provide an upper bound on the optimal value of the original problem R_0, which can be made tighter by minimizing over $\boldsymbol{\lambda}$. This upper bound can then be

computed below via decomposition into K small dynamic programs of fixed dimension that can be solved efficiently, even as K grows large.

Our upper bound is provided in the following theorem.

Theorem 4. *The term*

$$\inf_{\boldsymbol{\lambda}\geq\mathbf{0}} \sup_{\pi\in\Pi} \mathbb{E}^{\pi}\Big[R - \sum_{\ell=1}^{U}(\lambda_\ell \sum_{x} a_{\ell,x})\Big] + \sum_{\ell=1}^{U}\lambda_\ell \tag{4.11}$$

forms an upper bound to R_0.

Proof.

$$\begin{aligned}
&\sup_{\pi\in\Pi} \mathbb{E}^{\pi}\Big[R - \sum_{\ell=1}^{U}(\lambda_\ell \sum_{x} a_{\ell,x})\Big] + \sum_{\ell=1}^{U}\lambda_\ell \\
&= \sup_{\pi\in\Pi} \mathbb{E}^{\pi}\left[R - \sum_{\ell=1}^{U}\left(\lambda_\ell(\sum_{x} a_{\ell,x} - 1)\right)\right] \\
&\geq \sup_{\pi\in\Pi_1} \mathbb{E}^{\pi}\left[R - \sum_{\ell=1}^{U}\left(\lambda_\ell(\sum_{x} a_{\ell,x} - 1)\right)\right] \\
&= \sup_{\pi\in\Pi_1} \mathbb{E}^{\pi}\left[R\right] - \sum_{\ell=1}^{U}\lambda_\ell\left(\mathbb{E}^{\pi}[\sum_{x} a_{\ell,x}] - 1\right) \\
&\geq \sup_{\pi\in\Pi_1} \mathbb{E}^{\pi}\left[R\right] \\
&\geq \sup_{\pi\in\Pi_0} \mathbb{E}^{\pi}\left[R\right]
\end{aligned} \tag{4.12}$$

The first inequality is due to $\Pi_1 \subseteq \Pi$. The second inequality is because $\boldsymbol{\lambda} \geq \mathbf{0}$ and $\mathbb{E}^{\pi}[\sum_x a_{\ell,x}] \leq 1$ for any $\pi \in \Pi_1$. The third inequality is due to $\Pi_0 \subseteq \Pi_1$. Since (4.12) holds true for any value of $\boldsymbol{\lambda} > \mathbf{0}$, we obtain Theorem 4. □

Calculating the supremum term in Theorem 4 directly by dynamic programming is again computationally infeasible, because the state space of this dynamic program again is over all of $\mathbb{S}$, which has $3K + 1$ dimensions. We avoid this issue by decomposing this

supremum term into the sum of the optimal values for K dynamic programs, one for each task, each of which has a much more manageable 4 dimensions.

To support this decomposition, we write the state $S_n \in \mathbb{S}$ for the whole system (K tasks) as $S_n = (S_{n,1}, \ldots, S_{n,K})$, where $S_{n,x}$ is the state for task x when n events have occurred, and includes $\alpha_x, \beta_x, t, w_x$, and the global counter ℓ. We let $\mathbb{S}^{(x)} = \mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{N} \times \mathbb{N}$ be the set of possible values for this single-task state $S_{n,x}$.

Following a development identical to that in Section 4.4, but for the single task x, we may define a space of policies $\pi^{(x)}$ that map the single-task state $S_{n,x}$ and the elapsed time since the last event $\Delta_n^{(x)}$ (counting worker arrivals over the whole system, and completions of task x only) onto a binary decision of whether or not to allocate an incoming worker to task x, so that $\pi^{(x)}(S_n, \Delta_n) \in \{0, 1\}$. Following this development, we construct K independent Markov chains, one for each task, where each one is controlled by its respective single-task policy $\pi^{(x)}$. We define N as before, to be the first time that the time horizon elapses, or our worker budget has been exhausted and all outstanding workers have completed their work. We then let $R_x = R(\alpha_x(S_N), \beta_x(S_N))$ be the reward obtained from this the single task at this time.

The following theorem shows that the bound in Theorem 4 can be re-written in terms of the sum of solutions of single-task dynamic programming problems, where each obtains the reward R_x, and is penalized for assigning workers to its task.

Theorem 5.

$$\inf_{\boldsymbol{\lambda} \geq \mathbf{0}} \sum_{x=1}^{K} \sup_{\pi^{(x)} \in \Pi^{(x)}} \mathbb{E}^{\pi^{(x)}} \Big[R_x - \sum_{\ell=1}^{U} \lambda_\ell a_{\ell,x} \Big] + \sum_{\ell=1}^{U} \lambda_\ell \tag{4.13}$$

forms an upper bound on R_0.

Proof. For any $\boldsymbol{\lambda} \geq \mathbf{0}$:

$$\begin{aligned}
&\sup_{\pi\in\Pi} \mathbb{E}^{\pi}[R - \sum_{\ell=1}^{U}(\lambda_\ell \sum_{x} a_{\ell,x})] \\
=&\sup_{\pi\in\Pi} \mathbb{E}^{\pi}[\sum_{x=1}^{K} R_x - \sum_{x=1}^{K}\sum_{\ell=1}^{U} \lambda_\ell a_{\ell,x}] \\
=&\sup_{\pi\in\Pi} \mathbb{E}^{\pi}\Big[\sum_{x=1}^{K}\Big(R_x - \sum_{\ell=1}^{U} \lambda_\ell a_{\ell,x}\Big)\Big] \\
=&\sup_{\pi\in\Pi} \sum_{x=1}^{K} \mathbb{E}^{\pi}\Big[R_x - \sum_{\ell=1}^{U} \lambda_\ell a_{\ell,x}\Big].
\end{aligned}$$

This is bounded above by,

$$\begin{aligned}
&\sum_{x=1}^{K} \sup_{\pi\in\Pi} \mathbb{E}^{\pi}\Big[R_x - \sum_{\ell=1}^{U} \lambda_\ell a_{\ell,x}\Big] \\
=&\sum_{x=1}^{K} \sup_{\pi^{(x)}\in\Pi^{(x)}} \mathbb{E}^{\pi^{(x)}}\Big[R_x - \sum_{\ell=1}^{U} \lambda_\ell a_{\ell,x}\Big]. \qquad (4.14)
\end{aligned}$$

The equality at (4.14) is because $\sup_{\pi\in\Pi} \mathbb{E}^{\pi}\Big[R_x - \sum_{\ell=1}^{U} \lambda_\ell a_{\ell,x}\Big]$ depends only on $(\alpha_{t,x}, \beta_{t,x}, w_{t,x} : 0 \leq t \leq T)$, which is in turn governed by $\pi^{(x)}$. By Theorem 4, for any $\boldsymbol{\lambda} \geq \mathbf{0}$,

$$\sum_{x=1}^{K} \sup_{\pi^{(x)}\in\Pi^{(x)}} \mathbb{E}^{\pi^{(x)}}\Big[R_x - \sum_{\ell=1}^{U} \lambda_\ell a_{\ell,x}\Big] + \sum_{\ell=1}^{U} \lambda_\ell$$

forms an upper bound on R_0. This hold for any $\boldsymbol{\lambda} \geq \mathbf{0}$, and so we have thus proved Theorem 5. □

Since the state space is much smaller for a single-task system, we can use dynamic programming to solve for the supremum term

$$\sup_{\pi^{(x)}\in\Pi^{(x)}} \mathbb{E}^{\pi^{(x)}}\Big[R_x - \sum_{\ell=1}^{U} \lambda_\ell a_{\ell,x}\Big], \qquad (4.15)$$

for any $\boldsymbol{\lambda}$ value. What remains in the computing of the upper bound is to solve for the infimum in Theorem 5. We explore the convexity property of the problem follow by a binary search. Define $B(\lambda) = \sum_{x=1}^{K} \sup_{\pi^{(x)}\in\Pi^{(x)}} \mathbb{E}_0^{\pi^{(x)}}\Big[R_x - \sum_{\ell=1}^{U} \lambda_\ell a_{\ell,x}\Big] + \sum_{\ell=1}^{U} \lambda_\ell$, which is

the upper bound derived in Theorem 5 without the infimum. First we prove that $B(\boldsymbol{\lambda})$ is convex in $\boldsymbol{\lambda}$.

Lemma 12. *$B(\boldsymbol{\lambda})$ is convex in $\boldsymbol{\lambda}$.*

Proof. First note $\sum_{\ell=1}^{U} \lambda_\ell$ is convex in $\boldsymbol{\lambda}$. To prove $\sup_{\pi^{(x)}\in\Pi^{(x)}} \mathbb{E}^{\pi^{(x)}}\Big[R_x - \sum_{l=1}^{U} \lambda_l a_{l,x}\Big]$ is convex in $\boldsymbol{\lambda}$, pick any $\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2 \geq \mathbf{0}$ and $t \in [0,1]$. Let

$$\pi' = \sup_{\pi^{(x)}\in\Pi^{(x)}} \mathbb{E}^{\pi^{(x)}}\Big[R_x - \sum_{l=1}^{U}(t\lambda_{1,l} + (1-t)\lambda_{2,l})a_{l,x}\Big] \tag{4.16}$$

We have

$$\begin{aligned}
&t \sup_{\pi^{(x)}\in\Pi^{(x)}} \mathbb{E}^{\pi^{(x)}}\Big[R_x - \sum_{l=1}^{U} \lambda_{1,l} a_{l,x}\Big] \\
&+(1-t) \sup_{\pi^{(x)}\in\Pi^{(x)}} \mathbb{E}^{\pi^{(x)}}\Big[R_x - \sum_{l=1}^{U} \lambda_{2,l} a_{l,x}\Big] \\
\geq& t\mathbb{E}^{\pi'}\Big[R_x - \sum_{l=1}^{U} \lambda_{1,l} a_{l,x}\Big] + (1-t)\mathbb{E}^{\pi'}\Big[R_x - \sum_{l=1}^{U} \lambda_{2,l} a_{l,x}\Big] \\
=& \mathbb{E}^{\pi'}\Big[R_x - \sum_{l=1}^{U}(t\lambda_{1,l} + (1-t)\lambda_{2,l})a_{l,x}\Big] \\
=& \sup_{\pi^{(x)}\in\Pi^{(x)}} \mathbb{E}^{\pi^{(x)}}\Big[R_x - \sum_{l=1}^{U}(t\lambda_{1,l} + (1-t)\lambda_{2,l})a_{l,x}\Big].
\end{aligned}$$

Hence $\sup_{\pi^{(x)}\in\Pi^{(x)}} \mathbb{E}^{\pi^{(x)}}\Big[R_x - \sum_{l=1}^{U} \lambda_l a_{l,x}\Big]$ is convex in $\boldsymbol{\lambda}$ for any $x \in [K]$, subsequently the sum of $\sup_{\pi^{(x)}\in\Pi^{(x)}} \mathbb{E}^{\pi^{(x)}}\Big[R_x - \sum_{l=1}^{U} \lambda_l a_{l,x}\Big]$ across x is also convex in $\boldsymbol{\lambda}$. Thus we complete the proof that $B(\boldsymbol{\lambda})$ is convex in $\boldsymbol{\lambda}$. □

With the convexity in $\boldsymbol{\lambda}$, we approximate $\boldsymbol{\lambda}'$ that achieves the infimum by setting $\boldsymbol{\lambda}' = \lambda' \times (1,,,1)$, and use a Fibonacci search to find the infimum. Here we constrain all the units of $\boldsymbol{\lambda}'$ to be the same for simpler computation, a tighter bound can be obtained by allowing each unit of $\boldsymbol{\lambda}'$ to vary and use sub-gradient descent to locate the infimum.

4.6 Index Policy

We introduce an index-based heuristic policy built on the Lagrangian relaxation we used in proving the upper bound. In this policy, we compute some λ_x^* for each task x based on its state $S_{n,x}$, such that λ_x^* is the greatest value of λ that the optimal policy will decide to hire the worker on state $S_{n,x}$ when solving (4.15) with $\boldsymbol{\lambda} = \lambda\mathbf{1}$, $\mathbf{1} = (1, \ldots, 1)$. We then assign the incoming worker to the task with the highest λ^*. The intuition behind this policy is that in a single-task problem described by (4.15), we view λ_ℓ as a cost of employing the ℓ^{th} worker. As λ_ℓ increases, our decision switches from hiring the worker to not hiring, where the switching point is at λ_x^*. Hence tasks with a high λ_x^* are the tasks that are worth hiring more workers to work on. Below we present the algorithm in a more formal way. A useful technique to reduce the amount of computation is to

Algorithm 4: Index Policy

1: **while** $\ell < U$ **do**
2: For each $x \in \{1, \ldots, K\}$, compute $\lambda_x^* = \inf\{\lambda \in \mathbb{R}_+ : a_{\ell,x}^\lambda = 1\}$, where $a_{x,\ell}^\lambda$ is the optimal decision from (4.15) when $\boldsymbol{\lambda} = \lambda\mathbf{1}$.
3: Let $x_* = \arg\max_x \lambda_x^*$. Break tie arbitrarily.
4: Assign task x_* to the ℓ^{th} worker.
5: **end while**

put a cap on the total number of workers that can be assigned to a task, for this reduces the size of the state space of the dynamic program involved in solving for (4.15). This additional cap does not affect the decision made by the Index Policy. Intuitively it is unlikely for any reasonable policy to assign all the U number of workers to one task, so it is unlike for any task to get more than a certain number of workers assigned. One can check the validity of the cap by running simulations with the capped index policy, and

see whether there are tasks that use all the workers that the cap allows.

We show in section 4.7 that this policy's performance is close to optimal.

4.7 Numerical Experiment

For numerical experiment we concentrate on the case in which $T = \infty$. In this case we stop when we reach the maximum number of workers the budget allows. The Bellman's recursion to compute 4.15 in the computation of upper bound for this special case is given in the supplement. In the first set of simulation study, we compare the performances of different policies on simulated data against the corresponding upper bounds. In the second set of simulation study, we use a real dataset for simulation.

4.7.1 Simulation using simulated data

In the first set of simulations we evaluate the performance of the Index Policy using simulated data, and compared the total reward given in (4.2) generated by the Index Policy to the upper bound 4.11. We also compare the performance of the Index Policy to Optimistic Knowledge Gradient(OKG) method [10], which is a state-of-art Bayesian allocation policy. A round of simulated process includes generating either an arrival of worker or a completion of task based on the arrival rate $r = 0.1$ and and completion rate $\mu = 0.4$ with distributions specified in Section 4.3. If it is a completion of task, we generate a label based on the posterior parameters. The process stops when we exhaust all the budget, i.e., the number of workers that are allowed to hire, and we get a reward as in (4.2). We vary the number of tasks to be $K = 10, 100, 1000$, and set the budget to be $U = 1.2K$. We use a non-informative prior with $\boldsymbol{\alpha} = \mathbf{1}$ and $\boldsymbol{\beta} = \mathbf{1}$. We use a threshold

$d_x = 0.5$ for all the tasks. For each value of $K = 10, 100, 1000$, we simulate the process 5000 times, and obtain a 95% confidence interval for the simulated total reward. In Figure 4.1 we show a Semi-log plot of the number of tasks K against the average reward per task with the corresponding confidence intervals.

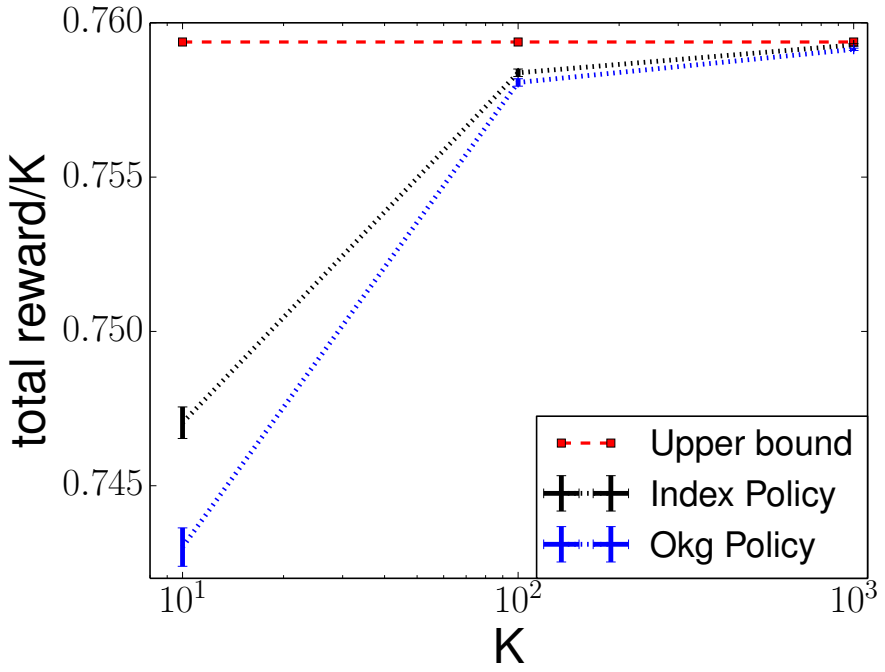

Figure 4.1: Semi-log plot of K against average per task reward (R/K) for $K = 10, 10^2, 10^3$.

We would like to make a note that in this numerical study, we truncate the time horizon of the single-task DP from $1.2K$ to 6 to reduce computational complexity. This truncation preserves the original dynamics as it is very unlikely for a task to be assigned 6 workers when the total number of workers is just 1.2 times the number of tasks. We use simulation to demonstrate that 6 is still a loose cap numerically. We run a simulation with simulated data with 1000 replications for number of workers $K = 10$ and 100, and $U = 1.2K$ and count the number of tasks that uses 0 worker, 1 worker and up to 6 workers. The results are shown in Figure 4.2. The results show that a task uses at most

4 workers, hence set a cap at 6 does not affect the performance of the index policy.

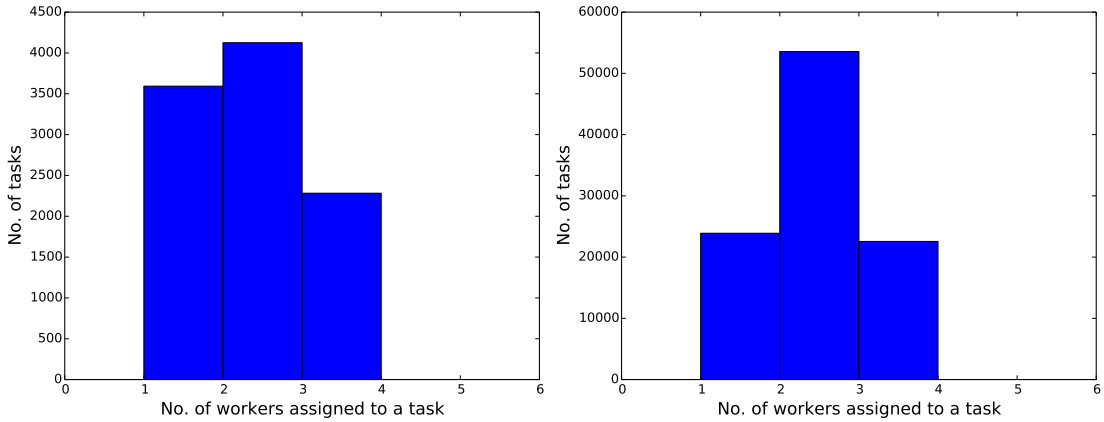

Figure 4.2: Histogram of number of workers assigned to a task

The performance of the index policy is consistently better than the OKG policy, especially for smaller number of tasks. Moreover, the gap between the upper bound and the simulated value gets smaller as K increases, which demonstrates both that the upper bound is tight and the Index Policy performs close to an optimal policy as the number of tasks gets larger.

We emphasize that the improved performance over OKG offered by our Index Policy is only one aspect of the contribution of this work. The other aspect is the tightness of the upper bound, especially for problem instances with many tasks. This tight upper bound for K=1200 shows that the index policy is within 0.03% of optimal, and that continued algorithmic development will not provide significantly increased performance for large-scale crowdsourcing problems with characteristics matching this particular simulated dataset. The ability to bound the improvement from continued algorithmic development for a particular problem instance, or class of problem instances, is useful for managers at companies that use crowdsourcing and wish to allocate engineering\R&D effort.

4.7.2 Simulation using real data

This set of simulations uses a real dataset, PASCAL RTE-1[43], which consists of 800 tasks, each comes with 10 labels obtained from crowdworkers and a gold standard label. (A gold standard label of a classification task is its true label.) We evaluate the performance of the Index Policy against the OKG policy, the Thompson Sampling [8] and a widely used frequentist approach - the upper confidence bound(UCB) policy [3]. (The specific version of UCB used here is UCB1-tuned.) The metric used to evaluate the performance is the accuracy score. More specifically, it is the number of correctly predicted labels over the total number of tasks. In each round of simulation process, we still simulate events (either arrival or completion) the same way as we did in Section 4.7.1. If the event is a completion, we read the most recent label for that task in the dataset. At the end of each process, predicted labels are compared against the gold standard labels. d_x is still 0.5 for all tasks. For each value of $K = 10, 100, 750$, we simulate the process 5000 times, and obtain a 95% confidence interval for the simulated total reward.

Before simulating, we use the remaining 50 tasks from the RTE dataset as the 'historical data' to estimate the parameters of the Beta prior, which is set to be the same across all tasks. This comes with an assumption that all tasks are homogeneous, hence a subset of them are representative of a larger population. We first obtain an estimate of θ_x for each of the 45 tasks, then use these empirical values of θ_x to fit a Beta distribution by Method of moments. In Figure 4.3 we show a Semi-log plot of the number of tasks K against the accuracy score with the corresponding confidence intervals. All the Bayesian policies see an smaller optimality gap when K gets larger. Index Policy performing consistently the best among all the policies. It is proven in [10] that OKG policy is consistent: it achieve 100% accuracy almost surely when number of workers goes to infinity. We demonstrate numerically that the Index Policy performs better

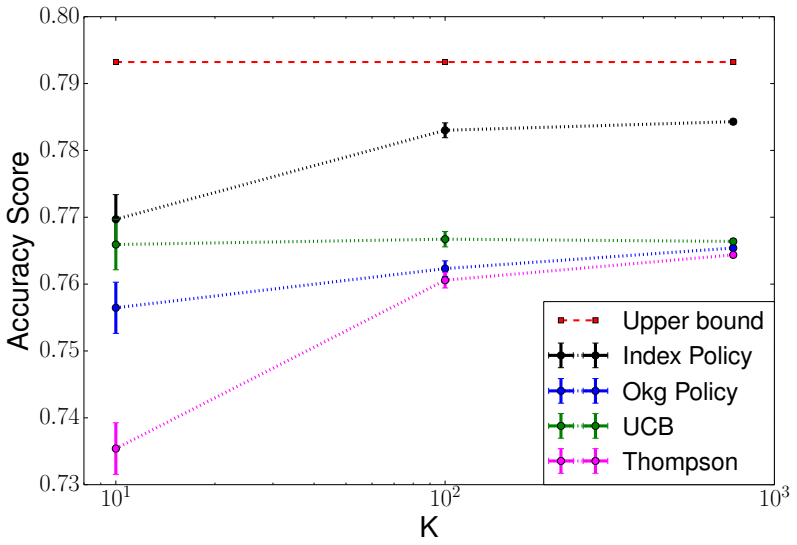

Figure 4.3: Semi-log plot of K against accuracy score for $K = 10, 100, 750$.

than the OKG policy. It is thus reasonable to anticipate that the Index Policy is not only consistent, but is asymptotically optimal when both the number of workers and the number of tasks goes to infinity, while keeping the ratio of the number of workers and the number of tasks constant.

4.8 Conclusion

We formulated the effort-allocation problem in crowdsourcing in a continuous time setting with budget constraint and time constraint. We also provide a computationally feasible upper bound on value of the Bayes-optimal policy using Lagrangian relaxation. Using the Lagrange multiplier used in proving the upper bound, we also derived an index-based policy and showed in numerical experiments that it performs close to optimal.

APPENDIX A

A.1 Notation of Chapter 2

$\mathbb{S}^K, \mathbb{A}^K, \mathbb{P}^{\cdot}(\cdot\|\cdot), R(\cdot,\cdot)$	State space, action space, transition kernal and reward function of the original MDP.
$\mathbb{S}, \mathbb{A}, P^{\cdot}(\cdot,\cdot), r(\cdot,\cdot)$	State space, action space, transition kernal and reward function of the sub-processes of the original MDP.
$\mathbf{s}$, $\mathbf{S}$	Generic element and random element of $\mathbb{S}^K$.
s, S	Generic element and random element of $\mathbb{S}$.
K	Number of sub-processes.
T	Time horizon
m_t	Number of sub-processes to be set active at time step t
$\mathbf{\Pi}$	Set of all Markov policies of the original MDP
$\boldsymbol{\pi}(\mathbf{s}, \mathbf{a}, t)$	the probability of choosing action $\mathbf{a}$ in state $\mathbf{s}$ under policy $\boldsymbol{\pi}$ at time t.
Π	Set of all Markov policies of the sub-MDP
$\pi(s, a, t)$	the probability of choosing action a in state s under policy π at time t.
$\Pi^*(\boldsymbol{\lambda})$	Set of Markov deterministic optimal policies for sub-MDP $Q(\boldsymbol{\lambda})$, given $\boldsymbol{\lambda} \in \mathbb{R}^T$.
$\pi^{\boldsymbol{\lambda}}$	An element in $\Pi^{\boldsymbol{\lambda}}$.
$\boldsymbol{\pi}^{\boldsymbol{\lambda}}$	A deterministic optimal policy for the relaxed problem which obtained by the decomposition method in Lemma 2, given $\boldsymbol{\lambda} \in \mathbb{R}^T$.
$\mathbf{P}(\boldsymbol{\lambda})$	Optimal value of the relaxed problem, given $\boldsymbol{\lambda} \in \mathbb{R}^T$.
$\boldsymbol{\lambda}^*$	An value that attains $\inf_{\boldsymbol{\lambda}} \mathbf{P}(\boldsymbol{\lambda})$

π^{**}	An optimal markov policy for the sub-MDP which satisfies $\mathbb{E}^{\pi^{**}}[A_t] = \frac{m}{K}, \forall 1 \leq t \leq T$.
$\hat{\boldsymbol{\pi}}$	The index based policy proposed by this paper
$\bar{\beta}_t$	Indices of the tied sub-processes.
I_t	The set of states occupied by the tied sub-processes.
$N_t(s)$	The number of sub-processes in state s at time t under index policy $\hat{\boldsymbol{\pi}}$.
$P_t(s)$	The probability of an individual sub-process landing in state s at time t under π^{**}. $P_t(s) = P^{\pi^{**}}[S_t = s]$
α	The ratio between the number of sub-processes set active, m, and the total number of sub-processes K.
$M_t(s)$	The number of sub-processes in state s at time t that are set active under our index policy $\hat{\boldsymbol{\pi}}$.
$Y_t(s', s)$	The number of sub-processes set active by $\hat{\boldsymbol{\pi}}$ in s' at time t which transition to state s at time $t+1$.
$X_t(s', s)$	The number of sub-processes set inactive by $\hat{\boldsymbol{\pi}}$ in s' at time t which transition to s at time $t+1$.
$U_t(s)$	The set of states whose indices are greater than the index of state s at time t. $U_t(s) = \{s'' \in \mathbb{S} : \beta_t(s'') > \beta_t(s)\}$.
$V_t(s)$	The set of states whose indices are equal to that of s. $V_t(s) = \{s'' \in \mathbb{S} : \beta_t(s'') = \beta_t(s)\}$.
$\lvert\mathbf{v}\rvert$	An operation that sums all the elements in vector $\mathbf{v}$.
H_1, H_2	random variables due to the rounding rules in Algorithm 2
$Z(\boldsymbol{\pi}, m, K)$	the expected reward of the original MDP obtained by policy $\boldsymbol{\pi}$

Table A.1: List of notation

A.2 Upper Bound

Proof. Proof of Lemma 1 Let $\mathbf{\Pi}_P = \{\boldsymbol{\pi} \in \mathbf{\Pi} : P^{\boldsymbol{\pi}}(|\mathbf{A}_t| = m_t) = 1,\ \forall 1 \leq t \leq T\}$. Let $\mathbf{\Pi}_E = \{\boldsymbol{\pi} \in \mathbf{\Pi} : \mathbb{E}^{\boldsymbol{\pi}}[|\mathbf{A}_t|] = m_t,\ \forall 1 \leq t \leq T\}$. For any $\boldsymbol{\lambda} \in \mathbb{R}^T$, we have

$$
\begin{aligned}
& P(\boldsymbol{\lambda}) \\
&= \max_{\boldsymbol{\pi} \in \mathbf{\Pi}} \mathbb{E}^{\boldsymbol{\pi}} \left[\sum_{t=1}^{T} R_t(\mathbf{S}_t, \mathbf{A}_t) \right] - \mathbb{E}^{\boldsymbol{\pi}} \left[\sum_{t} \lambda_t (|\mathbf{A}_t| - m_t) \right] \\
&\geq \max_{\boldsymbol{\pi} \in \mathbf{\Pi}_E} \mathbb{E}^{\boldsymbol{\pi}} \left[\sum_{t=1}^{T} R_t(\mathbf{S}_t, \mathbf{A}_t) \right] - \mathbb{E}^{\boldsymbol{\pi}} \left[\sum_{t} \lambda_t (|\mathbf{A}_t| - m_t) \right] \\
&= \max_{\boldsymbol{\pi} \in \mathbf{\Pi}_E} \mathbb{E}^{\boldsymbol{\pi}} \left[\sum_{t=1}^{T} R_t(\mathbf{S}_t, \mathbf{A}_t) \right] \\
&\geq \max_{\boldsymbol{\pi} \in \mathbf{\Pi}_P} \mathbb{E}^{\boldsymbol{\pi}} \left[\sum_{t=1}^{T} R_t(\mathbf{S}_t, \mathbf{A}_t) \right],
\end{aligned}
$$

which is the optimal value of the original MDP. The first inequality is due to $\mathbf{\Pi}_E \subseteq \mathbf{\Pi}$. The first equality is due to the fact that any policy $\boldsymbol{\pi}$ in $\mathbf{\Pi}_E$ satisfies $\mathbb{E}^{\boldsymbol{\pi}}[\mathbf{A}_t|] = m_t$. The last inequality is due to $\mathbf{\Pi}_P \subseteq \mathbf{\Pi}_E$. □

A.3 Decomposition

Proof. Proof of Lemma 2:

$$\begin{aligned}
&\max_{\boldsymbol{\pi}\in\boldsymbol{\Pi}} \mathbb{E}^{\boldsymbol{\pi}}\left[\sum_{t=1}^{T} R_t(\mathbf{S}_t, \mathbf{A}_t)\right] - \mathbb{E}^{\boldsymbol{\pi}}\left[\sum_{t=1}^{T} \lambda_t(|\mathbf{A}_t| - m_t)\right] \\
&= \max_{\boldsymbol{\pi}\in\boldsymbol{\Pi}} \mathbb{E}^{\boldsymbol{\pi}}\left[\sum_{t=1}^{T} R_t(\mathbf{S}_t, \mathbf{A}_t) - \lambda_t|\mathbf{A}_t|\right] + \sum_{t=1}^{T} m_t\lambda_t \\
&= \max_{\boldsymbol{\pi}\in\boldsymbol{\Pi}} \mathbb{E}^{\boldsymbol{\pi}}\left[\sum_{t=1}^{T}\sum_{x=1}^{K} r_t(S_{t,x}, A_{t,x}) - \lambda_t A_{t,x}\right] + \sum_{t=1}^{T} m_t\lambda_t \\
&= \sum_{x=1}^{K} \max_{\pi\in\Pi} \mathbb{E}^{\pi}\left[\sum_{t=1}^{T} r_t(S_t, A_t) - \lambda_t A_t\right] + \sum_{t=1}^{T} m_t\lambda_t
\end{aligned}$$

□

The first equality is due to linearity of expectation. The second equality is obtained by the definition of $r_t(\cdot,\cdot)$ and $|\cdot|$. The third equality is obtained by the independence of the process under policies in $\boldsymbol{\Pi}$.

A.4 Show $\arg\inf_{\boldsymbol{\lambda}\in\mathbb{R}^T} \mathbf{P}(\boldsymbol{\lambda})$ is non-empty

Proof. When $\boldsymbol{\lambda} \geq \mathbf{0}$, $\mathbf{P}(\boldsymbol{\lambda}) = \sum_x R_x(\boldsymbol{\lambda}) + m\sum_t \lambda_t \geq 0 + 0 = 0$. $R_x(\boldsymbol{\lambda})$ is bounded below by 0 since a policy of not playing at all gives a total reward of 0. When $\boldsymbol{\lambda} < \mathbf{0}$, the cost of playing is negative, an optimal policy will always play at all time steps. Hence $\mathbf{P}(\boldsymbol{\lambda}) \geq m(\sum_t(0 - \lambda_t)) + m\sum_t \lambda_t = 0$. For the case in which $\boldsymbol{\lambda}$ contains both positive and negative entries, writing $\boldsymbol{\lambda}$ as a convex combination of $\boldsymbol{\lambda}_1 > 0$ and $\boldsymbol{\lambda}_2 < 0$ and we have that $P(\boldsymbol{\lambda})$ is still bounded below by zero, since $P(\boldsymbol{\lambda})$ is convex in $\boldsymbol{\lambda}$. Hence we

can conclude that $\inf_{\boldsymbol{\lambda}\in\mathbb{R}^T} \mathbb{P}(\boldsymbol{\lambda})$ exists (note here we make no claim about whether this infimum is attained by any finite $\boldsymbol{\lambda}$) and denote this value by h^*.

Recall we have assumed in the setup that all the rewards are bounded and non-negative, let $\bar{r}$ be an upper bound for all the reward values. For any $\boldsymbol{\lambda}$ with $\lambda_t \geq T\bar{r}$, the corresponding optimal policies for a single-arm problem will be not play at time t, for $T\bar{r}$ is at least the maximum reward obtainable by the single-arm problem. Hence $\mathbf{P}(\boldsymbol{\lambda}) \geq 0 + mT\bar{r}$. For any $\boldsymbol{\lambda} \geq \mathbf{0}$, $\mathbf{P} = m\mathbb{E}[\sum_t r_{t,x}(S_{t,x}, 1) - \lambda_t | s_{1,x}] + m\sum_t \lambda_t = m\mathbb{E}[\sum_t r_{t,x}(S_{t,x}, 1) | s_{1,x}]$, which is independent of $\boldsymbol{\lambda}$. Hence the infimum is attained on the set $H = \{\boldsymbol{\lambda} : \lambda_t \geq \forall t \text{ and } \max_t \lambda_t \leq t\bar{r}\}$. Since H is compact, there exists a $\boldsymbol{\lambda}^* \in H$ s.t. $\mathbf{P}(\boldsymbol{\lambda}^*) = h^*$. Hence $\arg\inf_{\boldsymbol{\lambda}\in\mathbb{R}^T} \mathbf{P}(\boldsymbol{\lambda})$ is non-empty. □

A.5 Proof of the existence of π^{**}

Let $L(\pi, \boldsymbol{\lambda})$ be the Lagrangian of the sub-problem

$$\begin{aligned} \underset{\pi}{\text{maximize}} \quad & \mathbb{E}^{\pi}[\sum_{t=1}^{T} r_t(S_t, A_t)] \\ \text{subject to} \quad & \mathbb{E}^{\pi}[A_t] = \frac{m_t}{K}, \quad \forall 1 \leq t \leq T. \end{aligned}$$

Then

$$\max_{\pi} L(\pi, \boldsymbol{\lambda}) = \max_{\pi} \mathbb{E}^{\pi}[\sum_{t=1}^{T} r_t(S_t, A_t)] - \sum_{t=1}^{T} \lambda_t(\mathbb{E}^{\pi}[A_t] - \frac{m_t}{K}) = Q(\boldsymbol{\lambda}) + \sum_{t=1}^{T} \lambda_t \frac{m_t}{K}.$$

Let π^{**} be a policy that attains $Q(\boldsymbol{\lambda}^*)$. Hence

$$\begin{aligned} Q(\boldsymbol{\lambda}^*) + \sum_{t=1}^{T} \lambda_t^* \frac{m_t}{K} &= L(\pi^{**}, \boldsymbol{\lambda}^*) \\ &= \inf_{\boldsymbol{\lambda}} \max_{\pi} L(\pi, \boldsymbol{\lambda}) \\ &= \max_{\pi} \inf_{\boldsymbol{\lambda}} L(\pi, \boldsymbol{\lambda}) \\ &= \max_{\pi} \inf_{\boldsymbol{\lambda}} \mathbb{E}^{\pi}[\sum_{t=1}^{T} r_t(S_t, A_t)] - \sum_{t=1}^{T} \lambda_t(\mathbb{E}^{\pi}[A_t] - \frac{m_t}{K}). \end{aligned}$$

We are allowed to switch inf and max in the third equality because $L(\pi, \boldsymbol{\lambda})$ satisfies Slater's condition, which is implied by the existence of a feasible policy of $Q(\boldsymbol{\lambda})$. The fourth equality is just a re-write of what $L(\pi, \boldsymbol{\lambda})$ is. Since $Q(\boldsymbol{\lambda}^*) + \sum_{t=1}^{T} \lambda_t^* \frac{m_t}{K}$ is finite, $\max_{\pi} \inf_{\boldsymbol{\lambda}} L(\pi, \boldsymbol{\lambda})$ is finite. Any policy π that attains $\max_{\pi} \inf_{\boldsymbol{\lambda}} L(\pi, \boldsymbol{\lambda})$ has to satisfy that $\mathbb{E}^{\pi}[A_t] - \frac{m_t}{K}$ is zero for all t, otherwise $\inf_{\boldsymbol{\lambda}} L(\pi, \boldsymbol{\lambda})$ will be negative infinity. Hence $\mathbb{E}^{\pi^{**}}[A_t] = \frac{m_t}{K}$.

A.6 Proof of $T * \max_{s,a,t} r_t(s,a)$ upper bounds $\beta_t(s)$

It is sufficient to show that for any $\boldsymbol{\lambda} \geq \mathbf{0}$ with $\lambda_t > T * \max_{s,a,t} r_t(s,a)$, $V^{\boldsymbol{\lambda}}(s,t)$ is attained by choosing $a = 0$. When $t = T$, $r_T(s,1) - \lambda_T < r_T(s,1) - T * \max_{s,a,t} r_t(s,a) \leq 0$. On the other hand $r_T(s,0) \geq 0$ as all rewards are non-negative by the setting of our original MDP. Hence it is optimal to choose $a = 0$. When $t < T$, $r_t(s,1) - \lambda_t + \sum_{s' \in \mathbb{S}} P^a(s,s') V^{\boldsymbol{\lambda}}(s', t+1) < \max_{s,a,t} r_t(s,a) - T * \max_{s,a,t} r_t(s,a) + (T - t) * \max_{s,a,t} r_t(s,a) \leq 0 \leq r_t(s,0)$. Hence it is also optimal to choose $a = 0$. Therefore $\beta_t(s) \leq T * \max_{s,a,t} r_t(s,a)$ for all s, t.

A.7 A result that justifies using bisection

Lemma 13. *If there exists an optimal policy π that takes action $a = 1$ in state $s \in \mathbb{S}$ at time t for a sub-MDP $(\mathbb{S}, \mathbb{A}, r, P_\cdot)$, and satisfies $P^\pi[S_t = s] > 0$, then $a = 1$ is strictly optimal in state s at time t under a modified sub-MDP $(\mathbb{S}, \mathbb{A}, r', P_\cdot)$ with $r'_t(s, 1) > r_t(s, 1)$, and r'_t equals r_t otherwise.*

Proof. Proof of Lemma 13 We prove by contradiction. Let $V(\pi, r)$ denote the total expected reward obtained by policy π with reward function r. Assume that there exists an optimal policy π' for sub-MDP $(\mathbb{S}, \mathbb{A}, r'(\cdot, \cdot), P_\cdot(\cdot, \cdot))$ such that $\pi'(s, 0, t) = 1$. Since neither $r_t(s, 1)$ nor $r'_t(s, 1)$ contributes to the total expected reward, $V(\pi', r) = V(\pi', r')$. Let π be an optimal policy for $(\mathbb{S}, \mathbb{A}, r(\cdot, \cdot), P_\cdot(\cdot, \cdot))$. Then we have $V(\pi, r) \geq V(\pi', r)$. On the other hand, $V(\pi, r')$ is greater than $V(\pi, r)$ by $(r'_t(s, 1) - r_t(s, 1))\mathbb{P}^\pi[S_t = s] > 0$. Hence we get that $V(\pi, r') > V(\pi, r) \geq V(\pi', r) = V(\pi', r')$ contradicting that π' is an optimal policy of sub-MDP $(\mathbb{S}, \mathbb{A}, r'(\cdot, \cdot), P_\cdot(\cdot, \cdot))$. □ □

A.8 Proof of Lemma 3

Proof. Proof of Lemma 3 To prove (1), when $\beta_t(s) > \lambda_t^*$, by definition of the index in (2.6), there exists an $\epsilon > 0$ such that there is a $\pi \in \Pi^*(\boldsymbol{\lambda}^*[\lambda_t^* + \epsilon, t])$ and $\pi(s, 1, t) = 1$. Recall how we construct set $\Pi^*(\boldsymbol{\lambda})$ in Section 2.3.1, the value function $V^{\boldsymbol{\lambda}^*[\lambda_t^*+\epsilon,t]}$ corresponding to sub-MDP $Q(\boldsymbol{\lambda}^*[\lambda_t^* + \epsilon, t])$ has to satisfy

$$r_t(s, 1) - \lambda_t^* - \epsilon + \sum_{s' \in \mathbb{S}} V^{\boldsymbol{\lambda}^*[\lambda_t^*+\epsilon,t]}(s', t + 1)P^1(s, s')$$
$$\geq r_t(s, 0) + \sum_{s' \in \mathbb{S}} V^{\boldsymbol{\lambda}^*[\lambda_t^*+\epsilon,t]}(s', t + 1)P^0(s, s').$$

Since $\boldsymbol{\lambda}^*[\lambda_t^* + \epsilon, t]$ and $\boldsymbol{\lambda}^*$ share the same elements from the $(t+1)^{th}$ position onwards, $V^{\boldsymbol{\lambda}^*[\lambda_t^*+\epsilon,t]}(s,t') = V^{\boldsymbol{\lambda}^*}(s,t')$, for all $s \in \mathbb{S}$ and $t' \geq t+1$. Hence

$$r_t(s,1) - \lambda_t^* + \sum_{s' \in \mathbb{S}} V^{\boldsymbol{\lambda}^*}(s', t+1) P^1(s,s') > r_t(s,0) + \sum_{s' \in \mathbb{S}} V^{\boldsymbol{\lambda}^*}(s', t+1) P^0(s,s'). \quad \text{(A.1)}$$

Next we consider two separate cases: 1) State s is visited with positive probability under π^{**}, that is, $P^{\pi^{**}}(S_t = s) > 0$; 2) State s is visited with zero probability, i.e., $P^{\pi^{**}}(S_t = s) = 0$. If 1) $P^{\pi^{**}}(S_t = s) > 0$, since π^{**} is an optimal policy for the unconstrained sub-MDP $Q(\boldsymbol{\lambda}^*)$ in (2.3), and $a = 1$ attains $\max\{r_t(s,a) - a\lambda_t^* + \sum_{s' \in \mathbb{S}} P^a(s,s') V^{\boldsymbol{\lambda}^*}(s', t+1)\}$ alone, hence $\pi^{**}(s,1,t) = 1$. If 2) $P^{\pi^{**}}(S_t = s) = 0$, we get $\pi^{**}(s,1,t) = 1$ directly from the construction of π^{**} in (2.8).

Statement (2) can be proven using a similar argument. We therefore skip the proof to avoid redundancy. □ □

A.9 Proof of Lemma 4

Proof. Proof of Lemma 4 To prove (1), suppose, for the sake of contradiction, that $\pi^{**}(s,1,t) < 1$. By Lemma 3, we have $\beta_t(s) \leq \lambda_t^*$. Therefore $U_t(s) \cup V_t(s)$ forms a superset to the set of states with indices of at least λ_t^*. We also know that π^{**} takes active action with probability α at time t. Hence we can write α as the sum of the probabilities of taking the active action in all states s' with $\pi^{**}(s',1,t) = 1$ and the probabilities of taking the active action in all states s' with $0 < \pi^{**}(s',1,t) < 1$:

$$\begin{aligned} \alpha &= \sum_{s' \in \{s'' : \pi^{**}(s'',1,t)=1\}} P_t(s') * 1 + \sum_{s' \in \{s'' : 0 < \pi^{**}(s'',1,t) < 1\}} P_t(s') * \pi^{**}(s',1,t) \\ &< \sum_{s' \in \{s'' : \pi^{**}(s'',1,t)=1\}} P_t(s') + \sum_{s' \in \{s'' : 0 < \pi^{**}(s'',1,t) < 1\}} P_t(s'). \end{aligned} \quad \text{(A.2)}$$

Taking the contrapositives of both statements in Lemma 3, we get if $0 < \pi^{**}(s', 1, t) < 1$ then $\beta_t(s') = \lambda_t^*$. Hence

$$(\text{A.2}) = \sum_{s' \in \{s'' : \beta_t(s'') \geq 1\}} P_t(s') \leq \sum_{s' \in U_t(s) \cup V_t(s)} P_t(s') \leq \alpha$$

We get $\alpha < \alpha$, which is a contradiction, as desired.

To prove (2), we again use contradiction. Assume $\pi^{**}(s, 1, t) > 0$; by the contrapositive of the second statement of Lemma 3 we know $\beta_t(s) \geq \lambda_t^*$. Then $U_{t+1}(s)$ is a subset of $\{s' : \beta_t(s') > \lambda_t^*\}$, which in turn is a subset of $\{s' : \pi^{**}(s', 1, t) = 1\}$ by Lemma 3. Hence by the fact that $\alpha = \sum_{s' \in \{s'' : \pi^{**}(s'', 1, t) = 1\}} P_t(s') * 1 + \sum_{s' \in \{s'' : 0 < \pi^{**}(s'', 1, t) < 1\}} P_t(s') * \pi^{**}(s', 1, t)$, we must have either 1)

$$\alpha > \sum_{s' \in \{s'' : \pi^{**}(s'', 1, t) = 1\}} P_t(s') \geq \sum_{s' \in U_t(s)} P_t(s')$$

when there exists some $s' \in \{s'' : 0 < \pi^{**}(s'', 1, t) < 1\}$ such that $P_t(s') > 0$, or 2)

$$\alpha \geq \sum_{s' \in \{s'' : \pi^{**}(s'', 1, t) = 1\}} P_t(s') > \sum_{s' \in U_t(s)} P_t(s')$$

otherwise, as we must have that $\pi^{**}(s, 1, t) = 1$. In either case we get $\alpha > \sum_{s' \in U_t(s) = 1} P_t(s')$, which again forms a contradiction. □ □

A.10 Lemma 14

Lemma 14. *Let $X^{(k)}$ be a sequence of non-negative random variables such that $\lim_{k \to \infty} \frac{1}{k} X^{(k)} = \gamma$, a.s.. If $Y^{(k)} | X^{(k)} \sim Bin(X^{(k)}, p)$, then $\lim_{k \to \infty} \frac{Y^{(k)}}{k} = \gamma p$, a.s..*

Proof. Proof of Lemma 14 We consider two cases: 1) $X^{(k)} \to \infty$; 2) $X^{(K)}$ is bounded. When 1) $X^{(k)} \to \infty$, we have

$$\begin{aligned}
&\lim_{k\to\infty} \frac{Y^{(k)}}{k} \\
=&\lim_{k\to\infty} \frac{Y^{(k)}}{X^{(k)}} \frac{X^{(k)}}{k} \\
=&\lim_{k\to\infty} \frac{Y^{(k)}}{X^{(k)}} \lim_{k\to\infty} \frac{X^{(k)}}{k} \\
=&p * \gamma
\end{aligned}$$

If 2) $X^{(K)}$ is bounded, then $\gamma = 0$. $Y^{(k)}$ is also bounded, so $\lim_{k\to\infty} = 0$. □ □

A.11 Proof of Lemma 5

Proof. Proof of Lemma 5 Before attempting the proof, we make some observations on the properties of function Rounding-c. These observations will also be useful for proofs that come later in Appendix A.12. Based on how Rounding-c works in Algorithm 3, Figure A.11 plots the number of sub-processes allocated to the i^{th} state b_i against total $- \sum_{j\neq i} \min\{\text{total} * \text{frac}_j, \text{avail}_j\}$. b_i is a piecewise linear function of total $- \sum_{j\neq i} \min\{\text{total} * \text{frac}_j, \text{avail}_j\}$ with the changes of gradients occur when the value of some avail_j drops to $\text{total} * \text{frac}_j$. The function line intersect with y-axis at $\min\{\text{total} * \text{frac}_i, \text{avail}_i\}$ when there is no left-over after assigning each state $\min\{\text{total} * \text{frac}_j, \text{avail}_j\}$. The line ends when it reaches $\min\{\text{total}, \text{avail}_i\}$ as that is the largest possible assignment to state i.

To give an explicit form, we first sort for $j \neq i$ and $j \in \{1, ..., n\}$ by $c_j = \text{avail}_j - \text{total} * \text{frac}_j$. Let $c'_1, ..., c'_\ell$ be the sorted values of c_js from the smallest to the largest with no repetition, and let n_j denote the number of states with c'_j. Let x denote the value

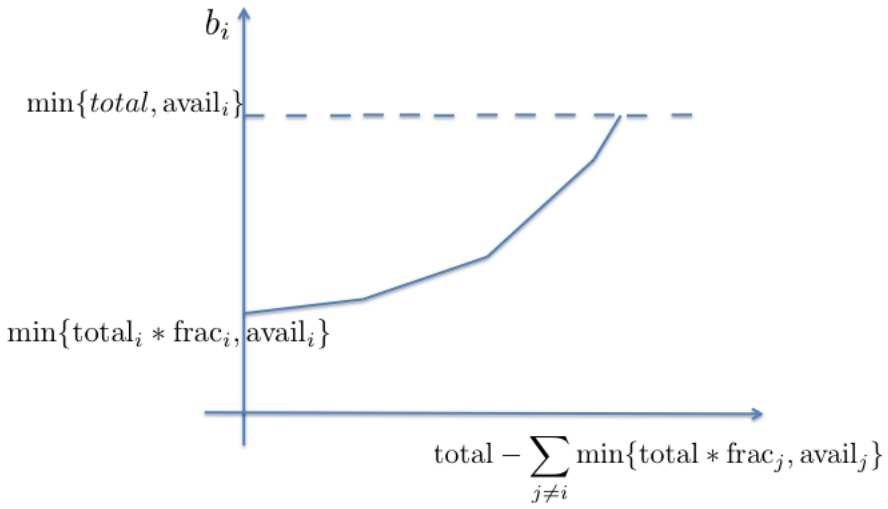

Figure A.1: Plot of Rounding-c

represented on the x-axis $\text{total} - \sum_{j \neq i} \min\{\text{total} * \text{frac}_j, \text{avail}_j\}$, we can write b_i as

$$b_i = \begin{cases} \min\{\text{total} * \text{frac}_i, \text{avail}_i\} + \frac{x}{n_1+\ldots+n_\ell+1} & \text{for } 0 \leq x \leq n_1 c'_1 \\ \min\{\text{total} * \text{frac}_i, \text{avail}_i\} + \frac{c'_1}{n1+\ldots+n_\ell} + \frac{x-n_1 c'_1}{n_2+\ldots+n_\ell+1} & \text{for } n_1 c'_1 \leq x \leq n_2 c'_2 \\ \vdots & \\ \min\{\text{total} * \text{frac}_i, \text{avail}_i\} + \frac{c'_1}{n1+\ldots+n_\ell} + \ldots + \frac{x-n_\ell c'_\ell}{n_\ell+1} & \text{for } n_\ell c'_\ell \leq x \leq \text{total} \end{cases} \tag{A.3}$$

Let b_i be the i^{th} component of the output by Rounding(total, frac, avail) and bc_i be the i^{th} component of the output by Rounding-c(total, frac, avail). We first show that $|b_i - bc_i| \leq 2$ by comparing the algorithms of Rounding and Rounding-c. If $\min\{\text{total} * \text{frac}_i, \text{avail}_i\}$ is attained by avail_i, then $b_i = bc_i$. If avail_i is larger than $\lfloor \text{total} * \text{frac}_i \rfloor$ (hence also larger than $\text{total} * \text{frac}_i$), and $total = \sum_j \min\{\lfloor \text{total} * \text{frac}_j \rfloor, \text{avail}_j\}$ (meaning no residue left to distribute in Rounding), then the difference between b_i and bc_i is no

more than 1. When $total > \sum_j \min\{\lfloor \text{total} * \text{frac}_j, \text{avail}_j \rfloor\}$ and there is residue left to be distributed, the amount distributed to the i^{th} state in Rounding may be smaller or larger than the that in Rounding-c, depending on the position of i in an array of 1 to n. This together with the possible difference between $\lfloor \text{total} * \text{frac}_j \rfloor$ and $\text{total} * \text{frac}_j$ gives a total difference of at most 2.

Subsequently we have $|b_s(\mathbf{N}_t, \lfloor \alpha_t K \rfloor) - \bar{b}_s(\mathbf{N}_t, \lfloor \alpha_t K \rfloor)| \leq 2$, and $|f_s(\mathbf{N}_t, \lfloor \alpha_t K \rfloor) - \bar{f}_s(\mathbf{N}_t, \lfloor \alpha_t K \rfloor)| = |f_s(\mathbf{N}_t, \lfloor \alpha_t K \rfloor) - K * \bar{f}_s(\frac{\mathbf{N}_t}{K}, \frac{\lfloor \alpha_t K \rfloor)}{K}|$. □

A.12 Proof of Lemma 6

Proof. Proof of Lemma 6 First note that $\bar{b}_s(\mathbf{N}_t, \lfloor \alpha_t K \rfloor)$ is continuous as illustrated in the proof of Lemma 5 in Appendix A.11. Hence $\bar{f}_s(\mathbf{N}_t, \lfloor \alpha_t K \rfloor)$ is continuous. This is because when $\lfloor \alpha_t K \rfloor - N_t^+(s) \neq N_t^=(s)$, $\bar{f}_s(\mathbf{N}_t, \lfloor \alpha_t K \rfloor)$ is continuous as $N_t(s)$ and $\bar{b}_s(\mathbf{N}_t, \lfloor \alpha_t K \rfloor)$ are continuous. When $\lfloor \alpha_t K \rfloor - N_t^+(s) = N_t^=(s)$, we have $N_t(s) = \bar{b}_s(\mathbf{N}_t, \lfloor \alpha_t K \rfloor)$. By the continuous mapping theorem, and the fact that $\lim_{K\to\infty} \frac{N_t(s)}{K} = P_t(s)$ almost surely and $\lim_{K\to\infty} \frac{\lfloor \alpha_t K \rfloor}{K} = \alpha_t$, we have

$$\lim_{K\to\infty} \bar{f}_s(\frac{\mathbf{N}_t}{K}, \frac{\lfloor \alpha_t K \rfloor}{K}) = \bar{f}_s(\lim_{K\to\infty} \frac{\mathbf{N}_t}{K}, \lim_{K\to\infty} \frac{\lfloor \alpha_t K \rfloor}{K}) = \bar{f}_s(\mathbf{P}_t, \alpha_t), \quad a.s.$$

□

A.13 Proof of Lemma 7

Proof. Proof of Lemma 7 We first prove that $\bar{f}_s(\mathbf{N}_t, \lfloor \alpha_t K \rfloor)$ equals to

$$g_s(\mathbf{P}_t) = \begin{cases} \min\{P_t(s), [\alpha - \sum_{s' \in U_t(s)} P_t(s')]^+ \frac{\rho(s,1,t)}{\sum_{s' \in V_t(s)} \rho(s',1,t)}\}, \\ \quad \text{if } \sum_{s' \in V_t(s)} \rho(s',1,t) > 0 \\ \min\{P_t(s), [\alpha - \sum_{s' \in U_t(s)} P_t(s')]^+ \frac{P_t(s)}{\sum_{s' \in V_t(s)} P_t(s')}\}, \\ \quad \text{if } \sum_{s' \in V_t(s)} \rho(s',1,t) = 0, \end{cases} \tag{A.4}$$

then we show $g_s(\mathbf{P}_t) = P_t(s)\pi^{**}(s,1,t)$. To prove the former, We divide our discussion into two main cases.

- When

$$[\alpha - \sum_{s' \in U_t(s)} P_t(s')]^+ - \sum_{s' \in V_t(s)} P_t(s') \geq 0, \tag{A.5}$$

  if $\alpha - \sum_{s' \in U_t(s)} P_t(s') \leq 0$, then $g_s(\mathbf{P}_t) = 0$ in both cases. Moreover, (A.5) implies $P_t(s) = 0$ by Lemma 4, hence $f_s(\mathbf{P}_t, \alpha) = 0$.

  If $\alpha - \sum_{s' \in U_t(s)} P_t(s') > 0$, (A.5) implies that $P_t(s) = 1$ by Lemma 4.

$$\alpha - \sum_{s' \in U_t(s)} P_t(s') \geq \sum_{s' \in V_t(s)} P_t(s') = \sum_{s' \in V_t(s)} \rho(s',1,t) \geq P_t(s).$$

  Hence $P_t(s)$ attains the minimum in both cases of $g_s(\mathbf{P}_t)$, and $f_s(\mathbf{P}_t, \alpha) = P_t(s) = g_s(\mathbf{P}_t)$.

- When

$$[\alpha - \sum_{s' \in U_t(s)} P_t(s')]^+ - \sum_{s' \in V_t(s)} P_t(s') < 0, \tag{A.6}$$

  if $\alpha - \sum_{s' \in U_t(s)} P_t(s') \leq 0$, then both $f_s(\mathbf{P}_t, \alpha)$ and $g_s(\mathbf{P}_t)$ are zero.

  If $\alpha - \sum_{s' \in U_t(s)} P_t(s') > 0$, $f_s(\mathbf{P}_t, \alpha)$ is either

  $\min\{(\alpha - \sum_{s' \in U_t(s)} P_t(s')) \frac{\rho(s,1,t)}{\sum_{s' \in V_t(s)\rho(s',1,t)}}, P_t(s)\}$ or

$\min\{(\alpha - \sum_{s' \in U_t(s)} P_t(s')) \frac{P_t(s)}{\sum_{s' \in V_t(s)} P_t(s')}, P_t(s)\}$ depending on whether $\sum_{s' \in V_t(s)} \rho(s', 1, t)$ is greater than or equal to zero, which matches exactly the two cases in $g_s(\mathbf{P}_t)$.

We first consider the case when
$\sum_{s' \in V_t(s)} \rho(s', 1, t) > 0$. This can be further divided into two sub-cases,

- Case 1: When $g_s(\mathbf{P}_t) = P_t(s)$,

  if $\alpha - \sum_{s' \in U_t(s)} P_t(s') \leq 0$, by Lemma 4, $\pi^{**}(s, 1, t) = 0$. Since $P_t(s)$ attains the minimum in this case, $P_t(s) = 0$. Hence $g_s(\mathbf{P}_t) = 0 = P_t(s)\pi^{**}(s, 1, t)$.

  If $\alpha - \sum_{s' \in U_t(s)} P_t(s') > 0$, this leads to two cases by Lemma 4: $\pi^{**}(s, 1, t) = 1$ or $0 < \pi^{**}(s, 1, t) < 1$. If it is the latter, we have the second term in $g_s(\mathbf{P}_t)$ becomes $\rho(s, 1, t)$, since $\alpha - \sum_{s' \in U_t(s)} P_t(s')$ cancels with the denominator. $\rho(s, 1, t) = P_t(s)\pi^{**}(s, 1, t) < P_t(s)$, which contradicts that $P_t(s)$ attains the minimum. Hence $\pi^{**}(s, 1, t) = 1$. We have $g_s(\mathbf{P}_t) = P_t(s) = P_t(s) * \pi^{**}(s, 1, t)$.

- Case 2: When $g_s(\mathbf{P}_t) = [\alpha - \sum_{s' \in U_t(s)} P_t(s')]^+ \frac{\rho(s,1,t)}{\sum_{s' \in V_t(s)} \rho(s',1,t)}$,

  if $\alpha - \sum_{s' \in U_t(s)} P_t(s') \leq 0$, again by Lemma 4, $\pi^{**}(s, 1, t) = 0$, $g_s(\mathbf{P}_t) = 0 = P_t(s)\pi^{**}(s, 1, t)$.

  If $\alpha - \sum_{s' \in U_t(s)} P_t(s') > 0$, again we have two cases: $\pi^{**}(s, 1, t) = 1$ or $0 < \pi^{**}(s, 1, t) < 1$. If it is the latter, we have $\alpha - \sum_{s' \in U_t(s)} P_t(s') = \sum_{s' \in V_t(s)} P_t(s')$. If it is the former, by assumption
  $[\alpha - \sum_{s' \in U_t(s)} P_t(s')] \frac{P_t(s)}{\sum_{s' \in V_t(s)} P_t(s')}$ attains the minimum, $\alpha - \sum_{s' \in U_t(s)} P_t(s') \leq \sum_{s' \in V_t(s)} P_t(s')$. Hence $\alpha - \sum_{s' \in U_t(s)} P_t(s')$ can only be equal to $\sum_{s' \in V_t(s)} P_t(s')$. Subsequently for both cases $\pi^{**}(s, 1, t) = 1$ and $0 < \pi^{**}(s, 1, t) < 1$, we have $g_s(\mathbf{P}_t) = \rho(s, 1, t) = P_t(s)\pi^{**}(s, 1, t)$.

Now we look at the case when $\sum_{s' \in V_t(s)} \rho(s', 1, t) = 0$. We have either 1) $P_t(s') = 0$ for all $s' \in V_t(s)$ 2) $\pi^{**}(s', 1, t) = 0$ for all $s' \in V_t(s)$.

When $\pi^{**}(s', 1, t) = 0$ for all $s' \in V_t(s)$, $\alpha - \sum_{s' \in U_t(s)} P_t(s') \leq 0$ hence $g_s(\mathbf{P}_t) = 0 = P_t(s)\pi^{**}(s, 1, t)$. □

A.14 Bellman's recursion for $T = \infty$

Below is the Bellmans's recursion on value functions for a single-task system when $T = \infty$:

if $l = U$ and $w_x = 0$:

$$V_x(\alpha_x, \beta_x, w_x, l) = R(\alpha_x, \beta_x); \tag{A.7a}$$

if $l = U$ and $w_x > 0$:

$$\begin{aligned} & V_x(\alpha_x, \beta_x, w_x, l) \\ =& \frac{\alpha_x}{\alpha_x + \beta_x} V_x(\alpha_x + 1, \beta_x, w_x - 1, l) + \\ & \frac{\beta_x}{\alpha_x + \beta_x} V_x(\alpha_x, \beta_x + 1, w_x - 1, l); \end{aligned} \tag{A.7b}$$

If $l < U$:

$$\begin{aligned} & V_x(\alpha_x, \beta_x, w_x, l) \\ =& \frac{r}{q_x} \max_{a_{l,x} \in \{0,1\}} \{V_x(\alpha_x, \beta_x, w_x + a_{l,x}, l + 1) - \lambda_{l+1} a_{l,x}\} \\ & + \frac{\mu w_x}{q_x} \Big[\frac{\alpha_x}{\alpha_x + \beta_x} V_x(\alpha_x + 1, \beta_x, w_x - 1, l) \\ & \qquad + \frac{\beta_x}{\alpha_x + \beta_x} V_x(\alpha_x, \beta_x + 1, w_x - 1, l) \Big]. \end{aligned}$$

BIBLIOGRAPHY

[1] D Adelman and A Mersereau. Relaxations of Weakly Coupled Stochastic Dynamic Programs. *Operations Research*, 2008.

[2] S Andradóttir and S H Kim. Fully Sequential Procedures for Comparing Constrained Systems via Simulation. *Naval Research Logistics*, 57(5):403–421, 2010.

[3] P Auer, N Cesa-Bianchi, and P Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 2002.

[4] Y Bacharach, T Grapple, T Minka, and J Guiver. How To Grade A Test Without Knowing the Answers - A Bayesian Graphical Model for Adaptive Crowdsourcing And Aptitude Testing. *ICML*, 2013.

[5] D Batur and S H Kim. Finding Feasible Systems in the Presence of Constraints on Multiple Performance Measures. *ACM Transactions on Modeling and Computer Simulation*, 20(3):13:1–13:26, 2010.

[6] D A Berry and B Fristedt. Bandit problems: sequential allocation of experiments (Monographs on statistics and applied probability). *Springer*, 1985.

[7] E Bofinger and G J Lewis. Two-Stage Procedures for Multiple Comparisons with a Control. *American Journal of Mathematical and Management Sciences*, 12(4):253–275, 1992.

[8] O Chapelle and L Li. An Empirical Evaluation of Thompson Sampling. *NIPS*, 2002.

[9] C Chen, D He, M Fu, and L Lee. Efficient Simulation Budget Allocation for Selecting an Optimal Subset. *Informs Journal on Computing*, 2008.

[10] X Chen, Q Lin, and D Zhou. Statistical Decision Making for Optimal Budget Allocation in Crowd Labeling. *arXiv*, 2011.

[11] Xi Chen, Qihang Lin, and D Y Zhou. Optimistic Knowledge Gradient Policy for Optimal Budget Allocation in Crowdsourcing. *ICML*, 2013.

[12] Xi Chen, Qihang Lin, and Dengyong Zhou. Optimistic Knowledge Gradient Policy for Optimal Budget Allocation in Crowdsourcing. In *Proceedings of the 30th International Conference on Machine Learning*, pages 64–72, 2013.

[13] H Damerdji and M K Nakayama. Two-Stage Procedures for Multiple Comparisons with a Control in Steady-State Simulations. In *Proceedings of the 1996 Winter Simulation Conference*, pages 372–375, Piscataway, New Jersey, 1996. Institute of Electrical and Electronics Engineers, Inc.

[14] M H DeGroot. *Optimal Statistical Decisions*. McGraw Hill, New York, 1970.

[15] E J Dudewicz and S R Dalal. Multiple Comparisons with a Control When Variances Are Unknown and Unequal. *American Journal of Mathematics and Management Sciences*, 3(4):275–295, 1983.

[16] C W Dunnett. A Multiple Comparison Procedure for Comparing Several Treatments with a Control. *Journal of the American Statistical Association*, 50(272):1096–1121, 1955.

[17] E B Dynkin and A A Yushkevich. *Controlled Markov Processes*. Springer, New York, 1979.

[18] E.B Dynkin and Yushkevish A.A. *Controlled Markov Processes*. Springer, 1979.

[19] P I Frazier. Learning with dynamic programming. In *Wiley Encyclopedia of Operations Research and Management Science*, pages 1–13, Hoboken, New Jersey, 2011. Wiley.

[20] A Ghosh, S Kale, and P McAfee. Who moderates the moderators? Crowdsourcing Abuse detection in user-generated content. *ACM*, 2011.

[21] C Gittins, J. Bandit Processes and Dynamic Allocation Indices. *Journal of the Royal Statistical Society. Series B*, 1979.

[22] J C Gittins. *Multi-Armed Bandit Allocation Indices*. John Wiley and Sons, New York, 1989.

[23] John Gittins, Kevin Glazebrook, and Richard Weber. *Multi-armed Bandit Allocation Indices*. Wiley, Hoboken, New Jersey, 2nd edition, 2011.

[24] Y Gocgun and A Ghatey. Lagrangian Relaxation and Constraint Generation For Allocation And Advanced Scheduling. *Computer and Operations Research*, 2011.

[25] J Hawkins. A Lagrangian decomposition approach to weakly coupled dynamic optimization problems and its applications. *Ph.D Thesis, Center of Operations Research, MIT*, 2002.

[26] Christopher Healey, Sigrún Andradóttir, and Seong-Hee Kim. Selection procedures for simulations with multiple constraints under independent and correlated sampling. *ACM Transactions on Modeling and Computer Simulation*, 24(3):14:1–14:25, 2014.

[27] CJ Ho, S Jabbari, and J W Vaughan. Adaptive Task Assignment For Crowdsourced classification. *ICML*, 2013.

[28] D Karger, S Oh, and D Shah. Budget-Optimal Task Allocation for Reliable Crowdsourcing Systems. *Operations Research*, 2013.

[29] D R Karger, S Oh, and D Shah. Iterative Learning for Reliable Crowdsourcing System. *NIPS*, 2011.

[30] D R Karger, S Oh, and D Shah. Efficient crowdsourcing for multi-class labeling. *ACM*, 2013.

[31] S H Kim. Comparison with a Standard via Fully Sequential Procedures. *ACM Transactions on Modeling and Computer Simulation*, 15(2):155–174, 2005.

[32] S H Kim and B L Nelson. Recent Advances in Ranking and Selection. In *Proceedings of the 2007 Winter Simulation Conference*, pages 162–172, Piscataway, New Jersey, 2007. Institute of Electrical and Electronics Engineers, Inc.

[33] L. W. Koenig and A. M. Law. A procedure for selecting a sub- set of size m containing the l best of k independent normal populations. *Comm. Statist.Simulation Comm. 14 719734*, 1985.

[34] W S Lovejoy. A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research*, 28(1):47–65, 1991.

[35] G E Monahan. A survey of partially observable Markov decision processes: Theory, models, and algorithms. *Management Science*, 28(1):1–16, 1982.

[36] B L Nelson and D Goldsman. Comparisons with a Standard in Simulation Experiments. *Management Science*, 47(3):449–463, 2001.

[37] E Paulson. On the Comparison of Several Experimental Categories with a Control. *The Annals of Mathematical Statistics*, 23(2):239–246, 1952.

[38] W B Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*. John Wiley and Sons, New York, 2007.

[39] Warren B Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Wiley, 2011.

[40] M.L Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 2008.

[41] H Robbins. A note on gambling systems and birth statistics. *American Mathematical Monthly*, 1952.

[42] A Ruszczynski. *Nonlinear Optimization*. Princeton University Press, 206.

[43] R Snow, B OConnor, D Jurafsky, and A Ng. Cheap and Fast But is it Good? Evaluating Non-Expert Annotations for Natural Language Tasks. *EMNLP*, 2008.

[44] R Szechtman and E Yücesan. A New Perspective on Feasibility Determination. In *Proceedings of the 2008 Winter Simulation Conference*, pages 273–280, Piscataway, New Jersey, 2008. Institute of Electrical and Electronics Engineers, Inc.

[45] T L Thanh, M Venanzi, A Rogers, and N R Jennings. Efficient budget allocation with accuracy guarantees for crowdsourcing classification tasks. *ACM*, 2013.

[46] R Weber and G Weiss. On An Index Policy For Restless Bandits. *Journal of Applied Probability*, 1990.

[47] R Weber, R and G Weiss. Addendum to 'On An Index Policy For Restless Bandit'. *Advances in Applied Probability*, 1991.

[48] P. Whittle. Restless Bandits: Activity Allocation in a Changing World. *Journal of Applied Probability*, 1988.

[49] J Xie and P I Frazier. Sequential Bayes-Optimal Policies for Multiple Comparisons with a Known Standard. *Operations Research*, 61(3):1174–1189, 2013.

[50] J. Xie and P.I. Frazier. Upper Bounds on the Bayes-Optimal Procedure for Ranking & Selection with Independent Normal Priors. *Proceedings of the 2013 Winter Simulation Conference*, 2013.

[51] Y Yan, R Rosales, G Fung, and J Dy. Active Learning From crowds. *ICML*, 2011.

[52] F Ye, H Zhu, and E Zhou. Weakly Coupled Dynamic Program: Information and Lagrangian Relaxations. *arXiv preprint arXiv:1405.3363*, 2014.