

Bayes-Optimal Effort Allocation in Crowdsourcing: Bounds and Index Policies

Weici Hu
wh343@cornell.edu

School of Operations Research & Information Engineering
Cornell University
206 Rhodes Hall
Ithaca, NY 14853, USA

Peter I. Frazier
pf98@cornell.edu

Abstract

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 Adelman & Mersereau (2008), 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 other state-of-arts and performs close to optimal solution.

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 (Karger et al., 2011). 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 paper, 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 paper 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 variability 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 be-

haviors 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 (Powell, 2007) 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 Adelman & Mersereau (2008) and Hawkins (2002). The proofs we present in Section 5 are very similar in spirit to Adelman & Mersereau (2008), 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 Adelman & Mersereau (2008) and Hawkins (2002). 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 Gittins (1989) and the Whittle index policy for restless bandits Whittle (1988). 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 paper 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 in 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 (Yan et al., 2011), (Chen et al., 2013), (Karger et al., 2013a). 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.

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 Karger et al. (2013b); Ghosh et al. (2011); Karger et al. (2011); Thanh et al. (2013); Ho et al. (2013). Karger et al. (2013b) 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. Thanh et al. (2013) 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(Yan et al. (2011); Bacharach et al. (2013)). 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 Chen et al. (2011) 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 Lovejoy (1991); Monahan (1982). Our use of Lagrangian relaxation is an application of the relaxation method of weakly coupled dynamic program discussed in (Adelman & Mersereau, 2008). The setting in this paper 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 paper, which uses Lagrangian Multipliers to assign indices, draws inspiration from Whittle (1988).

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 $z_\ell \in \{1, \dots, K\}$ indicate the task assigned to the ℓ^{th} worker. (We use $[K]$ to denote $\{1, \dots, K\}$ 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 Chen et al. (2013)), 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 $\theta_1, \dots, \theta_K$ and z_ℓ is

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

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 (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 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 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

$$R(\boldsymbol{\alpha}', \boldsymbol{\beta}') = \sum_{x=1}^K R(\alpha'_x, \beta'_x), \quad (2)$$

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 Dynamic Programming Formulation

We now formalize the problem statement from Section 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 $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_K)$ and $\boldsymbol{\beta} = (\beta_1, \dots, \beta_K)$ that will describe the posterior distribution on each θ_x given the labels observed thus far (θ_x will be distributed according to $\text{Beta}(\alpha_x, \beta_x)$ under this posterior).
- a length- K vector $\mathbf{w} = (w_1, \dots, w_K)$ 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 $s = (\boldsymbol{\alpha}, \boldsymbol{\beta}, t, \mathbf{w}, \ell)$ 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

$$\Pi_0 = \{\pi \in \Pi : |\pi(s, \Delta)| \leq 1, \forall s \in \mathbb{S}, t \in \mathbb{R}\}, \quad (3)$$

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 ($S_n : n \in \{0, 1, \dots\}$) 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 then 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 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

$$\sup_{\pi \in \Pi_0} \mathbb{E}^\pi [R(\boldsymbol{\alpha}(S_N), \boldsymbol{\beta}(S_N))]. \quad (4)$$

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

$$V(s) = \sup_{\pi \in \Pi_0} \mathbb{E}^\pi [R(\boldsymbol{\alpha}(S_N), \boldsymbol{\beta}(S_N)) | S_0 = s], \quad (5)$$

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 $\ell < U$:

$$\begin{aligned} V(s) = & \left(1 - \exp(-q(T-t))\right) \cdot \left[\right. \\ & r \int_0^{T-t} \max_z V(\boldsymbol{\alpha}, \boldsymbol{\beta}, t+y, \mathbf{w} + \mathbf{e}_z, \ell+1) e^{-qy} dy + \\ & \sum_{x=1}^K \mu w_x \left(\frac{\alpha_x}{\alpha_x + \beta_x} \int_0^{T-t} V(\boldsymbol{\alpha} + \mathbf{e}_x, \boldsymbol{\beta}, t+y, \mathbf{w} - \mathbf{e}_x, \ell) e^{-qy} dy \right. \\ & \left. \left. + \frac{\beta_x}{\alpha_x + \beta_x} \int_0^{T-t} V(\boldsymbol{\alpha}, \boldsymbol{\beta} + \mathbf{e}_x, t+y, \mathbf{w} - \mathbf{e}_x, \ell) e^{-qy} dy \right) \right] \\ & \left. + \exp(-q(T-t)) \left\{ R(\boldsymbol{\alpha}, \boldsymbol{\beta}) \right\} \right]. \end{aligned} \quad (6)$$

If $\ell \geq U$:

$$\begin{aligned} V(s) = & \left(1 - \exp(-q(T-t))\right) \cdot \left[r \int_0^{T-t} V(\boldsymbol{\alpha}, \boldsymbol{\beta}, t+y, \mathbf{w}, \ell) e^{-qy} dy + \right. \\ & \sum_{x=1}^K \mu w_x \left(\frac{\alpha_x}{\alpha_x + \beta_x} \int_0^{T-t} V(\boldsymbol{\alpha} + \mathbf{e}_x, \beta, t+y, \mathbf{w} - \mathbf{e}_x, \ell) e^{-qy} dy \right. \\ & \left. \left. + \frac{\beta_x}{\alpha_x + \beta_x} \int_0^{T-t} V(\boldsymbol{\alpha}, \beta + \mathbf{e}_x, t+y, \mathbf{w} - \mathbf{e}_x, \ell) e^{-qy} dy \right) \right] \\ & + \exp(-q(T-t)) \{R(\boldsymbol{\alpha}, \boldsymbol{\beta})\}. \end{aligned} \quad (7)$$

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.

5 Upper Bound on the Bayes-Optimal Policy

Although solving (4) directly using the stochastic dynamic program (6),(7) is computationally intractable, in this section we show how to obtain a computationally feasible upper bound on the value (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 (inclusive), 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 :

$$\Pi_1 = \left\{ \pi \in \Pi : \mathbb{E}^\pi \left[\sum_{x=1}^K a_{\ell,x} \right] \leq 1 \forall \ell \right\}. \quad (8)$$

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) is then,

$$R_0 = \sup_{\pi \in \Pi_0} \mathbb{E}^\pi [R], \quad (9)$$

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

$$R_1 = \sup_{\pi \in \Pi_1} \mathbb{E}^\pi [R]. \quad (10)$$

Here we introduce a non-negative vector $\boldsymbol{\lambda} = \{\lambda_1, \dots, \lambda_U\} \geq \mathbf{0}$, 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 1. *The term*

$$\inf_{\boldsymbol{\lambda} \geq \mathbf{0}} \sup_{\pi \in \Pi} \mathbb{E}^\pi \left[R - \sum_{\ell=1}^U (\lambda_\ell \sum_x a_{\ell,x}) \right] + \sum_{\ell=1}^U \lambda_\ell \quad (11)$$

forms an upper bound to R_0 .

Proof.

$$\begin{aligned} & \sup_{\pi \in \Pi} \mathbb{E}^\pi \left[R - \sum_{\ell=1}^U (\lambda_\ell \sum_x a_{\ell,x}) \right] + \sum_{\ell=1}^U \lambda_\ell \\ &= \sup_{\pi \in \Pi} \mathbb{E}^\pi \left[R - \sum_{\ell=1}^U \left(\lambda_\ell (\sum_x a_{\ell,x} - 1) \right) \right] \\ &\geq \sup_{\pi \in \Pi_1} \mathbb{E}^\pi \left[R - \sum_{\ell=1}^U \left(\lambda_\ell (\sum_x a_{\ell,x} - 1) \right) \right] \\ &= \sup_{\pi \in \Pi_1} \mathbb{E}^\pi [R] - \sum_{\ell=1}^U \lambda_\ell \left(\mathbb{E}^\pi [\sum_x a_{\ell,x}] - 1 \right) \\ &\geq \sup_{\pi \in \Pi_1} \mathbb{E}^\pi [R] \\ &\geq \sup_{\pi \in \Pi_0} \mathbb{E}^\pi [R] \end{aligned} \quad (12)$$

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 (12) holds true for any value of $\boldsymbol{\lambda} > \mathbf{0}$, we obtain Theorem 1. \square

Calculating the supremum term in Theorem 1 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 $S_n = (S_{n,1}, \dots, S_{n,K})$, 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, 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 1 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 2.

$$\inf_{\lambda \geq 0} \sum_{x=1}^K \sup_{\pi^{(x)} \in \Pi^{(x)}} \mathbb{E}^{\pi^{(x)}} \left[R_x - \sum_{\ell=1}^U \lambda_\ell a_{\ell,x} \right] + \sum_{\ell=1}^U \lambda_\ell \quad (13)$$

forms an upper bound on R_0 .

Proof. For any $\lambda \geq 0$:

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

This is bounded above by,

$$\begin{aligned} & \sum_{x=1}^K \sup_{\pi \in \Pi} \mathbb{E}^\pi \left[R_x - \sum_{\ell=1}^U \lambda_\ell a_{\ell,x} \right] \\ &= \sum_{x=1}^K \sup_{\pi^{(x)} \in \Pi^{(x)}} \mathbb{E}^{\pi^{(x)}} \left[R_x - \sum_{\ell=1}^U \lambda_\ell a_{\ell,x} \right]. \end{aligned} \quad (14)$$

The equality at (14) is because $\sup_{\pi \in \Pi} \mathbb{E}^\pi \left[R_x - \sum_{\ell=1}^U \lambda_\ell a_{\ell,x} \right]$ 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 1, for any $\lambda \geq 0$,

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

forms an upper bound on R_0 . This hold for any $\lambda \geq 0$, and so we have thus proved Theorem 2. \square

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

$$\sup_{\pi^{(x)} \in \Pi^{(x)}} \mathbb{E}^{\pi^{(x)}} \left[R_x - \sum_{\ell=1}^U \lambda_\ell a_{\ell,x} \right], \quad (15)$$

for any λ value. What remains in the computing of the upper bound is to solve for the infimum in Theorem 2. We explore the convexity property of the problem follow by a binary search. Define $B(\lambda) = \sum_{x=1}^K \sup_{\pi^{(x)} \in \Pi^{(x)}} \mathbb{E}_0^{\pi^{(x)}} \left[R_x - \sum_{\ell=1}^U \lambda_\ell a_{\ell,x} \right] + \sum_{\ell=1}^U \lambda_\ell$, which is the upper bound derived in Theorem 2 without the infimum. First we prove that $B(\lambda)$ is convex in λ .

Lemma 1. $B(\lambda)$ is convex in λ .

Proof. First note $\sum_{\ell=1}^U \lambda_\ell$ is convex in λ . To prove $\sup_{\pi^{(x)} \in \Pi^{(x)}} \mathbb{E}^{\pi^{(x)}} \left[R_x - \sum_{\ell=1}^U \lambda_\ell a_{\ell,x} \right]$ is convex in λ , pick any $\lambda_1, \lambda_2 \geq 0$ and $t \in [0, 1]$. Let

$$\pi' = \sup_{\pi^{(x)} \in \Pi^{(x)}} \mathbb{E}^{\pi^{(x)}} \left[R_x - \sum_{\ell=1}^U (t\lambda_{1,\ell} + (1-t)\lambda_{2,\ell}) a_{\ell,x} \right] \quad (16)$$

We have

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

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

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

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 (15) with $\lambda = \lambda 1$, $1 = (1, \dots, 1)$. 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 (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 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 (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

Algorithm 1 Index Policy

```

1: while  $\ell < U$  do
2:   For each  $x \in \{1, \dots, K\}$ , compute  $\lambda_x^* = \inf\{\lambda \in \mathbb{R}_+ : a_{\ell,x}^\lambda = 1\}$ , where  $a_{x,\ell}^\lambda$  is the optimal decision from (15) when  $\lambda = \lambda 1$ .
3:   Let  $x_* = \arg \max_x \lambda_x^*$ . Break tie arbitrarily.
4:   Assign task  $x_*$  to the  $\ell^{th}$  worker.
5: end while

```

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 7 that this policy's performance is close to optimal.

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 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.

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 (2) generated by the Index Policy to the upper bound 11. We also compare the performance of the Index Policy to Optimistic Knowledge Gradient(OKG) method (Chen et al., 2011), 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 completion rate $\mu = 0.4$ with distributions specified in Section 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 (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 $\alpha = 1$ and $\beta = 1$. 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 1 we show a Semi-log plot of the number of tasks K against the average reward per task with the corresponding confidence intervals.

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

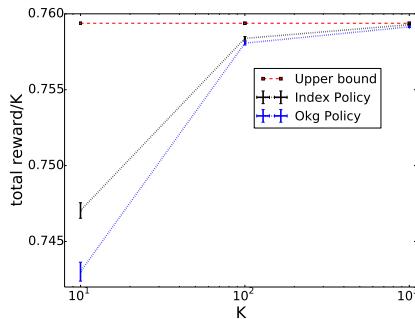


Figure 1: Semi-log plot of K against average per task reward (R/K) for $K = 10, 10^2, 10^3$.

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.

7.2 Simulation using real data

This set of simulations uses a real dataset, PASCAL RTE-1(Snow et al., 2008), 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 (Chapelle & Li, 2002) and a widely used frequentist approach - the upper confidence bound(UCB) policy (Auer et al., 2002). (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 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 2

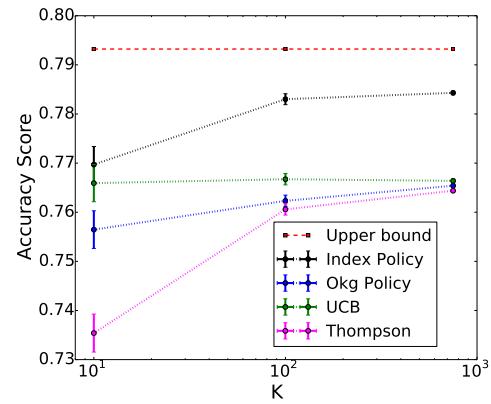


Figure 2: Semi-log plot of K against accuracy score for $K = 10, 100, 750$.

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 Chen et al. (2011) 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 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.

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.

9 Acknowledgment

The authors were partially supported by NSF CAREER CMMI-1254298, NSF IIS-1247696, NSF CMMI-1536895, AFOSR FA9550-12-1-0200, AFOSR FA9550-15-1-0038, and the ACSF AVF (Atkinson Center for a Sustainable Future Academic Venture Fund).

References

- Adelman, D and Mersereau, A. Relaxations of Weakly Coupled Stochastic Dynamic Programs. *Operations Research*, 2008.
- Auer, P, Cesa-Bianchi, N, and Fischer, P. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 2002.
- Bacharach, Y, Grapple, T, Minka, T, and Guiver, J. How to grade a test without knowing the answers - a Bayesian graphical model for adaptive crowdsourcing and aptitude testing. *ICML*, 2013.
- Chapelle, O and Li, L. An Empirical Evaluation of Thompson Sampling. *NIPS*, 2002.
- Chen, X, Lin, Q, and Zhou, D. Statistical Decision Making for Optimal Budget Allocation in Crowd Labeling. *arXiv*, 2011.
- Chen, Xi, Lin, Qihang, and Zhou, D Y. Optimistic Knowledge Gradient Policy for Optimal Budget Allocation in Crowdsourcing. *ICML*, 2013.
- Ghosh, A, Kale, S, and McAfee, P. Who moderates the moderators? Crowdsourcing Abuse detection in user-generated content. *ACM*, 2011.
- Gittins, J C. *Multi-Armed Bandit Allocation Indices*. John Wiley and Sons, New York, 1989.
- Hawkins, J. A Lagrangian decomposition approach to weakly coupled dynamic optimization problems and its applications. *Ph.D Thesis, Center of Operations Research, MIT*, 2002.
- Ho, CJ, Jabbari, S, and Vaughan, J W. Adaptive task assignment for crowdsourced classification. *ICML*, 2013.
- Karger, D, Oh, S, and Shah, D. Budget-Optimal Task Allocation for Reliable Crowdsourcing Systems. *Operations Research*, 2013a.
- Karger, D R, Oh, S, and Shah, D. Iterative Learning for Reliable Crowdsourcing System. *NIPS*, 2011.
- Karger, D R, Oh, S, and Shah, D. Efficient crowdsourcing for multi-class labeling. *ACM*, 2013b.
- Lovejoy, W S. A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research*, 28(1):47–65, 1991.
- Monahan, G E. A survey of partially observable Markov decision processes: Theory, models, and algorithms. *Management Science*, 28(1):1–16, 1982.
- Powell, W B. *Approximate Dynamic Programming: Solving the curses of dimensionality*. John Wiley and Sons, New York, 2007.
- Snow, R, OConnor, B, Jurafsky, D, and Ng, A. Cheap and Fast But is it Good? Evaluating Non-Expert Annotations for Natural Language Tasks. *EMNLP*, 2008.
- Thanh, T L, Venanzi, M, Rogers, A, and Jennings, N R. Efficient budget allocation with accuracy guarantees for crowdsourcing classification tasks. *ACM*, 2013.
- Whittle, P. Restless bandits: Activity Allocation in a Changing World. *Journal of Applied Probability*, 25: 287–298, 1988.
- Yan, Y, Rosales, R, Fung, G, and Dy, J. Active learning from crown. *ICML*, 2011.