Contents

1 Problem Statement and Motivation

With the unprecedented rise of crowdsourcing, various market places for generic micro-crowdsourcing have emerged such as Amazon Mechanical Turk (AMT), oDesk, eLance, LeadGenius, and Microsoft's and Google's internal crowds. The worker pools in different market places have different strengths. For example, AMT workers have much more variable skill and usually output lower quality at a lower price compared to oDesk workers that often charge higher but also produce higher quality work [8]. Then there are domain specific worker pools – ArcBazar is crowdsourcing specialised for architectural design, 99 Designs has expert artists, and TopCoder maintains a crowd of programmers and data scientists. In general specialised crowds charge significantly higher than generic crowds like AMT.

Within this crowdsourcing scenario, standard marketplace dynamics prevail. For instance, worker arrivals to the marketplace vary with the time of the day and the day of the year, roughly repeating every two weeks. After arriving onto the marketplace, a worker chooses to perform a specific task depending on the perceived attractiveness of that task, vis-a-vis other tasks on the marketplace. This notion of attractiveness may vary from worker to worker, but is assumed to be affected by certain measurable attributes, such as wage earned by the worker for solving that task, the time required for solving that task, etc. It is only natural that increasing (or decreasing) the price of some task on the marketplace would suitably change the rate at which workers flock to perform the task, which in turn directly affects the rate of completion of the task. This rudimentary intuition, therefore, gives us a tool to manipulate the completion rates of tasks on the crowdsourcing marketplace.

Another important aspect of crowdsourcing is the wide variability in workers that are employed on these platforms. This variability gives rise to the need for optimising the quality of answers for solved tasks. Often, the most basic quality assurance measure is aggregation: the same task is asked to be performed by multiple workers. The assumption being made is that by taking a 'majority vote' of the total responses collected, the confidence in the correctness of the answer can be greatly increased. However, many methods have been developed that compete with, and outperform, the elementary majority voting, such as [12], [2] and [7].

However, for most of these strategies, the relation between final answer quality and the price of the task is ubiquitously acknowledged. The quality of answers is expected to decrease on increasing the wage paid per task to each worker, as shown in [10], since getting a response from a worker is comparatively more expensive.

Therefore, the downside of increasing the per-task wages is that the overall quality of completed tasks decreases, even though the rate of completion of the task increases.

Finally, there may be additional constraints in the form of limitations on maximum expenditure that can be spent on a set of tasks, which curbs the flexibility of manipulation available. Optimising all three of these parameters: (1) the total time taken to complete a task; (2) the final quality of completed task, and (3) the total expenditure used for completing the task, is a challenging task that has not been solved by the larger research community. Furthermore, there is a need to develop a cogent framework that takes online decisions on this 3-way optimisation. Therefore, the goal of this thesis is to solve the following problem:

> *To solve the 3-way optimisation problem of Cost-Time-Quality in crowdsourcing by developing a cohesive framework, supported by theoretical justifications, that can operate on real world crowdsourcing markets.*

2 Literature Review

Our work aims to address the Cost-Quality-Time optimization problem. In this section, we discuss related work and then describe how we build on previous work.

Ho, Jabbari and Vaughn [5] deal with the problem of adaptive task routing for heterogeneous task difficulty, where the worker's accuracy on some task is a function of worker skill and question difficulty. They however assume a homogeneous cost across workers of different skill levels, and the skill level of the worker is learned against a set of gold-labels, as opposed to our algorithm's unsupervised approach (which leverages a modified version of the expectation maximisation algorithm [12]. They also classify tasks based on different types, so that each worker's skill level varies with a particular task type. Lastly, their model assumes that each worker will announce the number of tasks they are willing to perform on arrival; whereas in any real world online platform, worker retention is a complex problem to model, and assuming prior knowledge of it is inherently problematic. In Ho and Vaughn [6], similar assumptions on worker wages are assumed, along with having prior information about question difficulty levels (in one scenario) and worker training for specific task types (in the other scenario).

2.1 Cost-Quality Optimization

Prior work by [2] has addressed the question of how to minimize the amount of money spent on tasks, to optimize some requester defined reward function (assumed to be linear in the number of tasks). Their approach uses POMDPs to minimize the expected penalty incurred by the quality agent. Our own work, [10], extended this formulation to the problem of task routing in a crowdsourced setting. We show how to leverage the concept of worker 'qualifications? (often awarded on crowdsourcing platforms) to come up with lower cost solutions to tasks, while maintaining quality.

In addition, other work by [12] and [11], provides an explicit mechanism to manage quality issues for general tasks, and can be easily thought of as trading off cost with respect to some threshold of quality.

Our work aims to extend this by adding in the issue of time management while solving tasks, since time is often a concern for requesters who must meet strict deadlines.

2.2 Cost-Time Optimization

Previous work by [4] has looked at the problem of determining the optimal pricing mechanism to meet a requester specified deadline, as well as the problem of minimizing the latency of task solving, with a fixed budget. They take a dynamic-programming approach to the problem, backing up values in a Markov Chain to get the optimal policy in each state of the Markov Chain. We take cues from their work in our own formulation; however, we extend this formulation to a Markov Decision Process, and provide a compressed representation of the state space to make the policy computation both tractable, and computable offline.

The time-centered component of both their work and ours is derived from [3], who formulate a Thinned Non-Homogenous Poisson Process to model worker arrivals as well as task acceptance probability in a crowdsourced marketplace.

Importantly, no work has so far managed to provide a comprehensive framework to tradeoff these 3 variables. Our research aims to provide a decision-theoretic solution to this problem, and we describe our theoretical model in the next sections.

3 Problem Formulation

In a crowdsourcing setting, the rate at which HITs are satisfactorily completed is determined by the price at which the HITs are offered at the market. Furthermore, the quality of answers to HITs is determined by the price of HITs, since the price of asking for new ballots by workers can be prohibitive vis-a-vis the penalty incurred for submitting an incorrect answer. Therefore, the problem of 3-way optimization is to create a pipeline for pricing a batch of HITs on the market such that:

- The batch of HITs is completed either by a deadline, or, if the deadline has been missed, then in the least time possible.
- The accuracy of the HITs is maintained in-spite of increasing HIT prices.
- The total expenditure is minimized.

$$\begin{tikzcd}\boxed{\textsc{Pricing MDP}} \arrow[r] & \boxed{\textsc{Controller}} \arrow[r] & \boxed{\textsc{Quality MDP}}\end{tikzcd}$$

3.1 Terminology

The problem formulation has already been laid out above, and we formally describe the problem here.

- **Atomic Tasks:** The atomic unit that must be solved is termed a *task*. We are given N *boolean* tasks by the requester which have 0/1 answers. The set of all tasks is denoted by $\mathcal{T}$, single tasks by t_i where i is appropriately chosen, and any subset of tasks by T where $T \subseteq \mathcal{T}$.
- **HITs:** Tasks are posted in the marketplace in the form of *HITs*. We denote the set of all HITs (*batch* of HITs) by $\mathcal{H}$. Each individual HIT is denoted by $H(T,p)$ where T is the subset of tasks included in the HIT, and p is the *overall* price of the HIT. The per-task rate for any $t_i \in T$ is then $\frac{p}{|T|}$. The batch of HITs is $\mathcal{H} = \{H_1(T_1,p_1), H_2(T_2,p_2), \ldots, H_M(T_M,p_M)\}$ where M is the total number of HITs.
- **Time Indexing:** Variables that are time indexed are denoted by a superscript index, $.^{(\tau)}$. For instance, if we index HIT prices in time, the price at time τ for H_i would be $p_i^{(\tau)}$.

3.2 Assumptions

To model the problem as closely as possible to the real world scenario, the following assumptions have been made that ensure that our formulation stands valid even in the complex market dynamics of Mechanical Turk.

- **Batch Pricing:** The entire batch of HITs is priced equally at any time instant *i.e.* $\forall H_i, H_j \in \mathcal{H}$, $p_i^{(\tau)} = p_j^{(\tau)} = p^{(\tau)}$, where we denote the batch price as a whole by $p^{(\tau)}$.
- **Monotonic Pricing:** $p^{(\tau)}$ is non-decreasing *i.e.* the batch price for $\mathcal{H}$ will never decrease. This is to because workers on MTurk do not approve of frequent price changes for what is essentially the same HIT. Decreasing the price is often seen as bad requester policy on MTurk.
- **Discrete Time:** Time is discretized *i.e.* we only consider discrete instants of time $\{0, 1, \dots, \tau, \tau + 1, \dots\}$.
- **Single Worker Quality Distribution:** In terms of worker quality, all workers are assumed to be sampled from a single, identical distribution, denoted by $p(\Gamma)$. Note that this means that there are no separate worker pools.
- **Task Pickup Rate:** Worker arrival is modeled as a *Non-Homogenous Poisson Process (NHPP)*, and on arrival, a worker chooses a particular HIT to work on based on the *Discrete Choice Model.* Both of these models are described in the next section.
- **Batch Submission:** A *submission* (to the requester) is carried out for the entire batch of HITs *i.e.* for $\mathcal{H}$. Even if a HIT is deemed *completed*, we do not submit it to the requester, but instead wait to submit on completion of $\mathcal{H}$. We will also describe the case where we allow individual submissions, at a later stage.
- **Time-Based Reward:** We allow a requester defined time-based reward for submission of $\mathcal{H}$ denoted by $\Omega : \tau \rightarrow \Re$ where $\Omega(\tau)$ gives the time-reward for submitting $\mathcal{H}$ at time τ. Ω is constrained to be a piecewise linear function. As a simple case, we will consider Ω to be piecewise constant, so that if $\mathcal{H}$ is submitted before a deadline (τ_0), then a fixed reward ω is given, and this reward falls to 0 after τ_0.

$$\Omega(\tau) = \begin{cases} \omega & \tau \leq \tau_0 \\ 0 & \tau > \tau_0 \end{cases}$$

4 Quality MDP Formulation

This formulation of the Quality MDP as a Partially-Observable Markov Decision Process (POMDP), was developed in [1].

POMDPs provide a flexible framework with which to model decision-making when making noisy, or uncertain observations by relaxing the assumption that an agent has complete knowledge of his surroundings. The agent makes noisy observations about its current state, and the framework is used to model many single-agent, real-world problems, since the agent rarely has perfect information of its surroundings.

A POMDP is a six-tuple $< \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{P}, \mathcal{R} >$, where

- $\mathcal{S}$ is a finite set of discrete states.
- $\mathcal{A}$ is a finite set of all actions.
- $\mathcal{O}$ is a finite set of observations.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0,1]$ is the transition function describing the probability that taking an action in a given state will result in another state.
- $\mathcal{P} : \mathcal{S} \times \mathcal{O} \rightarrow [0,1]$ is the observation function describing the probability that taking an action in a given state will result in an observation.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$ is the reward for taking an action in a state.

POMDPs extend MDPs by adding an observation set $\mathcal{O}$ along with an observation model $\mathcal{P}$. Since the agent has no direct knowledge of the world's current state, it maintains a probability distribution over states, called a *belief state* $\mathbf{b}$, reflecting its estimate of their corresponding likelihood(s):

$$\mathbf{b}(\mathbf{s}) \in \ [0,1], \sum_{\mathbf{s} \in \mathcal{S}} \mathbf{b}(\mathbf{s}) = 1 \tag{1}$$

where $\mathbf{b}(\mathbf{s})$ is the probability that the current state is $\mathbf{s}$, inferred from the previous belief state, the most recent action, and the resulting observation.

Given this theoretical framework, Dai et al. [1] apply this model to the problem of taking responses (or ballots) from workers in a crowdsourced setting in a dynamic fashion. Specifically, they consider the task of solving binary classification problems (where responses can be either of 0/1). They model the worker's response using a generative model that is defined as:

$$a(d, \gamma) = \frac{1}{2}(1 + (1 - d)^{\gamma}) \tag{2}$$

where $d \in [0, 1]$ is the *intrinsic difficulty* of the task, and $\gamma \in [0, \infty]$ is the worker *error parameter*. A plate model representation of this generative model is given in Figure 1. Thus, a task with a difficulty approaching 1, leads to an accuracy near 1/2 which is like a random guess from the worker. Similarly, as a worker's error parameter approaches ∞, the accuracy of the worker's responses approach random guesses. The accuracy function defined above thus gives the probability that a worker will give the correct response, given the difficulty of the question and the worker's error parameter.

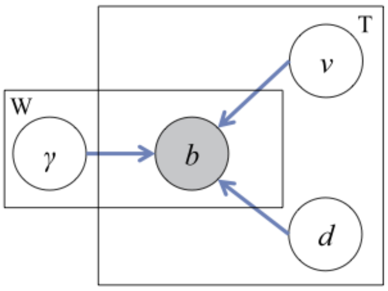

Figure 1: Plate Model Respresentation of the Generative Model. Reproduced from Dai, et al (2013).

Dai et al. then define their POMDP as follows,

- $\mathcal{S} = \{(d, v) | d \in [0, 1], v \in \{0, 1\}\}$ where d is the difficulty of the task and v is the true answer.
- $\mathcal{A} = \{$*query, submit true, submit false*$\}$
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$ contains the penalty for submitting an incorrect answer, and the cost of asking for a ballot.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1] = ((d, v), a, (d, v)) \mapsto 1$. All other probabilities are 0.
- $\mathcal{O} = \{true, false\}$ is a Boolean response from a worker.
- $\mathcal{P} : \mathcal{S} \times \mathcal{O} \to [0, 1]$ is defined by their generative model.

The POMDP uses an average γ value that it uses to model new workers i.e., every new worker is assumed to be an average worker with value equaling $\bar{\gamma}$. The POMDP manages belief states over the cross-product of the Boolean answer, and the task difficulty. Dai et al [1] discretize this difficulty into 11 possible values (to avoid a continuous state space), leading to a state space of $2 \times 11 = 22$ states.

They defined the POMDP's objective function as the minimisation of expected cost, where cost is the sum of money paid to the workers and a requester-defined penalty for the wrong answer. This naturally trades off cost and accuracy, with higher penalties leading to possibly higher payments and higher quality output. To solve POMDPs, they run the ZMDP package for $300s$ using the default Focused Real-Time Dynamic Programming search strategy. We continue to use these details in our model.

5 Controller Formulation

The controller is important for deciding which tasks within a HIT are to be prioritised to ensure that in a given discretised time segment, those tasks are completed which stand to gain maximum from new arriving ballots. As an example, if the agent operates in an environment which is very close to the deadline, then it is intuitively more important for all tasks to be at least partially complete, as opposed to the scenario wherein a few tasks are completed with full confidence, and the other are entirely incomplete. Therefore, a controller needs to have some notion of incremental utility while prioritising tasks.

To this end, we'd explored three strategies in our project, which have been outlined below.

5.1 Greedy 1-step look-ahead

The intuition behind this strategy is to predict which tasks will gain maximum from receiving one ballot in the immediately next time segment. This is implemented using the following strategy.

1. Assume that from the distribution of workers $p(\Gamma)$, an average worker arrives and performs each task. This worker will give one of two responses (i.e. *True* or *False*), which will then increase our belief over the correct answer for this task.
2. Calculate the expected increase in belief over the correct value of this task after receiving this new ballot for every task in the HIT.
3. Within the controller, sort the tasks in order of decreasing expected gain in utility, and push out the highest priority tasks from this order.

This strategy was formulated to ensure that a logical priority order over tasks is maintained. However, as we later understood, this formulation will lead to starvation of ballots for a certain number of tasks. For instance if a task gets two *True* responses early on, then the belief over the correct answer for this task will be *True* with some certainty. If then an average ballot is expected to arrive, then the change in utility for this task after getting this additional ballot will not be significant. Therefore, in the ordering that the controller maintains, this sort of a task will have low priority, and will be starved of responses that allow us to get a more accurate estimate of this task's correct answer.

To correct for this starvation and to ensure that during prioritising, a longer horizon is maintained, the following strategy was developed.

5.2 Psuedo look-ahead

This strategy was developed in [9], to avoid the starvation of votes scenario as described above. The intuition behind this strategy is to measure change in utility over a long term horizon by estimated the minimum number of ballots required to *reverse* the current belief over the correct answer of the task.

For instance, is the current belief in the correct value of the task is *True* with probability 0.8 and *False* with probability 0.2, then it may take say, 5 ballots that give the response of *False* each, to reverse our current belief estimate of the correct response of the task, so that the dominant belief is now *False*. In this case, since it took us 5 ballots to reverse our beliefs, therefore the utility of each ballot is $\frac{1}{5}$. Using this method, the utility of receiving a ballot, for each task, can be calculated by extrapolating and then normalising the number of ballots required to reverse dominant belief estimates.

With this intuition, the methodology for Controller operation is designed as follows:

- Calculate the expected number of ballots required to reverse the current belief over the correct answer for a task.
- Sort the tasks in order of gain in fractional utility for asking for the next ballot.

However, the problem with this strategy, and why it is not applicable to our controller design, is that the notion of *reversal* doesn't really capture the purpose of receiving ballots for tasks. Additional ballots are desirable for our task even in the scenario wherein ballots only strengthen our belief over the correct response, as opposed to reversing it, since for a question to be completed, there should be a high degree of confidence in the response value to avoid the negative penalty of submitting an incorrect answer. Therefore, the third strategy described below.

5.3 Increase in Fractional Utility

The intuition is the same, of using a longer term horizon for calculating gain in expected utility. However, we use information that we already maintain in the state space of the Cost-MDP to provide an efficient method of utility gain calculation.

The methodology is as follows:

- For each task, obtain the expected number of ballots to completion, $E[B]$, from the Cost-MDP.
- For each task, calculate the following ρ value:

$$\rho = \frac{U_{current+E[B]} - U_{current}}{E[B]}$$

- Sort each task within the controller in order of decreasing ρ value, and push out the task with the maximum ρ values at each time step.

Not only does this strategy efficiently avoid starvation of ballots, but it also provides a method to to utilise information already computed, and therefore is very well suited to our model.

6 Cost MDP Formulation

The Cost MDP is the over-arching MDP that sets the price of the batch of HITs, while optimising both time deadlines and total expenditure. To understand this MDP formulation, we use the NHPP (Non-homogenous Poisson Process) model developed in [3], as well as the model we developed to calculate the expected number of ballots.

6.1 Non-homogenous Poisson Process

The arrival of workers into the marketplace, as modelled in [3], is an NHPP, which is a generalisation of the Poisson Process with rate parameter $\lambda(\tau)$ (a function of time). The number of events (arrivals in the marketplace) occurring in an arbitrary time interval, $[\tau, \tau']$ is given by

$$\mathbf{N}[\tau, \tau'] \sim \mathbf{Pois}\left(\cdot | \lambda = \int_{\tau}^{\tau'} \lambda(x) dx \right)$$

where $\mathbf{Pois}\,(\cdot|\lambda)$ is a Poisson distribution with mean λ.

$\lambda(\tau)$ is considered to be a weekly periodic function, and the value of the function over time can be estimated using historical data.

In addition to the NHPP, the Discrete Choice Model examines the tendency of individual workers to pick our task over other tasks that are posted to the marketplace. We assume that each arrived worker has an independent probability p of picking up our task.

6.2 Expected Number of Ballots to Task Completion

Assume that we are currently at time instant τ and we are interested in calculating how many more ballots we require for a task t. Let B be a random variable that equals the number of ballots to completion for t, *e.g.* $B = 1$ implies that we require 1 more ballot to submit t. We are interested in $p(B)$, and more precisely in $\mathrm{E}[B]$, the expected number of ballots to task completion.

At τ, suppose that $p_t^{(\tau)}(d)$ denotes the (discrete) probability distribution over the difficulty of t. We will assume that this distribution is fixed for any future-looking calculations that we do, and consequently drop the superscript. We believe this is justified since this is the difficulty distribution to the best of our knowledge at τ. Also denote by $p_t^{(\tau)}(v)$, the belief over the solution space (boolean in this case) at τ for t.

6.2.1 Search Tree

Our strategy will be to simulate the POMDP's response to future ballots until we find a stage where the POMDP is ready to submit. Notice that every time a ballot is received by the POMDP, we can go down 1 of 2 branches, corresponding to either $v = 0$ or $v = 1$. For either of these responses, the POMDP would update both $p_t(v)$ and $p_t(d)$, and reach a new state. Consequently, we can construct a binary tree rooted at time τ as follows. Note that we do *not* update $p_t(d)$ for any forward-looking calculation.

- **Root:** The POMDP's current state (at τ) corresponds to the root node r of the tree.
- **Edges:** Each node has out-degree 2, with an edge annotation of either $v = 0$ or $v = 1$.
- **Nodes:** Any node u in the tree captures the POMDP's state ($p_t(v)$) on receiving ballots in identical order to the edges that form the path from r to u.

6.2.2 Frontier Finding Approach

Our problem is to find the shallowest frontier $\mathcal{F}$ of nodes (a set of nodes spanning the width of the tree), where for every node in $\mathcal{F}$, the POMDP would either SUBMIT TRUE or SUBMIT FALSE. Each of these nodes captures a state of the POMDP where t is completed, which are the states we are interested in looking for. Ensuring that the frontier is shallow (for any node u in $\mathcal{F}$, no ancestor of u can be a candidate for $\mathcal{F}$), ensures that we are picking the minimum number of ballots that it would take to submit for a frontier node.

Node Reachability Probability

For any node u, we would like to calculate the probability of generating that node. Assume that every simulated ballot is generated by an average worker with $\gamma = \mathrm{E}[p(\Gamma)]$. Since we have $p_t(d)$ rather than d, we can calculate the expected accuracy of an average worker. Therefore,

$$\begin{aligned} a(d,\gamma) &= \frac{1}{2}(1+(1-d)^{\gamma}) \\ \mathrm{E}[a] &= \frac{1}{2}\sum_{d} p_t(d)(1+(1-d)^{\gamma}) \\ \mathrm{E}[a] &= \frac{1}{2}(1+\sum_{d} p_t(d)(1-d)^{\gamma}) \end{aligned}$$

Notice that $\mathrm{E}[a]$ gives us the probability of generating the branch with the correct value. If we assume WLOG that $v = 1$ corresponds to the ground truth, then it gives us the probability of generating the $v = 1$ branch, while the probability of generating the other branch is then $1 - \mathrm{E}[a]$. Given a node u in the tree, assume that the path from r to u contains n_0 branches for $v = 0$ and n_1 branches for $v = 1$. The probability of reaching u becomes,

$$p(u) = (\mathrm{E}[a])^{n_1}(1-\mathrm{E}[a])^{n_0}$$

Also notice that the following equality holds, allowing us to define a valid probability distribution over nodes in $\mathcal{F}$,

$$\sum_{u\in\mathcal{F}} p(u) = 1$$

Expected Number of Ballots
Define $\delta(.)$ where $\delta(u)$ denotes the depth of u from the root ($\delta(r) = 0$). For any $u \in \mathcal{F}$, $\delta(u)$ is also the number of ballots after which the POMDP submitted an answer for the sequence of ballots leading to u. Then, we can find $\mathrm{E}[B]$ as simply,

$$\mathrm{E}[B] = \sum_{u\in\mathcal{F}} p(u)\delta(u)$$

We now have a way to compute $\mathrm{E}[B]$, if we can find the shallowest frontier in the search tree. To find this frontier, we can simply do DFS starting from r, and simulate the action taken by the POMDP at every node in the tree. In practice, heuristics on $|n_1 - n_0|$ values may help in reaching a solution quickly.

6.3 MDP Specification

The MDP to model the pricing decision problem is formulated as follows:

- State Space $\mathcal{S}$: $\{Expected\ number\ of\ ballots\ to\ completion, E[B]\}$
- Transition Function $\mathcal{T}$: $NHPP\ model$
- Reward Function $\mathcal{R}$: Ω $(discretised\ linear\ time - based\ reward)$
- Action Space $\mathcal{A}$: $+1, 0, submit$

However, one problem we discovered while working with this sort of a formulation was the insufficient information we had to make state-to-state transitions. As stated earlier, on changing the price per task, the Quality MDP would be less willing to accept new ballots, since the ballots are now more expensive. Therefore, our estimate of what the final expected ballots to completion remain after changing price was not accounted for.

For this, we plan to use Upper-Confidence Bounded Trees to learn parameters that provide us with information about how the Quality MDP's behaviour changes when the price per task is increased. Therefore, we plan to run simulations, and using simulations create a look-up table that provides us with appropriate discounting factors for changes in total expected ballots for respective price changes.

6.3.1 The Problem of Instant Gratification

Soon, we observed another issue with our MDP formulation: that of instant gratification. The problem of instant gratification is related to the lack of a quality based reward function in the Cost MDP, since the reward function is dependent only on the time-based reward. A consequence of this is the fact that the Cost MDP has no incentive to wait till more tasks are solved to an acceptable quality, and therefore the optimal decision for the Cost MDP is to submit the batch of HITs at the first possible instance.

To solve this issue, we need to maintain information about the quality levels of all tasks in the HIT within the MDP state space formulation. However, this would require that transitions between states also have information about what changes in each task's beliefs over the correct response changes as the price per batch is increased. Modelling this problem is extremely difficult since the next possible state after taking an action can be only of an extremely large number of states. Even if the batch of HITs contains 100 tasks, the belief of each task can change by a continuous value, and therefore the next possible state is impossible to map.

Possible Solution

One possible solution that was considered was a binning together of tasks in the HIT on the basis of equivalence classes generated over correct response beliefs. Since the belief states are maintained over continuous values, small differences in beliefs of two tasks should not require a different quality measure. While this reduces the total number of tasks, the reduction is not significant enough to make the state space formulation tractable, and the number of state spaces are still exponential.

Therefore, trying to factor in a *time and quality* based reward for the Cost MDP makes the transition function intractable. To solve this problem, we need a *compressed representation* of the belief values maintained for *every task*, so that the distribution of questions so that the degree of completion (and therefore the expected number of ballots required) can be reconstructed. This solution is explained in the subsequent section.

6.4 Compressed Representation of State Spaces

To begin with our proposal for a compressed representation, we first define the 'Degree of Completeness' mapping.

6.4.1 Degree of Completeness

The degree of completeness ($g_c(x)$) is a function that maps the current belief over the correct response for a task, to a notion of 'completeness'. The intuition behind this is the following: if some task τ_A has the following belief:

Correct Response	**Belief**
True	0.85
False	0.15

then the agent can be fairly certain about the true belief of τ_A, and it is more complete as compared to some other task, τ_B, which has the following belief state:

Correct Response	**Belief**
True	0.55
False	0.45

Degree of completeness is therefore a mapping from the task belief state to an integer between $[0, 1]$. The mapping, g_c, should be symmetric about the 0.5 value, since a strong belief in both a dominating *True* value and a dominating *False* value indicates a high degree of completeness. The function we currently use is a quadratic mapping that has the minima centred at 0.5 and symmetric maximas at 0 and 1.

6.4.2 Modified State Space

The state space can now be designated as a tuple of $(E[B], \overline{v})$, where $E[B]$ is the expected number of ballots required till completion, and $\overline{v}$ is the average over all tasks of the belief values. This formulation captures all complete information about all POMDP states be being able to reconstruct the distribution from which they were sampled.

We assume that the beliefs over questions follow a $\beta_{\alpha,\beta}$ distribution, with shape parameters α and β. By using the following equation:

$$E[B] = \int_0^1 g_c(x)\beta_{\alpha,\beta}(x)N dx,$$

we can obtain a closed form expression of the values of α and β, that allow us to reconstruct the distribution from which the beliefs were sampled, assuming that $g_c(x)$ is a quadratic function of v, given by the following expressions:

$$\alpha = \frac{(E - Na)\overline{v}}{Nb\overline{v}(1 - \overline{v}) - E + Na}$$
$$\beta = \frac{(E - Na)(1 - \overline{v})}{Nb\overline{v}(1 - \overline{v}) - E + Na}$$

where a and b are factors used to parameterise g_c, and E is shorthand for $E[B]$.

6.4.3 Modified Transition Function

As described earlier, using a more informative state space representation makes the transition function more complicated. During each transition, the value of $\overline{v}$ must be updated correctly. We already have a functioning model for updating $E[B]$. We therefore do not use a closed form transition function, and instead use table lookup while updating $\overline{v}$. This table is populated by running UCT (Upper Confidence Bounded Trees) on many Quality POMDP simultaneously, and learning the relevant parameters that inform us how $\overline{v}$ changes as more and more ballots are collected. This task can be completed offline, so that before learning the optimal policy for the Cost MDP, the transition function lookup table is available.

After changing the state space and transition function formulations, the reward function can be changed appropriately to a linear combination of the time based reward, as well as an aggregate penalty for submitting incorrect answers, such as:

$$\mathcal{R} = \Omega(t) + P(1 - \overline{v})$$

This allows us to learn a static optimal policy offline, so that the online execution of the algorithm is efficient.

7 Conclusion

During the course of the semester, we formulated a cohesive, multi-agent decision problem that provides a streamlined framework for interaction of many Markov Decision Processes operating in parallel, maximising a global, multi-dimensional objective. In this process, we created a novel compressed representation that allows us to represent the states of many different POMDPs in a condensed form. With this, we complete the theoretical formulation of the system that we're developing.

For future work, we plan on conducting extensive experiments in simulation that model all aspects of the online crowdsourcing marketplace, including worker arrival, worker task selection, time taken by workers to answer queries, as well as the implementation of the entire model. We also plan on conducting experiments on Amazon Mechanical Turk, the most popular crowdsourcing platform, to compare the performance of our system with existing models.

References

[1] Peng Dai, Christopher H Lin, Daniel S Weld, et al. Pomdp-based control of workflows for crowdsourcing. *Artificial Intelligence*, 202:52–85, 2013.

[2] Peng Dai, Mausam, and Dan Weld. Decision-theoretic control of crowd-sourced workflows. In *In the 24th AAAI Conference on Artificial Intelligence (AAAI?10*. Citeseer, 2010.

[3] Siamak Faradani, Björn Hartmann, and Panagiotis G Ipeirotis. What's the right price? pricing tasks for finishing on time. *Human computation*, 11, 2011.

[4] Yihan Gao and Aditya Parameswaran. Finish them!: Pricing algorithms for human computation. *Proceedings of the VLDB Endowment*, 7(14):1965–1976, 2014.

[5] Chien-Ju Ho, Shahin Jabbari, and Jennifer W Vaughan. Adaptive task assignment for crowdsourced classification. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 534–542, 2013.

[6] Chien-Ju Ho and Jennifer Wortman Vaughan. Online task assignment in crowdsourcing markets. In *AAAI*, volume 12, pages 45–51, 2012.

[7] Panagiotis G Ipeirotis, Foster Provost, and Jing Wang. Quality management on amazon mechanical turk. In *Proceedings of the ACM SIGKDD workshop on human computation*, pages 64–67. ACM, 2010.

[8] Panos Ipeirotis. A computer scientist in a business school. Mechanical Turk vs oDesk: My experiences, 2012.

[9] Christopher H Lin, Mausam, and Daniel S Weld. Reactive learning: Active learning with relabeling. In *AAAI*, 2015.

[10] Shreya Rajpal, Karan Goel, and Mausam. Pomdp-based worker pool selection for crowdsourcing.

[11] Peter Welinder, Steve Branson, Pietro Perona, and Serge J Belongie. The multidimensional wisdom of crowds. In *Advances in neural information processing systems*, pages 2424–2432, 2010.

[12] Jacob Whitehill, Ting-fan Wu, Jacob Bergsma, Javier R Movellan, and Paul L Ruvolo. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *Advances in neural information processing systems*, pages 2035–2043, 2009.