cause LazySusan should expect a small $k$ if $\theta$ is small. If $\theta = \theta_2$, LazySusan should see a smaller $k$ than if $\theta = \theta_1$. If it sees a $k$ that is larger than it expects, it correctly deduces that more people are getting the question wrong, and concludes the task is more difficult. Similarly, if $k$ is close to 1, and $\theta = \theta_1$, then LazySusan believes with more certainty that the task is easy than if $\theta = \theta_2$, since even though workers tend to produce more random answers, $k$ is small.

4.2 Utility Estimation

To determine what actions to take, LazySusan needs to estimate the utility of each action. The first step is to assign utilities to its beliefs. Since $B_i$ solely determines its belief at time $i$, we denote $U(B_i)$ to be utility of its current belief. Next, LazySusan computes the utilities of its two possible actions. Let $Q(B_i, \text{submit})$ denote the utility of submitting the most likely answer given its current state, and let $Q(B_i, \text{request})$ denote the utility of requesting another worker to complete the task and then performing optimally. Then

$$U(B_i) = \max\{Q(B_i, \text{submit}), Q(B_i, \text{request})\}$$

$$Q(B_i, \text{submit}) = \sum_{a \in A_i} V(a) \int_d P(v = a, d | B_i; i, k) dd$$

$$Q(B_i, \text{request}) = c + \sum_{a \in A_i} P(b_{i+1} = a | B_i) U(B_{i+1})$$

$$+ P(b_{i+1} \notin A_i | B_i) U(B_{i+1})$$

where $c$ is the cost of creating another job and $P(b_{i+1} | B_i) = \sum_{a \in A_i} \int_d P(b_{i+1} | v = a, d, B_i) P(v = a, d | B_i) dd$

LazySusan takes as inputs $C_C$, the utility of a correct answer, and $C_W$, the utility of an incorrect answer. These values are provided by the requester to manage tradeoffs between accuracy and cost. LazySusan uses its own estimate of the correct answer to calculate $V(a)$:

$$a^* = \arg \max_{a \in A_i} \int_d P(v = a, d | B_i; i, k) dd$$

$$V(a) = \begin{cases} C_C & \text{if } a = a^* \\ C_W & \text{otherwise} \end{cases}$$

4.3 Worker Tracking

After submitting an answer, LazySusan updates its records about all the workers who participated in the task using $a^*$. We follow the approach of Dai et al. [Dai et al., 2010], and use the following update rules:

1) $\gamma_w \leftarrow \gamma_w - de$ should the worker answer correctly, and
2) $\gamma_w \leftarrow \gamma_w + (1 - d)\epsilon$, should the worker answer incorrectly, where $\epsilon$ is a decreasing learning rate. Any worker that LazySusan has not seen previously begins with some starting $\gamma$.

4.4 Decision Making

We note that the agent’s state space continues to grow without bound as new answers arrive from crowdsourced workers. This poses a challenge since existing POMDP algorithms do not handle infinite-horizon problems in dynamic state spaces where there is no a-priori bound on the number of states. Indeed, the efficient solution of such problems is an exciting problem for future research. As a first step, LazySusan selects its actions at each time step by computing an $l$-step lookahead by estimating the utility of each possible sequence of $l$ actions. If the $l$th action is to request another response, then it will cut off the computation by assuming that it submits an answer on the $l + 1$th action.

In many crowdsourcing platforms, such as Mechanical Turk, we cannot preselect the workers to answer a job. Therefore, in order to conduct a lookahead search, we need to specify future workers’ parameters for our generative model. To simplify the computation, we assume that every future worker has $\gamma = \bar{\gamma}$.

4.5 Joint Learning and Inference

We now describe an EM algorithm that can be used as an alternative to the working-tracking scheme from above. Given a set of worker responses from a set of tasks, $b$, EM jointly learns maximum-likelihood estimates of $\gamma$, $d$, and $\theta$, while inferring the correct answers $v$. Thus, in this approach, after LazySusan submits an answer to a task, it can recompute all model parameters before continuing with the next task.

We treat the variables $d, \gamma,$ and $\theta$ as parameters. In the E-step, we keep parameters fixed to compute the posterior probabilities of the hidden true answers: $p(v_t | b, d, \gamma, \theta)$ for each task $t$. The M-step uses these probabilities to maximize the standard expected complete log-likelihood $L$ over $d, \gamma,$ and $\theta$:

$$L(d, \gamma, \theta) = E[\ln p(v, b | d, \gamma, \theta)]$$

where the expectation is taken over $v$ given the old values of $\gamma, d, \theta$.

5 Experiments

This section addresses the following three questions: 1) How deeply should the lookahead search traverse? 2)