# Octopus: A Framework for Cost-Quality-Time Optimization in Crowdsourcing

**Karan Goel***
Carnegie Mellon University
kgoel93@gmail.com

**Shreya Rajpal***
Univ. of Illinois at Urbana-Champaign
shreya.rajpal@gmail.com

**Mausam**
Indian Institute of Technology - Delhi
mausam@cse.iitd.ac.in

## Abstract

We present OCTOPUS, an AI agent to jointly balance three conflicting task objectives on a micro-crowdsourcing marketplace – the quality of work, total cost incurred, and time to completion. Previous control agents have mostly focused on cost-quality, or cost-time tradeoffs, but not on directly controlling all three in concert. A naive formulation of three-objective optimization is intractable; OCTOPUS takes a hierarchical POMDP approach, with three different components responsible for setting the pay per task, selecting the next task, and controlling task-level quality. We demonstrate that OCTOPUS significantly outperforms existing state-of-the-art approaches on real experiments. We also deploy OCTOPUS on Amazon Mechanical Turk, showing its ability to manage tasks in a real-world, dynamic setting.

## Introduction

Task control of workflows over micro-task crowdsourcing platforms, such as Amazon Mechanical Turk (AMT), has received significant attention in AI literature (Weld et al. 2015). Typically, a requester needs to balance three competing objectives – (1) total *cost*, owing to payments made to workers for their responses (or *ballots*), (2) overall *quality*, usually evaluated as accuracy of the final output, and (3) the total *time* for completing the task. These criteria are interrelated: increasing the pay per task attracts more workers to the task, thereby reducing completion time. However, it also exhausts the budget sooner, so requesters can afford fewer ballots per task, likely reducing the overall quality.

Most prior work on crowd controllers has focused on the tradeoff between cost (or no. of ballots) and quality (Dai et al. 2013; Lin, Mausam, and Weld 2012; Bragg, Mausam, and Weld 2013; Kamar et al. 2013; Parameswaran et al. 2012). A common approach is to define a Partially Observable Markov Decision Process (POMDP) *per task*, which decides on whether to get another ballot or submit the best answer for that task. However, this work is time-agnostic, and assumes that pay per ballot is given as input.

Recent work has also studied the tradeoff between cost and completion time for a *batch* of tasks (Gao and Parameswaran 2014). They model the problem as a Markov Decision Process (MDP) that changes the pay per ballot, so that all ballots can be obtained by the given deadline in a cost-efficient manner. However, this work assumes that the number of ballots needed to complete the whole batch is a constant known to the requester in advance.

There is limited research on simultaneously addressing tradeoffs between cost, quality and latency. We know of only one work that studies this for the specific workflow for finding *max* of a set of items (Venetis et al. 2012). This work assumes that latency is pay-independent – an assumption well-known to be incorrect (Faradani, Hartmann, and Ipeirotis 2011; Gao and Parameswaran 2014). Under a fixed latency-per-response assumption, they speed up the task and save cost by taking fewer responses. Our three-way optimization is for the broader case of answering a batch of tasks, and uses variable pricing to alter the latency of task completion, in line with crowdsourced marketplace dynamics.

Building upon these strands of research, we present OCTOPUS, an AI agent that can balance all three objectives (cost, quality, time) in concert on real crowdsourced marketplaces, by optimizing a requester-specified, *joint* utility function for a batch of tasks. It achieves this by controlling both the pay per ballot and the (predicted) accuracy of each individual task.

We could model the whole problem as a *single* POMDP, however, that is unlikely to scale. An alternative could be to use multi-objective MDPs, but they are also less tractable, because they produce a pareto-optimal set of solution policies (Chatterjee, Majumdar, and Henzinger 2006). OCTOPUS uses a three-component architecture – one to set the pay per ballot (COSTSETTER), another to choose the next available task (TASKSELECTOR) and a third to control each task's quality (QUALITYMANAGER). A key technical novelty is in the careful modeling of the COSTSETTER's state space in order to circumvent intractability – the state space contains aggregate statistics regarding completion levels of all tasks, so that it can decide the next best pay to set.

We perform extensive experiments using both simulated and real data, as well as online experiments on AMT. Since no existing system performs direct 3-way optimization in crowdsourced marketplaces, our experiments compare against existing state-of-the art approaches that optimize 2 of the 3 objectives. We find that in most settings,

*Most work was carried out when the authors were students at the Indian Institute of Technology - Delhi.

OCTOPUS *simultaneously* outperforms, or is at par with *multiple* variants of these baselines. Our contributions are:

1. We describe OCTOPUS, a novel framework to address cost-quality-time optimization for a batch of tasks in a crowdsourced marketplace setting. It contains three components that set the pay per ballot, select the next task and control each task's quality. A key technical novelty is the use of aggregate statistics of all tasks in the state space design for the COSTSETTER, ensuring tractability for real-time deployment.
2. OCTOPUS consistently performs at par with or better than state-of-the-art baselines, yielding up to 37% reward improvements on real data.
3. We deploy OCTOPUS on AMT and demonstrate that it is able to optimize utility effectively in a live, online experiment.[1]

## Related Work

**Cost-Quality Optimization.** There is significant work on getting more quality out of a fixed budget. One branch of this research focuses on *collective classification*, which develops aggregation mechanisms to infer the best output per task, given a static set of ballots (Whitehill et al. 2009; Welinder et al. 2010; Oleson et al. 2011; Welinder and Perona 2010). The other branch studies *intelligent control*, which dynamically decides whether to ask for a new ballot on a task, or stop and submit the answer. These include control of binary or multiple choice tasks (Dai et al. 2013; Parameswaran et al. 2012; Kamar et al. 2013), multi-label tasks (Bragg, Mausam, and Weld 2013), and tasks beyond multiple choice answers (Lin, Mausam, and Weld 2012; Dai, Mausam, and Weld 2011). All these works design agents to control a *single* task and assume a constant pay per ballot. Our work closely follows the POMDP formulation laid down in Dai *et al.* (2013) for binary tasks.

**Cost-Time Optimization.** Increasing pay per ballot can reduce completion times. Faradani *et al.* (2011) develop models to find upfront, the static price per ballot so that a desired deadline can be met. Gao & Parameswaran (2014) extend this by varying pay at discrete time-steps using an MDP. Both approaches assume a fixed number of ballots known a-priori, without dynamic quality control of tasks. There is also some work on price-independent latency reduction (Haas et al. 2015).

**Cost-Quality-Time Optimization.** There is limited work in this area. The only paper we are aware of is Venetis *et al.* (2012), which addresses cost-quality-time optimization but in a restrictive setting with important distinctions from our work: (i) they look at max-finding for a set of items, while our task-type is classification; (ii) they consider latency to be pay-independent and fixed per task, while we study the more realistic setting in which changing pay directly impacts workers' desire to work on our tasks; (iii) unlike us, they *don't* change pay per task directly, and instead, change the number of responses sought per task to control both cost and latency. Qualitatively, our work thus also highlights how workers perceive pay changes in a crowdsourcing marketplace and its overall effect on task completion.

**Worker Retention.** Previous work deals with incentivizing workers to perform more tasks via bonuses or diversification (Rzeszotarski et al. 2013; Ipeirotis and Gabrilovich 2014; Difallah et al. 2014; Dai et al. 2015). Recently, Kobren *et al.* (2015) model the process of worker retention. We present empirical results that suggest worker retention plays a dominant role in determining task completion rates.

**Task Routing.** Prior work deals with two issues; deciding which task from the batch to solve next, or which worker to route a task to. Ambati *et al.* (2011) rank tasks based on user preferences using a max-entropy classifier. Other work uses low-rank matrix approximations (Karger, Oh, and Shah 2014) for equal difficulty tasks. Rajpal *et al.* (2015) decide which worker pool to route a task to. Other papers study task routing on volunteer platforms (Bragg et al. 2014; Shahaf and Horvitz 2010).

Like AMT, we assume no control on which worker picks a ballot job, but we select the best next task to assign to an incoming worker. Following (Mason and Watts 2010; Gao and Parameswaran 2014), we assume that worker quality is independent of the pay per ballot. We re-verify this for our data in our experiments.

**Decentralized Approaches.** There is related work in decentralized Wald stopping problems (Teneketzis and Ho 1987) which considers how to optimize a common utility function given a set of agents who each make independent observations. However, these approaches do not scale well with the number of agents (tasks in our setting), which can be quite large. There is also work in decentralized metareasoning (Hansen and Zilberstein 2001) to decide when to stop optimizing a utility function. Metareasoning approaches typically assume that utility is monotonically increasing over time, which is not true in our setting since for instance, conflicting ballots on a single task would decrease utility.

## Problem Definition

A requester provides a batch of $n$ binary tasks $q \in 1 \dots n$, each having a $0/1$ response. They also provide a utility function $\mathcal{U}$, which describes how to tradeoff cost, time and quality. The agent can dynamically change pay per ballot $c$, and choose a variable number of ballots per task to optimize the final objective. We study the setting where $\mathcal{U}$ is expressed as a sum of task-level utilities ($U$) minus cost, *i.e.* $\mathcal{U} = \sum_{q=1}^{n}(U_q - C_q)$. Here, $C_q$ is total money spent on $q$. We assume that answers to all tasks are to be returned to the requester as *one* single batch.

**MDP/POMDP background.** An MDP models the long-term reward optimization problem under full observability and is defined by a five tuple $\langle S, A, T, R, \gamma \rangle$. Here, $S$ is a set of states, $A$ a set of actions, $T(s'|s, a)$ denotes the probability of transitioning to state $s'$ after taking action $a$ in state $s$, and $R(s, a)$ maps a state-action pair to a real-valued reward. $\gamma$ is the discount factor for making infinite-horizon MDPs well formed. A POMDP extends an MDP into a partially-observable setting, where the state is not fully

[1] Code can be found at `https://github.com/krandiash/octopus`.

observable and only a belief (probability distribution) over possible states can be maintained using observations from the model. A POMDP is represented as $\langle S, A, T, R, O, \gamma \rangle$ tuple, where a new function $O(o|s', a)$ denotes the distribution over observations on taking an action $a$ and arriving in a new state $s'$. Lack of space precludes a long discussion of the subject – there are existing solvers for solving MDPs and POMDPs of reasonable sizes, e.g. (Smith and Simmons 2012), which we use in our work.

To formulate the problem optimally, we would need to define a single, centralized POMDP over a state containing answers and difficulty estimates of all individual tasks as well as the current pay per ballot and the current time. The actions will include requesting a ballot on a task $q$, changing the pay, and a terminal submit action. Solving this POMDP would yield the optimal policy, which would decide which task to get ballots on next, when to change pay and when to submit. Since the number of tasks in a batch can be huge, this POMDP is unlikely to scale due to a large state and action space. Naive extensions to Dai *et al.*'s or Gao *et al.*'s state-of-the-art models for cost-quality and cost-time optimization respectively are not possible either – Dai *et al.*'s model is solved per task, whereas pay must be set based on progress of the whole batch of tasks; Gao *et al.*'s model assumes a fixed number of ballots per task, and has no natural way to optimize quality by taking a variable number of ballots based on each task's difficulty.

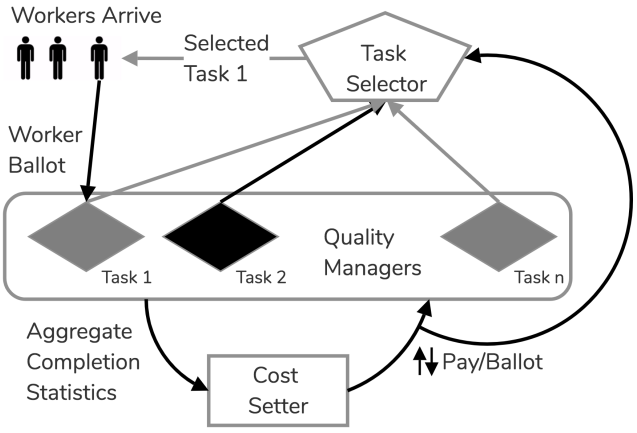



Figure 1: OCTOPUS architecture.

## OCTOPUS for Three-Way Optimization

We propose a three-component architecture (see Figure 1). In OCTOPUS each task has its own QUALITYMANAGER that decides, based on the current pay, whether it is worth taking another ballot for this task (light edge) or not (dark edge). This information is conveyed to the TASKSELECTOR, which selects an available (light) task to route to an incoming worker. Based on the current progress of the whole batch, the COSTSETTER decides what pay per ballot to set; this action is taken at periodic intervals. We design OCTOPUS such that it can allocate tasks as and when workers arrive and works instantaneously in practice, therefore utilizing crowdsourcing marketplaces with full parallelism.

### QualityManager

**Background.** QUALITYMANAGERS follow the worker response model and POMDP formulation of Dai *et al.* (2013). Each worker is assumed to have an error parameter $\gamma \in (0, \infty)$ ($\gamma = 0$ is error-free). An average worker has error parameter $\bar{\gamma}$. Each task $q$ has an unknown true Boolean answer $t_q$, and an associated difficulty $d_q \in [0, 1]$ ($d_q = 0$ is easy) – these are estimated using an Expectation-Maximization algorithm (Whitehill et al. 2009) as data is received. Each task has a prior difficulty distribution $p(d_q)$. A worker's ballot for $q$ depends on their $\gamma$, $t_q$ and $d_q$.

In Dai *et al.* (2013) the POMDP for $q$ maintains a belief state $\mathbf{b}_q$ over $(d_q, t_q)$ state tuples. For instance $\mathbf{b}_q(0.5, 1) = 0.4$ indicates a 40% belief that $d_q = 0.5$ and the answer to $q$ is 1. The POMDP has two actions; (1) request another ballot, or (2) mark $q$ as completed. The POMDP policy $\pi_q$ maps every belief state to an action. Each time the POMDP receives another ballot, a Bayesian update is performed to re-estimate a new belief. For OCTOPUS, the POMDP optimizes $U_q - C_q$. We note that $\pi_q$ depends on the pay per ballot $c$: if $c$ is smaller, the POMDP can afford more ballots. Dai *et al.*'s original model keeps $c$ constant, but in our case the COSTSETTER can change it, triggering a change in $\pi_q$.

We define $v_q$ as the confidence in the most probable answer for task $q$; $v_q = \max(v_q^0, v_q^1) \in [0.5, 1]$, where $v_q^0, v_q^1$ are the current probabilities that $q$'s answer is 0 or 1 respectively. $v_q$ can be computed from $\mathbf{b}_q$ by summing out $d_q$.

**Computation of Aggregate Statistics.** We now define two batch-level statistics, which will be a key part of the state space representation for the COSTSETTER.

We first define a normalized estimate of *task quality*, $\nu_q = 2v_q - 1$; $\nu_q$ normalizes $v_q$ so that it lies in $[0, 1]$. A high value of $\nu_q$ (near 1) indicates the POMDP's high confidence in its estimated answer for $q$, and vice versa for a low (near 0) value. We also define a related notion of *batch quality*, $\bar{\nu} = \frac{1}{n} \sum_{q=1}^{n} \nu_q$, which is an aggregate statistic estimating the current quality for the entire batch of $n$ tasks. Finally, we construct the *batch quality histogram* – a histogram built by binning tasks into equally sized bins based on their $\nu_q$ values. The bin width is denoted as $\Delta_\nu$.

The COSTSETTER also needs an estimate of the number of ballots remaining. We define $\theta_q(\pi_q)$ as an estimate of the expected number of ballots that will be needed (starting from the current time) until task $q$ will be marked completed. For notational ease we write $\theta_q$ to denote $\theta_q(\pi_q)$. Recall that $\pi_q$ can change with a change in $c$. Hence $\theta_q$ also depends on $c$.

How can we compute $\theta_q$? We use a trajectory-tree approach (similar to (Kearns, Mansour, and Ng 1999)) called FRONTIERFINDING. We construct a binary tree of future observations, rooted at the current time step, where each edge corresponds to an observation (a worker response of $0/1$). The node below any edge contains the belief state generated by updating the POMDP, using the observation associated with that edge. Each trajectory is a path from the root to a leaf, and is generated with an associated path probability (using Dai *et al.*'s generative model assuming an average worker). A leaf is created whenever the policy takes

the 'mark as completed' action, or when the path probability drops below a threshold. $\theta_q$ is simply the expected length of a trajectory in this tree.

We also estimate the expected ballots to completion for the batch: $\theta = \sum_{q=1}^{n} \theta_q$. $\theta$'s role is similar to that of $\bar{\nu}$ – it is an aggregate statistic that describes how far the batch is from completion. It also helps in quantifying the expected cost of completion: if $\theta = 1000$ and $c = 3$, we would expect to spend $c \cdot \theta = 3000$ units of money to complete the batch.

In summary, we described the design of a per-task QUALITYMANAGER. Collectively, $n$ of these help us in estimating two aggregated quantities, $\bar{\nu}$ and $\theta$, which measure the overall quality, and degree of completion of the batch, respectively. All notation for this and future sections is summarized in Table 1.

## TaskSelector

The TASKSELECTOR decides which incomplete task to assign to the next incoming worker. It must have an 'anytime' behavior, *i.e.* it must increase utility $\mathcal{U}$ quickly. This is because the time of final submission is not in its control, and the batch might be submitted at any time by the COSTSETTER.

To be prepared for any contingency, the TASKSELECTOR uses a 1-step greedy policy over expected utility gain. We define each task's priority ($\phi_q$) as the difference between the current utility ($U_q$) of $q$ and the expected utility after receiving 1 ballot ($U'_q$) from an average worker (error rate $\bar{\gamma}$), given the current belief state $\mathbf{b}_q$ of $q$'s QUALITYMANAGER. Thus, $\phi_q = \mathbf{E}[U'_q|\bar{\gamma}, \mathbf{b}_q] - U_q$. TASKSELECTOR assigns the task with the maximum $\phi_q$ value to the next available worker.

Unfortunately, $U_q$ (and therefore $\mathcal{U}$) is neither monotonic (conflicting ballots decrease utility) nor submodular (a skilled worker could arrive after an error-prone one), so we cannot utilize prior work on adaptive submodularity (Golovin and Krause 2011; Bragg et al. 2014) to guarantee solution quality. Providing quality bounds is left for future work.

Lastly, note that the task allocation process is instantaneous, as well as completely parallelized, since we don't wait for a task to be returned before allocating another task. Given enough workers, we could get ballots on every single task in parallel. This is important, since it allows us to take full advantage of micro-task marketplaces.

## CostSetter

The COSTSETTER is an MDP that changes $c$ (pay per ballot) in order to maximize $\mathcal{U}$. It uses information about the completion level of each task to assess whether the batch of tasks is completing on schedule or needs to be sped up or slowed down. To influence the rate of completion of the batch, it sets $c$ at discrete time steps $\tau \in \{0, \Delta_\tau, 2\Delta_\tau, \dots\}$.

The key challenge for the COSTSETTER is in defining the state space. Ideally, as stated earlier, each task's belief $\mathbf{b}_q$ should be part of the state, but that would make computations intractable. Instead, we approximate by using aggregate statistics over the whole batch of tasks. We describe the state space, actions, transition functions, and rewards of this MDP below.

| Symbol | Meaning |
|---|---|
| $\mathcal{U}$ | Overall utility (requester defined function) |
| $U_q$ | Per-task utility |
| $c$ | Pay per ballot |
| $C$ | Total cost incurred for all tasks |
| $C_q$ | Total cost incurred for task $q$ |
| $\pi_q$ | Policy for task $q$'s QUALITYMANAGER |
| $\gamma$, $\bar{\gamma}$ | Worker error parameter, average error |
| $d_q$ | Difficulty of task $q$ |
| $t_q$ | True binary answer for task $q$ |
| $\mathbf{b_q}$ | Belief state for task $q$ |
| $v_q$ | Probability of POMDP's most likely true answer for task $q$ |
| $\nu_q$ | Task quality ($\nu_q = 2v_q - 1$) |
| $\bar{\nu}$ | Batch quality ($\bar{\nu} = \frac{1}{n}\sum_q \nu_q$) |
| $\theta_q$ | Estimated number of ballots required to complete task $q$ |
| $\theta$ | Estimated number of ballots required for completing the batch of tasks ($\theta = \sum_q \theta_q$) |
| $\tilde{\theta}(\nu_q, c)$ | Mapping from $(\nu_q, c)$ to $\theta_q$ for task $q$ under $\pi$ |
| $\phi_q$ | Priority of task $q$ |
| $\tau_{max}$ | Deadline |
| $\tau$ | Current Time |
| $\Delta$ | Granularity (*e.g.* $\Delta_\tau$) |

Table 1: Notation used in the paper.

**State Space.** The choice of the best pay per ballot $c$ depends on its current value, the current time, and the aggregate degree of completion of the batch. We choose the state to be a 4-tuple $(\bar{\nu}, \theta, \tau, c)$. Both $\bar{\nu}$ and $\theta$ are important for this decision; $\bar{\nu}$ estimates the expected accuracy on the batch, while $\theta$ gives us an idea of how much more improvement in $\bar{\nu}$ is possible (at the current $c$). For a fixed $\bar{\nu}$ and $c$, a high $\theta$ would indicate the presence of several unsolved tasks and the possibility of improving $\bar{\nu}$. On the other hand, a low $\theta$ would indicate that most tasks are solved and there is little improvement possible. $\theta$ therefore captures the *spread* of the distribution of task qualities $\nu_q$, while $\bar{\nu}$ is the mean of this distribution. We use this intuition later to construct the transition function for the COSTSETTER.

As another example consider a case where both $\theta$ and $\bar{\nu}$ are high. This indicates that the QUALITYMANAGERs consider there be to scope for quality improvement despite the batch quality being high already, possibly due to very low $c$. If we did not have $\theta$ in the state, we would instead base our decision on the high $\bar{\nu}$ value, and believe that further improvement in utility was not possible.

All state variables are continuous, and for tractability we discretize them. $\theta$ is discretized with a granularity $\Delta_\theta$, $\bar{\nu}$ with a granularity $\Delta_\nu$, and $\tau$ with a granularity $\Delta_\tau$. We assume that $c$ can take values $\{c_1, c_2 \dots, c_k\}$. These values are defined by the requester, and in practice would respect marketplace constraints, such as minimum wages. Interestingly, the model is robust in that if a requester provides a very poor starting wage, workers will likely not pick up the task, and the model will subsequently respond by increasing the wage.

**Actions and Rewards.** Every state has access to two pay-change actions: $\uparrow$ and $\downarrow$. $\uparrow$ increases $c_i$ to $c_{i+1}$ while $\downarrow$ does the opposite. The $\uparrow$ and $\downarrow$ actions incur no cost to the system. However, in practice we assign a small cost to these actions to prevent frequent cyclical pay-changes in a policy. Changing $c$ has no impact on $\tau$ and $\bar{\nu}$, but it does change $\theta$, since the number of ballots remaining per task depends on the current pay per ballot (via the policy $\pi_q$). We discuss how to compute this when defining the transition function.

We also have a no-change action, which increments $\tau$ by $\Delta_\tau$, along with asking workers for more ballots at $c$, which remains unchanged (for the next $\Delta_\tau$ duration). This is essentially a marketplace action, where we post tasks to the marketplace with a pay per ballot equaling $c$. At the end of $\Delta_\tau$ minutes, we would then arrive in a new state. The cost of this transition (to the nearest $\Delta_\theta$ value) is just the amount paid to workers during this duration on the marketplace, equaling the number of ballots received during this time, multiplied by $c$.

The final action is a 'terminate' action that submits all answers to the requester. Its reward should be $\mathcal{U}$ based on batch quality and current time (cost is not needed, since that was already accounted in the no-change action). Unfortunately, the MDP has access to only the aggregate statistics, and not the full batch quality histogram, which is needed for computing $\mathcal{U}$. We now describe a novel $\beta$-reconstruction procedure that allows us to extrapolate the full histogram from aggregate statistics, useful for computing this reward as well as the transition function.

**$\beta$-Reconstruction.** The goal is to reconstruct an approximate batch quality histogram given the aggregate statistics, $\bar{\nu}$ and $\theta$, and the current $c$. The procedure assumes that the histogram can be approximated with a two parameter Beta distribution, $\beta_{\lambda_1,\lambda_2}$. Also assume that we are provided a function $\tilde{\theta}$ that maps a $(\nu_q, c)$ pair to $\theta_q$; it returns the expected number of ballots needed for a task $q$ given its current quality and pay per ballot. Note also that $\tilde{\theta}$ will be a non-increasing function of $\nu$. We now show that we can find suitable $\lambda_1$ and $\lambda_2$ given $\bar{\nu}$, $\theta$ and $\tilde{\theta}$.

To compute the best fit $\lambda_1, \lambda_2$ values, we solve two equations. The $1^{\text{st}}$ equation enforces that the mean of the reconstructed distribution is $\bar{\nu}$: $\frac{\lambda_1}{\lambda_1+\lambda_2} = \bar{\nu}$. We can reparameterize $\beta_{\lambda_1,\lambda_2}$ using $\lambda_1 = \lambda\bar{\nu}$; $\lambda_2 = \lambda(1-\bar{\nu})$ and write it as $\beta_\lambda(\bar{\nu})$. Our task now reduces to finding the best fit $\lambda$. A $2^{\text{nd}}$ equation imposes $\theta$ as an expectation over the batch quality distribution: $\hat{\theta}(\lambda) = n\int_0^1 \tilde{\theta}(\nu, c)\beta_\lambda(\nu)d\nu \approx \theta$.

Since $\tilde{\theta}$ is computed using a POMDP policy, it will rarely be available in closed form. The integral above is approximated using a numerical algorithm. The best $\lambda$ is found via $\text{argmin}_\lambda|\hat{\theta}(\lambda) - \theta|$ using a linear search, and works instantaneously in practice. Having found a suitable $\lambda$ value, we now bin all $n$ tasks into the $\beta$ distribution to recover the task quality histogram, as desired.

Finally, we describe the procedure for estimation of $\tilde{\theta}(\nu_q, c)$. First we calculate the corresponding $v_q = 0.5(\nu_q + 1)$. Intuitively, $\tilde{\theta}(\nu_q, c)$ assesses the number of ballots taken by the QUALITYMANAGER when its belief $\mathbf{b}_q$ in the current answer is $v_q$ and it has difficulty $d_q$. However, we haven't reconstructed $d_q$ – we use the prior distribution $p(d)$ as its belief on $d_q$. To compute $\tilde{\theta}(\nu_q, c)$ we initiate the QUALITY-MANAGER's POMDP from such a belief state, and compute $\theta_q$ under policy $\pi_q$ using FRONTIERFINDING.

**$\uparrow$ / $\downarrow$ Transitions.** We now describe the transitions for $\uparrow$ action in a given state $(\bar{\nu}, \theta, \tau, c_i)$ to reach $(\bar{\nu}, \theta', \tau, c_{i+1})$. As we change $c$, the main change is in $\theta$. We quantify this change by a simple observation: regardless of pay, the number of ballots taken until this point is fixed. Suppose that at $\tau = 0$ we compute the estimated number of ballots for the batch as $\theta_0(c_i)$. If we have taken $x$ ballots till now, our current estimate of $\theta$ will be simply $\theta_0(c_i) - x$. At pay $c_{i+1}$, our next state's estimate should be $\theta_0(c_{i+1}) - x$. Thus, when changing pay from $c_i$ to $c_{i+1}$ we can simply add $\theta_0(c_{i+1}) - \theta_0(c_i)$ to compute $\theta'$. A similar analysis works for the $\downarrow$ action.

**No-Change Transitions.** The no-change action emulates the setting that the tasks are posted on the platform for $\Delta_\tau$ time at pay $c$. Let the next state be $(\bar{\nu}', \theta', \tau + \Delta_\tau, c)$. Estimation of $\bar{\nu}'$ and $\theta'$ requires a model of task completion. Similar to Gao & Parameswaran (2014), we maintain a pay-dependent ballot completion model, $\Pr(n_b|\Delta_\tau, c)$, as the probability that OCTOPUS will receive $n_b$ ballots in duration $\Delta_\tau$ at pay $c$. However, different from their work, the probability model combines the effects of worker arrival, retention and time taken per task (and not just arrival). Thus, $\theta$ will reduce by $n_b$ with probability $\Pr(n_b|\Delta_\tau, c)$. The cost of the transition will be $-c \cdot n_b$. Since $n_b$ is discretized upto $\Delta_\theta$ granularity, the cost will be rounded off to the nearest bucket.

For updating $\bar{\nu}$, we $\beta$-reconstruct the batch quality histogram using the current state. We then bin $n$ tasks into this histogram, and simulate the TASKSELECTOR on this reconstructed batch. To do this, we first create a POMDP belief state for each reconstructed task by choosing $\nu$ from the histogram to recover $v$, and using the prior difficulty distribution. We then select a task, simulate a ballot using an average worker ($\bar{\gamma}$) and compute the posterior belief. We continue until all $n_b$ ballots are used up. At the end we compute the $\bar{\nu}'$ based on the updated state of the batch. For robustness, we repeat this entire procedure multiple times and average the $\bar{\nu}$ values from different runs.

**Implementation Details.** We construct the whole MDP with all transitions and rewards using simulations and $\beta$-reconstructions as described above. Since time can be unbounded, we keep a max time $\tau_{\max}$ when defining the total state space. We also recognize that the transition from $\bar{\nu}$ to $\bar{\nu}'$ depends on $n_b$ but not on any other part of the state. By caching a table of $(\bar{\nu}, n_b, \bar{\nu}')$ values once, we can save on a lot of simulations when computing the transition function of the no-change action in various states.

We use Value Iteration to learn the policy with $(\bar{\nu} = 0, \theta = n \cdot \max\tilde{\theta}(\nu = 0, c = c_1), \tau = 0, c = c_1)$ as our start state. Intuitively, we are starting at 0 quality, the maximum possible number of ballots to completion, and the lowest price.

In a real execution environment, it is possible that over

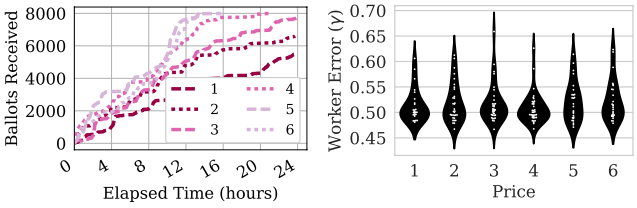



Figure 2: (a) Ballots received vs. elapsed time for different costs (in tenths of a cent); (b) Worker error distribution for different costs are similar.

time, the COSTSETTER's aggregates diverge from the set of QUALITYMANAGER's beliefs in the batch. To improve performance, we synchronize the COSTSETTER's $\bar{\nu}$ and $\theta$ to those of the real batch after every time interval. Experimentally OCTOPUS performs well even without synchronization, indicating that the learned transitions are effective.

In summary, we described the COSTSETTER, an MDP that keeps track of the aggregate statistics $\bar{\nu}$ and $\theta$ for a batch of $n$ tasks, and changes pay to optimize utility $\mathcal{U}$. A key contribution is a novel $\beta$-reconstruction procedure that approximates the batch quality histogram for a state.

## Experiments

We conduct three sets of experiments: (i) experiments on simulated data; (ii) offline evaluation on real data; (iii) live experiments on Amazon MTurk (AMT).

**Model Parameters.** For experiments, we initialize OCTOPUS with a hard deadline utility function: $U_q = -\infty$ if $\tau < \tau_{\max}$; otherwise, $U_q = -\mathcal{P}$ for every incorrect answer and zero for a correct answer. This joint utility combines the utilities from Dai *et al.*'s and Gao *et al.*'s models. Here, penalty $\mathcal{P}$ represents how important quality is to the requester. Notice that since a POMDP doesn't know whether it is submitting the correct answer, it cannot compute the utility exactly. It uses its belief to estimate *expected* utility as $-\mathcal{P}(1 - v_q)$. This linear utility makes the reward computations for COSTSETTER's 'terminate' action simple. Given the $\bar{\nu}$ of the state, the reward is calculated as $-0.5\mathcal{P}(1 - \bar{\nu})$ on termination.

Having a hard deadline makes the COSTSETTER state space finite, since we only consider states with $\tau \leq \tau_{\max}$. We use a time-independent Poisson process for the ballot completion model $\Pr(n_b|\Delta_\tau, c)$. These parameters are illustrative – OCTOPUS can accommodate other utilities/distributions.

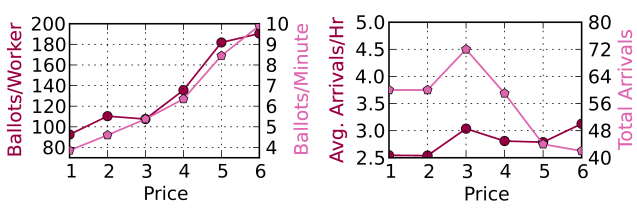



Figure 3: Task completion statistics for our data: (a) Retention & Completion Rate; (b) Arrivals (Rate & Total)

**Baselines.** Our main goal is to compare OCTOPUS's performance with state-of-the-art methods. However, we know of no algorithms that performs direct three-way optimization in a crowdsourced marketplace setting.[2] So we compare against state-of-the-art methods that optimize 2 of the 3 objectives, while giving them the added benefit of our TASKSELECTOR. We compare OCTOPUS to Dai *et al.*'s and Gao *et al.*'s models, which are related to our work. Code was provided by the authors.

All comparisons are on the requester's utility function, $\mathcal{U}$ computed against gold labels with $\mathcal{P} = 200$. We normalize $\mathcal{U}$ so that OCTOPUS always has 1.0 utility, *i.e.* the performance of baselines is represented as a proportion of OCTOPUS.

**Data collection.** We collect data on AMT at 6 pay points using a Twitter Sentiment dataset (Sheshadri and Lease 2013). Workers are asked to classify the sentiments of tweets into either positive or negative sentiment. At each price point ($\$0.001, \$0.002, \ldots, \$0.006$ per ballot), we post 400 tweets, and seek 20 ballots/tweet. A single HIT contains 10 tweets for a worker to solve. 40 tweets are common to all prices for a total of 2200 tweets. Each price is posted on a different weekday at the same time, and remains active for 24 hours. This ensures consistency in data collection across prices and minimizes interaction between different prices.

Figure 2a (task completion rates vs. pay) verifies that higher pricing results in faster task completion: at 0.1 cent, only 5500 ballots are received even after 24 hours, whereas at 0.6 cents, all 8000 ballots are received within 12 hours. We estimate the Poisson parameters using this data.

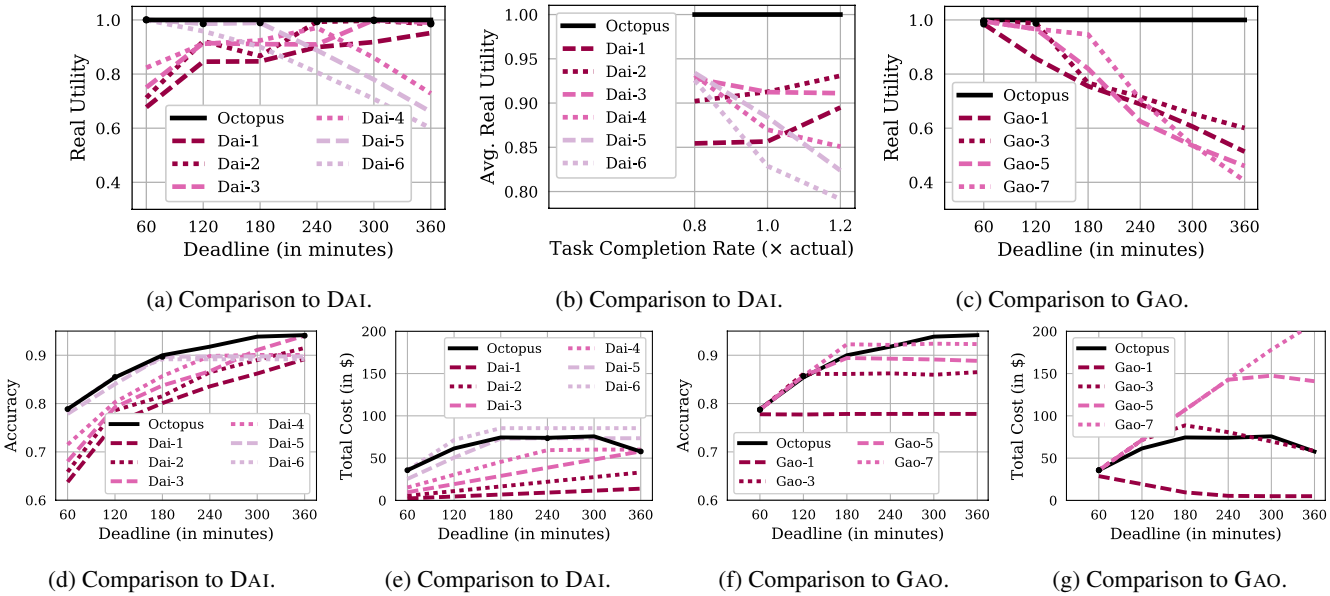
**Worker Retention vs. Arrival.** Contrary to prior work (Faradani, Hartmann, and Ipeirotis 2011; Gao and Parameswaran 2014), we observe that the increase in task completion rate with pay is *predominantly* due to higher worker *retention*, rather than a higher rate or number of worker *arrivals* (possibly due to the large number of HITs that we sought). Figure 3a shows that both retention and task completion rates are highly correlated, doubling as price goes from 0.1 to 0.6 cents. However, Figure 3b shows that the worker arrival rate does not rise much. To the best of our knowledge, there is no prior marketplace model that handles both retention and arrivals.

We also verify that worker quality is independent of pay. We run a K-S test for every pair of costs. We could not reject the null hypothesis (error rates drawn from cost-independent distributions) at $p < 0.05$ (Figure 2b).

## Simulation Experiments

Our main aim through simulations is to assess the quality of $\beta$-reconstruction. We use a variety of parameter settings and

[2] We don't compare to Venetis *et al.* (2012) since they run a tournament for max-finding, different from our task type. They also don't change pay directly or model latency as done by us.




(a) Comparison to DAI. (b) Comparison to DAI. (c) Comparison to GAO.

(d) Comparison to DAI. (e) Comparison to DAI. (f) Comparison to GAO. (g) Comparison to GAO.

Figure 4: Performance of OCTOPUS on simulated data for several deadlines. (a), (c) are plots of the utility achieved. Each data point in (b) represents the performance averaged over all deadlines considered in (a). (d), (f) compare the accuracy achieved while (e), (g) compare the total cost incurred by OCTOPUS and competing baselines. Round dots on the figures denote that the difference between OCTOPUS and the competing baseline (whether better or worse) is statistically insignificant at that point. All other differences are statistically significant.

simulate OCTOPUS with ballot arrivals simulated according to a pay-dependent Poisson process, and answers generated based on worker models. Note that this experiment is *without* any synchronization between the COSTSETTER state and the real batch. Figure 5a compares the $\theta$ values between the system and the batch (for one such setting[3]). Even after 10 time intervals, and across multiple cost changes (the sharp drops in the curve), our tracking is extremely accurate. Figure 5b shows that our $\bar{\nu}$ tracking is extremely effective as well, with very little divergence from the true value. Even when the values do diverge in both cases, they are highly correlated.

High quality tracking of $\theta$ and $\bar{\nu}$ are essential. If we diverge too far from the real values, we would likely optimize the long-term expected reward poorly. Overall, this reaffirms the hypothesis that the COSTSETTER is able to capture the global state of the whole batch using just the aggregate statistics.

[3] 500 tasks with $p(d) = \beta(2.0, 2.0)$, worker errors sampled from $\Gamma(2.0, 0.5)$, $\mathcal{P} = 200$, $\Delta_\tau = 15$ mins, $\Delta_\theta = 10$, $\Delta_{\bar{\nu}} = 100$.

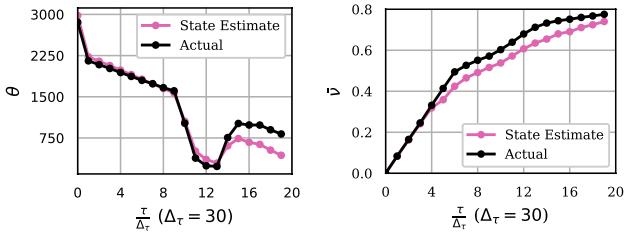



Figure 5: Tracking the real state using $\beta$-reconstruction in the COSTSETTER for the (a) $\theta$ value, and (b) $\bar{\nu}$ value.

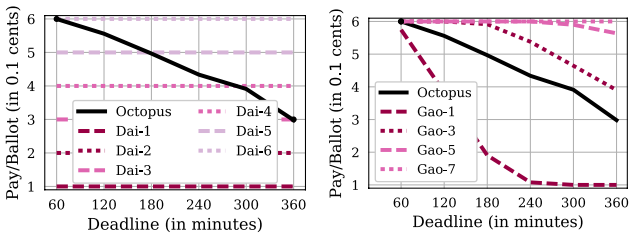
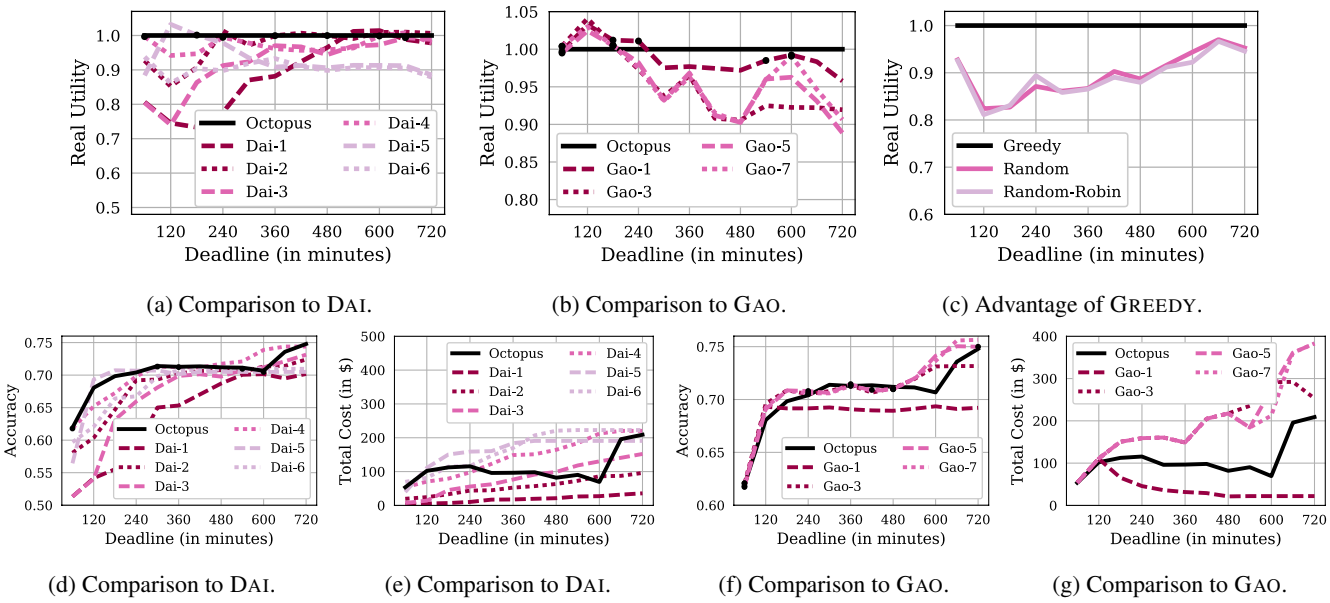



Figure 6: Comparison of the average pay/ballot against (a) DAI, and (b) GAO on simulated data for several deadlines.

Our other goal is to compare OCTOPUS's performance with baselines in simulation, which we do below.

**Comparison to DAI.** Comparing OCTOPUS with DAI highlights the benefit of changing cost on real-world utility. Ideally, our method should be able to vary pay to match or exceed the utility of the static cost baselines. We run both OCTOPUS and DAI with deadlines ranging from 60 to 360 minutes. We run DAI for different static pays, ranging from 1 to 6 (measured in a tenth of a cent), allowing it to take ballots until the deadline. Statistical significance is indicated on the plots.




(a) Comparison to DAI. (b) Comparison to GAO. (c) Advantage of GREEDY.

(d) Comparison to DAI. (e) Comparison to DAI. (f) Comparison to GAO. (g) Comparison to GAO.

Figure 7: Performance of OCTOPUS on real data for several deadlines. (a), (c) are plots of the utility achieved. Each data point in (b) represents the performance averaged over all deadlines considered in (a). (d), (f) compare the accuracy achieved while (e), (g) compare the total cost incurred by OCTOPUS and competing baselines. Round dots on the figures denote that the difference between OCTOPUS and the competing baseline (whether better or worse) is statistically insignificant at that point. All other differences are statistically significant.

Figure 4 shows the comparison. OCTOPUS simultaneously outperforms (or is at par with) all static cost baselines for *every* deadline, whereas each static cost baseline has a 'sweet spot' range of deadlines where it does best. OCTOPUS plans robustly, where despite making pricing decisions 24 times without synchronizing for the 6 hr deadline, it still outperforms DAI. Figure 4b depicts the utility averaged across all deadlines as a function of task-completion rates. As shown, OCTOPUS is robust to different task-completion rates, and continues to outperform DAI on changing them. Figure 4d & 4e explain why OCTOPUS achieves better utility scores than DAI – it maintains very high accuracy while keeping costs reasonable. Qualitatively, for every deadline, we observe that OCTOPUS tends to have an average cost that is close to the best static cost.

**Comparison to** GAO. Comparing with GAO allows us to delineate the effect of the QUALITYMANAGERs while optimizing for batch quality using aggregate statistics.

For fairness, we augment GAO's framework; fixing $r$ ballots/task up-front, and using the same worker response model as OCTOPUS for ballot aggregation. Figure 4c demonstrates OCTOPUS's performance against GAO for different values of $r$. OCTOPUS consistently outperforms GAO for all values of $r$ and across all deadlines. OCTOPUS's performance improves for longer deadlines, due to the higher quality achieved by the QUALITYMANAGER. For instance, OCTOPUS is around 100% better than GAO for the 6 hr deadline. We see that GAO-3 and OCTOPUS incur nearly the same cost (Figure 4g) but OCTOPUS is ∼6% better in terms of accuracy (Figure 4f). Examining Figure 6b reveals that this is in part due to OCTOPUS collecting $\times\frac{1}{3}$ *more* ballots than GAO-3 by changing pay/ballot more intelligently.

It is interesting to note that increasing $r$ does not significantly improve baseline performance. This is due to the fact that the QUALITYMANAGER only takes more ballots on tasks that really need them. For GAO the extra cost spent on every task is not offset by a corresponding increase in quality, especially for tasks that are very easy (don't require $r$ ballots) or too hard (unsolved even with $r$ ballots). In Figure 4f & 4g, we notice that OCTOPUS once again has near-highest accuracy, but spends much less than GAO variants that have higher accuracy than OCTOPUS.

Lastly, Figure 6 demonstrates that OCTOPUS has intuitive behavior – as the deadline length increases, OCTOPUS spends less pay/ballot on average since more time can be taken to finish the batch of tasks.

These experiments demonstrate that OCTOPUS *simultaneously* achieves higher utility than every variant of baseline methods, and does so without sacrificing accuracy or incurring high costs.

## Offline Experiments on Real Data

We run extensive offline experiments on our collected data against the state-of-the-art baselines described earlier.

When comparing algorithms we sample different execution trajectories from the real data by choosing a random ballot for each task (since quality is cost-independent), while keeping ballot arrivals as per the real data. Multiple trajectories help compute statistical significance over algorithms' performances. These experiments synchronize the COST-

SETTER's state with the real batch after every $\Delta_\tau$ time. We use an uninformed uniform distribution for the difficulty prior $p(d)$.

Figure 7a shows OCTOPUS when compared to DAI at different static price points, with the $x$-axis being different $\tau_{\max}$ values. Statistical significance is marked on the plots. OCTOPUS outperforms most DAI costs across all deadlines, with upto $37\%$ increase in real utility. No single static cost is able to match OCTOPUS across all deadlines. This underscores the benefits of changing pay dynamically based on current task completion. In Figure 7d & 7e, we see that OCTOPUS achieves high accuracy at relatively low cost.

Figure 7b compares OCTOPUS with GAO-$r$, with $r$ denoting the static number of ballots per task. We find that OCTOPUS outperforms GAO for most deadlines. Further analysis reveals that our batch exhibits a bi-modal task difficulty distribution – no algorithm exceeds around $76\%$ accuracy, while getting $70\%$ is easy with $\sim$1 ballot per task (in Figure 7f, we see a sharp jump in accuracy only when the deadline is long enough to take a lot of ballots). For shorter deadlines, where solving difficult tasks is infeasible, GAO outperforms OCTOPUS slightly, since OCTOPUS optimizes with respect to a uniform prior. For longer deadlines, estimates of task difficulties are refined by the QUALITYMANAGERS, and OCTOPUS gives large gains.

Lastly, OCTOPUS outperforms TASKSELECTOR baselines where GREEDY task selection is replaced by a random policy (RANDOM), or a single round robin followed by random selection (RANDOM-ROBIN), by large margins (Figure 7c). Note also that the baselines start to converge over time, as the advantage of doing intelligent task selection diminishes when nearly all tasks are run till completion.

Overall, we find that OCTOPUS learns robust policies, consistently outperforming all baselines.

## Live Online Experiments

Lastly, we deploy OCTOPUS on AMT to test performance in a dynamic, online setting, as well as gain qualitative insight into worker behavior. In this, we keep exactly 3 HITS (of 10 tasks each) on AMT at a time. If a worker accepts a HIT, another one is posted immediately – this enables greedy task routing while ensuring full power of worker parallelism. After every $\Delta_\tau$ mins, all available HITs are taken down and reposted with the new price output by OCTOPUS. OCTOPUS's policies are learned using task completion rates estimated from real data earlier. At runtime, querying OCTOPUS is instantaneous. We solve a batch of 500 tweets for 3 deadlines – 1, 2, and 4 hours. We compare against GAO-1, the best baseline in offline expts in Table 2.

For the 1 hr deadline, OCTOPUS maintains pay at 0.5 cent/ballot, before decreasing it to 0.1 cent/ballot in the last 15 minutes. This allows OCTOPUS to receive around 1 ballot/task; more ballots stagnate $\mathcal{U}$ for short deadlines due to conflicting workers, and OCTOPUS prefers to save money to optimize utility. On the other hand, GAO maintains pay at 0.6 cent throughout to also ensure it receives 1 ballot/task, but is unlucky in the responses it receives. In terms of decision making, the superiority of OCTOPUS is clear, since it recognizes the danger posed by disagreement on task quality estimates.

| Method | OCTOPUS | | | GAO ($r=1$) | | |
|---|---|---|---|---|---|---|
| Deadline | Cost | Acc. | $\mathcal{U}$ | Cost | Acc. | $\mathcal{U}$ |
| 1 hr | \$2.63 | 74.4% | **-25.9** | \$2.68 | 71.4% | -28.9 |
| 2 hrs | \$3.17 | 77.2% | **-23.1** | \$1.72 | 73.8% | -26.4 |
| 4 hrs | \$6.23 | 77.4% | **-23.2** | \$0.50 | 74.4% | -25.6 |

Table 2: Online performance comparison of OCTOPUS and GAO ($r=1$) for a single trial. $\mathcal{U}$ is shown in thousands.

In the 2 hr deadline, OCTOPUS once again increases pay initially, but gets more ballot arrivals than expected. In response, after 45 minutes OCTOPUS decreases pay so that it can take a higher number of ballots for difficult tasks, getting a substantial accuracy and utility improvement over GAO.

For the 4 hr deadline, OCTOPUS is aggressive in trying to solve all tasks till completion. Due to the bi-modal difficulty of the tasks, workers provide several conflicting ballots on harder tasks, which the TASKSELECTOR prefers to re-route for utility gain. The overall accuracy increases marginally over the 2 hour deadline.

Qualitatively, workers respond as predicted – flocking to the tasks when pay was set at 0.5 cent or more, and staying away at very low pay. Workers respond naturally to the price changing algorithm; dropping out immediately if the pay is suddenly lowered, and coming back if it is increased once again. No worker complained about the fluctuating pay.

## Discussion

In this work, we focused on experiments with OCTOPUS in a fixed time deadline setting to compare with past work. Extension to the popular fixed budget setting (where $\sum_q C_q$ is constrained) is simple: (i) never synchronize $\theta$ so that its value in any state is simply the start value of $\theta$ (which is fixed and known) minus the ballots received, making it a proxy for cost incurred so far; (ii) modify the COSTSETTER's reward to give a $-\infty$ reward for total cost (now computable using $\theta$) exceeding the budget. Another extension involves using a different formulation of the QUALITYMANAGER; any algorithm that defines an appropriate policy and from which we can compute $\theta$ and $\bar{\nu}$ is suitable.

# Conclusion

We present OCTOPUS, one of the first AI agents for a 3-way optimization of total cost, work quality and completion time in crowdsourcing. The agent combines three different subagents that control quality per task, select the best next task and set pay for the whole batch. A key technical contribution is the computation of aggregate statistics of the quality and completeness of the whole batch – this is used as the state for best setting the next pay.

OCTOPUS outperforms state-of-the-art baselines in a variety of simulated and real world settings, demonstrating the superiority of our approach. We also showcase OCTOPUS's real world applicability by deploying it directly on AMT. In the future, we hope to develop general purpose formulations

of OCTOPUS as a plug-and-play architecture for practitioners.

## Acknowledgments

This work is supported by Google language understanding and knowledge discovery focused research grants, a Bloomberg award, a Microsoft Azure sponsorship, and a Visvesvaraya faculty award by Govt. of India to the third author. We thank Chris Lin, Yihan Gao and Aditya Parameswaran for sharing code, and all the AMT workers who participated in our experiments.

## References

Ambati, V.; Vogel, S.; and Carbonell, J. G. 2011. Towards task recommendation in micro-task markets. In *AAAI Workshop on Human Computation*.

Bragg, J.; Kolobov, A.; Mausam; and Weld, D. S. 2014. Parallel task routing for crowdsourcing. In *HCOMP*.

Bragg, J.; Mausam; and Weld, D. S. 2013. Crowdsourcing multi-label classification for taxonomy creation. In *HCOMP*.

Chatterjee, K.; Majumdar, R.; and Henzinger, T. A. 2006. Markov decision processes with multiple objectives. In *STACS 2006*. Springer.

Dai, P.; Lin, C. H.; Mausam; and Weld, D. S. 2013. Pomdp-based control of workflows for crowdsourcing. *Artif. Intell.* 202:52–85.

Dai, P.; Rzeszotarski, J. M.; Paritosh, P.; and Chi, E. H. 2015. And now for something completely different: Improving crowdsourcing workflows with micro-diversions. In *CSCW*. ACM.

Dai, P.; Mausam; and Weld, D. S. 2011. Artificial intelligence for artificial artificial intelligence. In *AAAI*.

Difallah, D. E.; Catasta, M.; Demartini, G.; and Cudré-Mauroux, P. 2014. Scaling-up the crowd: Micro-task pricing schemes for worker retention and latency improvement. In *HCOMP*.

Faradani, S.; Hartmann, B.; and Ipeirotis, P. G. 2011. What's the right price? pricing tasks for finishing on time. In *AAAI Workshop on Human Computation*.

Gao, Y., and Parameswaran, A. G. 2014. Finish them!: Pricing algorithms for human computation. *PVLDB*.

Golovin, D., and Krause, A. 2011. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research*.

Haas, D.; Wang, J.; Wu, E.; and Franklin, M. J. 2015. Clamshell: speeding up crowds for low-latency data labeling. *VLDB*.

Hansen, E. A., and Zilberstein, S. 2001. Monitoring and control of anytime algorithms: A dynamic programming approach. *Artificial Intelligence* 126(1-2):139–157.

Ipeirotis, P. G., and Gabrilovich, E. 2014. Quizz: Targeted crowdsourcing with a billion (potential) users. In *WWW*. ACM.

Kamar, E.; Kapoor, A.; Horvitz, E.; and Redmond, W. 2013. Lifelong learning for acquiring the wisdom of the crowd. In *IJCAI*. Citeseer.

Karger, D. R.; Oh, S.; and Shah, D. 2014. Budget-optimal task allocation for reliable crowdsourcing systems. *Operations Research*.

Kearns, M. J.; Mansour, Y.; and Ng, A. Y. 1999. Approximate planning in large pomdps via reusable trajectories. Citeseer.

Kobren, A.; Tan, C. H.; Ipeirotis, P.; and Gabrilovich, E. 2015. Getting more for less: optimized crowdsourcing with dynamic tasks and goals. In *WWW*.

Lin, C. H.; Mausam; and Weld, D. S. 2012. Crowdsourcing control: Moving beyond multiple choice. In *UAI*.

Mason, W., and Watts, D. J. 2010. Financial incentives and the performance of crowds. *ACM SigKDD Explorations Newsletter* 11(2):100–108.

Oleson, D.; Sorokin, A.; Laughlin, G. P.; Hester, V.; Le, J.; and Biewald, L. 2011. Programmatic gold: Targeted and scalable quality assurance in crowdsourcing. *Human computation*.

Parameswaran, A. G.; Garcia-Molina, H.; Park, H.; Polyzotis, N.; Ramesh, A.; and Widom, J. 2012. Crowdscreen: algorithms for filtering data with humans. In *SIGMOD*.

Rajpal, S.; Goel, K.; and Mausam. 2015. Pomdp-based worker pool selection for crowdsourcing. *CrowdML Workshop, ICML*.

Rzeszotarski, J. M.; Chi, E.; Paritosh, P.; and Dai, P. 2013. Inserting micro-breaks into crowdsourcing workflows. In *HCOMP*.

Shahaf, D., and Horvitz, E. 2010. Generalized task markets for human and machine computation. In *AAAI*.

Sheshadri, A., and Lease, M. 2013. Square: A benchmark for research on computing crowd consensus. In *HCOMP*.

Smith, T., and Simmons, R. 2012. Point-based pomdp algorithms: Improved analysis and implementation. *arXiv preprint arXiv:1207.1412*.

Teneketzis, D., and Ho, Y.-C. 1987. The decentralized wald problem. *Information and Computation* 73(1):23–44.

Venetis, P.; Garcia-Molina, H.; Huang, K.; and Polyzotis, N. 2012. Max algorithms in crowdsourcing environments. In *Proceedings of the 21st international conference on World Wide Web*, 989–998. ACM.

Weld, D. S.; Mausam; Lin, C. H.; and Bragg, J. 2015. Artificial intelligence and collective intelligence. *Handbook of Collective Intelligence*.

Welinder, P., and Perona, P. 2010. Online crowdsourcing: rating annotators and obtaining cost-effective labels.

Welinder, P.; Branson, S.; Perona, P.; and Belongie, S. J. 2010. The multidimensional wisdom of crowds. In *NIPS*.

Whitehill, J.; Wu, T.-f.; Bergsma, J.; Movellan, J. R.; and Ruvolo, P. L. 2009. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *NIPS*.