Finish Them!: Pricing Algorithms for Human Computation

Yihan Gao
University of Illinois (UIUC)
Urbana, Illinois
ygao34@illinois.edu

Aditya Parameswaran
University of Illinois (UIUC)
Urbana, Illinois
adityagp@illinois.edu

ABSTRACT

Given a batch of human computation tasks, a commonly ignored aspect is how the price (i.e., the reward paid to human workers) of these tasks must be set or varied in order to meet latency or cost constraints. Often, the price is set up-front and not modified, leading to either a much higher monetary cost than needed (if the price is set too high), or to a much larger latency than expected (if the price is set too low). Leveraging a pricing model from prior work, we develop algorithms to optimally set and then vary price over time in order to meet a (a) user-specified deadline while minimizing total monetary cost (b) user-specified monetary budget constraint while minimizing total elapsed time. We leverage techniques from decision theory (specifically, Markov Decision Processes) for both these problems, and demonstrate that our techniques lead to upto 30% reduction in cost over schemes proposed in prior work. Furthermore, we develop techniques to speed-up the computation, enabling users to leverage the price setting algorithms on-the-fly.

1. INTRODUCTION

Crowdsourcing is often used to process and reason about unstructured data such as images, videos, and text. The data thus generated is typically used as training data for machine learning algorithms in applications such as content moderation (i.e., determining if images are suitable to be viewed by a general audience), spam detection, search relevance estimation, information extraction, and entity resolution. In fact, all of the following companies employ crowdsourcing frequently at a large scale to repeatedly process unstructured data: Google [6], Ebay [2], Microsoft [5], LinkedIn [7], Facebook [7], Yahoo! [9], Twitter [1], Cisco [6], and Yelp [46].

Even though crowdsourcing is often used in industry and academia, and has been the subject of many academic papers studying trade-offs between cost, latency and accuracy [16, 25, 36, 37, 38], there is little to no work on *task pricing* and its impact on overall cost and latency: that is, how the price of tasks (i.e., the monetary reward paid to workers on completion) must be set or varied in order to meet cost or latency constraints. Often, the price is set up-front and not modified, leading to either a much higher monetary cost than needed (if the price is set too high), or to a much larger latency

than expected (if the price is set too low). As a result, anecdotally, pricing is seen as somewhat of a "dark art".

In this paper, we wish to address the following question: *Given that we have n fixed tasks, how should we vary their price or reward over time so that they get completed by a certain deadline at the least cost possible?* Intuitively, it seems that we may want to start with a low price initially, and then increase it gradually as it gets closer to the deadline. However, there has been no work demonstrating that such strategies will indeed yield good results in practice. Furthermore, there are a number of additional complications, even given this very simple scheme:

- What should we price tasks initially?
- How can we adapt our price setting to the rate at which tasks are picked up? What if tasks get picked up very quickly at the initial price; should we lower the price, should we keep it same, or should we increase it? What if the opposite happens — that is, tasks get picked up very slowly at the initial price?
- At what time points should we increase the price? Increasing it too frequently may lead to computationally more expensive decision making (as we will see subsequently), but increasing it too infrequently may result in much higher costs.
- At what granularities do we increase the price, and how much does this affect overall cost?
- Should we price all the tasks the same, or should we price tasks differently?
- What if we had a fixed budget, and instead wanted to reduce total latency. Would similar techniques apply then? Would varying price help at all?
- How do we ensure that our pricing schemes can be computed within a reasonable time, and how can we speed them up?
- How are our algorithms impacted by inaccuracies in estimates of the marketplace dynamics?

In prior work, Faridani et al. [15] develop a model for latency in crowdsourcing applications based on Non-Homogeneous Poisson Processes. They then use this model to describe a simple scheme based on binary search for pricing tasks to complete by a deadline. However, their scheme is *not optimal*, that is, it wastes far too much monetary cost. In this paper, we leverage their model and instead focus on the optimization problem of minimizing cost while meeting the deadline with high probability. Overall, our techniques yield rich dividends — we get up to a 30% reduction in cost as compared to their scheme on realistic crowdsourcing workloads. This represents a significant reduction in cost especially for users who run large crowdsourcing workloads with strict deadlines.

In this paper, we develop algorithms for two optimization problems, given a set of tasks: one, minimizing cost while meeting time requirements, and second, minimizing latency while meeting monetary budget requirements. For the first, we develop an algorithm

based on decision theory that gives us near-optimal results. For the second, we develop a solution that uses linear programming, that can be shown to be optimal under some assumptions. A crucial concern for us is that the computation is as little as possible, and we propose various speed-up techniques for this purpose.

The contributions of this paper are as follows:

- We describe the two problems that we study in this paper formally in Section 2.
- We develop optimized pricing algorithms that meet a fixed time deadline in Section 3. Since these algorithms could be computationally expensive, we describe techniques to reduce the complexity of these algorithms.
- We develop optimized pricing algorithms that meet a fixed monetary cost budget in Section 4.
- We demonstrate that our pricing algorithms achieve a reduction in cost of up to 30% over prior work on simulations with real data from a crowdsourcing marketplace, as well as live experiments on the same marketplace in Section 5. Furthermore, we demonstrate that the algorithms are remarkably robust to errors in the estimates of parameters of the tasks and the marketplace.

We cover related work in Section 6 and conclude in Section 7.

2. PRELIMINARIES

In this section, we describe the basic model that we will leverage to design optimized pricing algorithms.

We operate on a crowdsourcing marketplace, such as Mechanical Turk [3]. In any crowdsourcing marketplace, users (or requesters) post *tasks*, often many at a time, and set a monetary *price or reward* for them. At any point, there are many tasks on offer in the marketplace. Human *workers* arrive at the marketplace at any time, and can leave at any time. When on the marketplace, workers can choose to work on any of the available tasks. They are allowed to work on a single task at a time. Once they complete a task, they will receive the reward or price assigned for the task by the requester.

In a marketplace, the reward of each task is positively correlated with the completion rate: the higher the reward, the shorter the completion time. However, in order to determine the best trade-off between cost and completion time, this relationship must be precisely quantified. For example, we must be able to answer questions like: if we adjust the reward per task from $0.25 to $0.3, how much do we gain in terms of task completion rate? To answer these questions, we need a formal model for reasoning about the crowdsourcing marketplace.

In previous work, Faridani et al. [15] studied the problem of modeling crowdsourcing marketplace dynamics; the dynamics is modeled using two independent processes: A *Non-Homogeneous Poisson Process* is used to model the worker arrivals in the market, and a *Discrete Choice Model* is used to model how workers choose between tasks in the marketplace. We adopt the same model in this paper, and focus instead on the optimal pricing problem. To enable this paper to be self-contained, we describe the worker arrival model in Section 2.1, and the task choice model in Section 2.2. These mathematical models will be used to define the pricing problem formally in Section 2.3.

2.1 Worker Arrival Model

Faridani et al. [15] show that the arrival of workers in a crowdsourcing marketplace follows a *Non-Homogeneous Poisson Process(NHPP)*. Note that the standard Poisson process is commonly used to characterize the counting process of stochastically occurring events. The Poisson process has a fixed rate λ. NHPP is a generalization of the Poisson process, with a rate parameter $\lambda(t)$, a

function of time [43]. In a NHPP, the number of events that occur during any period of time $[S, T]$ follows a Poisson distribution:

$$\mathbf{N}[S, T] \sim \mathbf{Pois}(\cdot | \lambda = \int_{t=S}^{T} \lambda(t)) \qquad (1)$$

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

Estimating the arrival-rate function $\lambda(t)$ of a NHPP is more difficult than that for a Homogeneous Poisson Process because of the infinite dimensionality of the arrival-rate parameter $\lambda(t)$. Therefore, a common approach is to assume a parametric form for $\lambda(t)$. For instance, Massey et al. [31] used a piece-wise linear function to approximate the traffic of telecommunication systems.

Figure 1 depicts the number of tasks completed every 6 hours for a time range of 4 weeks in Mechanical Turk. The figure depicts that the variation of worker arrivals follows a process that approximately repeats every week. In this paper, we assume that the arrival-rate function $\lambda(t)$ is periodic, and the variations in the number of worker arrivals are all due to the randomness of the Poisson process. Given historical data, the arrival-rate function $\lambda(t)$ can be estimated and used to predict arrival rates in the future. Faridani et al. [15] provide techniques for learning the $\lambda(t)$ function, and demonstrate the accuracy of these techniques. In this paper, we leverage these techniques, and assume that $\lambda(t)$ is known. As we will see in our experimental results in Section 5, our pricing strategies are not very sensitive to mistakes in the estimation of $\lambda(t)$.
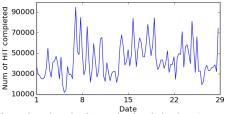

Figure 1: The total number of tasks completed each day from January 1st, 2014 to January 28th, 2014. Retrieved from http://www.mturk-tracker.com

Note that the NHPP models the arrival of workers to the entire marketplace, and does not capture whether those workers decide to work on our specific task. An independent Bernoulli process can be used to model whether each worker (who arrives at the marketplace) will decide to work on our task. In other words, we assume that each arrived worker has an independent probability p of picking our task. Therefore if in any period of time the number of workers arrived at marketplace is X, then, assuming there are adequate tasks on offer, the number of workers from those who choose to work on our tasks will follow a Binomial distribution $\mathbf{Bin}(X, p)$. The value of p, the task acceptance probability, is not directly observable. We describe how it is related to the price or reward for the task, and how it can be estimated in Section 2.2.

The task completion process is then a composition of NHPP and an independent Bernoulli process. In Statistics literature, such a process is called a *Thinned Non-Homogeneous Poisson Process* [43]. A Thinned NHPP is also a NHPP with a modified arrival-rate function $\lambda'(t) = \lambda(t)p$.

2.2 Task Acceptance Probability Estimation

Faridani et al. [15] used a *Discrete Choice Model* to characterize how workers select tasks from the marketplace. In economics, *Discrete Choice Models* are used to estimate the probability of consumers choosing a specific product among a range of alternatives [32]. Discrete Choice Models can be explained by utility theory: each worker chooses the task in the marketplace to maximize the utility (or net benefit) obtained. Workers may have different perceptions of his/her utility: it could depends on various factors

such as hourly wage, number of tasks, task type, easiness of the tasks, or the knowledge gain during the process of finishing a task. Utility can not be directly observed, the only aspect that can be observed is the worker's behavior in the marketplace.

Under this model, the task acceptance probability parameter p is simply the probability that the utility of our task exceeds the utility of every other task in the marketplace. Let U_i be the utility of task i in the marketplace based on some worker's perception, and without loss of generality we assume the utility of our task is U_1. Then $p = \mathbf{Pr}(U_1 > \max_{i \neq 1} U_i)$. In the *Conditional Logit Model* [32] [15], the utility U_i of ith task has the following expression: $U_i = \beta^T z_i + \epsilon_i$, where z_i are all observable attributes that may affect the utility of the task, and ϵ_i accounts for all unobserved factors that may affect the utility. In the model, the utility U_i is assumed to be linearly correlated with all observed attributes with the shared coefficient vector β. The parameters ϵ_i are assumed to be independent with each other and follow the Gumbel distribution. Based on these assumptions, it can be derived that the probability of choosing each task follows a *Multinomial Logit Distribution*:

$$p = \mathbf{Pr}(U_1 > \max_{i \neq 1} U_i) = \frac{\exp(\beta^T z_1)}{\sum_i \exp(\beta^T z_i)}$$

Now if we are able to change our task reward c, then the attribute vector of our task z_1 and the task acceptance probability p will also change accordingly:

$$p(c) = \frac{\exp(\beta^T z_1(c))}{\exp(\beta^T z_1(c)) + \sum_{i \neq 1} \exp(\beta^T z_i)} \quad (2)$$

Equation (2) captures how task acceptance probability is related to task price or reward. Faridani et al. [15] suggest using this equation directly in order to calculate the task acceptance probability, with parameters β estimated from historical marketplace data using logistic regression. Another approach is to assume a parametric form of task acceptance probability function, and estimate parameters during a separate training phase. If we assume that the utility of our task is a linear function of task reward c, and that the sum of exponentials of the utilities of the other tasks is a fixed constant, then Equation (2) can be rewritten as:

$$p(c) = \frac{\exp\{\frac{c}{s} - b\}}{\exp\{\frac{c}{s} - b\} + M} \quad (3)$$

Hence if we have some training data (e.g., estimated value of $p(c)$ for different task reward c), then parameters s, b, M can be estimated by statistical regression methods.

However, note that the inference of the mapping function $p(c)$ is not the focus of our paper. Here, we will assume that the expression of $p(c)$ is already known. We will then use this expression to determine the optimal reward for each task in various scenarios.

2.3 Problem Statement

Our goal is to design pricing algorithms for batch of N identical crowdsourcing tasks. The user may specify either a monetary budget restriction (that is, the algorithm must ensure that all tasks are completed within a certain expected cost), or a time deadline (that is, the algorithm must ensure that all tasks are completed within a certain time). The unconstrained variable (monetary cost or overall time) is minimized.

Following our discussion in the previous section, we model worker arrivals to the marketplace as a *Non-Homogeneous Poisson Process* with a known arrival-rate parameter $\lambda(t)$. Each worker will pick up our task and complete it with probability $p(c)$, where the value of p depends on the reward c (typically in cents or dollars) for each

task in our batch of tasks. The form of the mapping function $p(c)$ from task reward c to task acceptance probability p is assumed to be known: thus, we expect our techniques to be leveraged when the user ends up repeating similar tasks many times over a long period so that such history is available. This is not a drastic assumption to make: many companies, including Google, Ebay, Yahoo!, and Microsoft, repeatedly use human workers for tasks such as content moderation, categorization, spam detection, and search relevance.

At any time, we can monitor the number of remaining uncompleted tasks n. The task reward c can be changed at any time, and the task acceptance probability p will change accordingly. Note that some marketplaces may impose a minimum time only after which the task reward may be changed, and our algorithms adapt to that scenario as well. Overall, at any time t, the completion of tasks follows a NHPP with rate $\lambda(t)p(c)$, and for each completed task, c units of monetary compensation are paid based on the task reward at that time.

Then, the problem is to determine and dynamically vary the rewards for each as yet unsolved task, such that the total monetary cost expended and the total time used for completing N tasks are minimized. We focus on two scenarios:

- **Fixed Deadline Pricing** (Section 3): In this scenario, the total time used to complete all N tasks must be less than a deadline T. The goal is then to minimize the expected total expenditure.
- **Fixed Budget Pricing** (Section 4): In this scenario, the total monetary budget B for tasks is fixed upfront. The goal is then to minimize the expected total time to complete all tasks.

In our extended technical report [17], we describe a number of straightforward generalizations, including optimizing combinations of deadline and budget, capturing multiple task types, and incorporating accuracy and difficulty. In short, optimizing for combinations of deadline and budget is actually a simplification of the techniques described in the paper; capturing multiple tasks types is a straightforward generalization of the results of [36]; and incorporating accuracy and difficulty involves using results from [36] to first optimize for accuracy, then using that information while developing our techniques—we develop one exact, but intractable technique and two approximations for this case.

3. FIXED DEADLINE PRICING STRATEGY

It is common for task requesters in a crowdsourcing marketplace to require their tasks to be completed before a certain deadline. Under this scenario, the reward for each task in the batch of tasks should be as low as possible while making sure that all tasks can be completed before deadline.

In Faridani's work [15], a binary search process is used to find the smallest fixed task reward such that the total expected completion time is before the deadline. However, as implied by the NHPP worker-arrival model and also demonstrated in Figure 1, the task completion process is highly non-deterministic. Therefore, a dynamic pricing strategy should perform much better in terms of overall cost in this scenario: If the rate at which tasks are picked up by workers is faster than expected, we could decrease the reward for the remaining tasks to save money; on the other hand, if the tasks are picked up slower than expected, we could increase the reward to attract more workers to our tasks.

In this section, we design a pricing algorithm to determine how to set the reward for each task at each time point to minimize the expected total monetary cost, while meeting time constraints. We begin by modeling our decision process as a Markov process and use the model to present our basic pricing algorithm in Section 3.1. Since these algorithms may be expensive to compute, we present techniques that can help speed up the computation in Section 3.2.

3.1 Markov Decision Process-based Solution

Discretization: Although in principle we may be able to change the task reward c at any time, utilizing this freedom while designing pricing strategies would result in an intractable number of time points at which decisions need to be made. Instead, we discretize the total time before the deadline (i.e., the time between when the tasks were submitted to the marketplace and the deadline) into a number of equal-sized intervals. As we will see later on, beyond a point, discretization does not help, and therefore restricting our pricing algorithms to make decisions only at discrete time intervals does not affect the overall monetary cost, while significantly reducing the computation involved.

We partition all available time (from $t = 0$, i.e., start time, to $t = T$, i.e., the deadline) $[0, T]$ into N_T small intervals: $[0, T/N_T)$, $[T/N_T, 2T/N_T), \ldots, [T - T/N_T, T)$ and further enforce that the reward c for tasks may only be changed at the start of an interval.

State Space: After discretization, we can represent the state of processing of the batch of tasks at any time interval using a finite Markov chain. The states in this Markov chain are represented by a pair (n, t), where n is the remaining unsolved tasks and t is the index of current time interval. The initial state is $(N, 0)$ and all states in the form of (n, N_T) are final states (Recall that N_T is the total number of time intervals).

An illustration of the state diagram is shown in Figure 2. The states are represented on a grid, where the number of unsolved tasks increases along the y-axis, and the number of time intervals elapsed increases along the x-axis. Our goal is then to set the prices $c_{n,t}$ upfront for all n, t, such that we have as few unsolved tasks as possible when $t = N_T$.

Transitions: Based on Equation (1), X_t, the number of tasks completed during the tth time interval follows a Poisson distribution: $X_i \sim \mathbf{Pois}(\cdot|\lambda = \lambda_t p(c_t))$ where c_t is the task reward in ith time interval, and λ_t is the total expected number of workers who arrived at marketplace during the tth time interval:

$$\lambda_t = \int_{s=(t-1)T/N_T}^{tT/N_T} \lambda(s)ds \qquad (4)$$

At state (n, t), say the task reward is set to be $c_{n,t}$; then, the transition probability between states is:

$$\mathbf{Pr}\{(n, t) \to (n - s, t + 1)|c_{n,t}\} = \mathbf{Pois}(s|\lambda = \lambda_t p(c_{n,t})) \qquad (5)$$

$$= e^{-\lambda_t p(c_{n,t})} \frac{(\lambda_t p(c_{n,t}))^s}{s!} \qquad (6)$$

where λ_i is defined in Equation (4) and $p(c_{n,t})$ is the task acceptance probability for the task reward $c_{n,t}$. The transition probability is slightly different when we are close to completion:

$$\mathbf{Pr}\{(n, t) \to (0, t + 1)|c_{n,t}\} = \mathbf{Pr}(\mathbf{Pois}(\cdot|\lambda = \lambda_t p(c_{n,t})) \geq n)$$
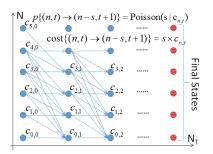

Figure 2: State diagram of Markov Decision Process. Some possible transitions are omitted in the figure for clarity.

Costs: In our problem, the transition cost between states is the total rewards paid for tasks completed in each time interval:

$$\mathbf{cost}\{(n, t) \to (n - s, t + 1)|c_{n,t}\} = sc_{n,t} \qquad (7)$$

For the final states (n, N_T), we assign a fixed penalty for each of the remaining unsolved tasks: $\mathbf{cost}\{(n, N_T)\} = n \times \mathbf{Penalty}$, where the value of the parameter **Penalty** could be based on actual expenses needed to complete them post deadline (possibly by the task requester themselves), or simply be set large enough to ensure that with high probability no task will remain uncompleted.

In the above formulation, we have combined the number of unsolved tasks by the deadline and the monetary cost incurred into one objective. In our extended technical report [17], we consider a range of other constraints and objectives, including:

- Optimizing monetary cost while enforcing a constraint on the probability that there are additional tasks left after the deadline
- Optimizing monetary cost while enforcing a constraint on the expected number of tasks left after the deadline

The techniques for these formulations are straightforward modifications of the techniques for the current formulation.

Markov Decision Processes: The problem of determining optimal task reward $c_{n,t}$ in tth time interval for state (n, t) can be viewed as a *Markov Decision Process (MDP)*. MDPs are commonly used to model optimization and decision making problems in a discrete time stochastic environment. The goal is of MDP optimization to determine the policy for every state to minimize the expected overall cost (in our problem it corresponds to determining the optimal task reward $c_{n,t}$ for each state).

Dynamic Programming: The above MDP optimization problem can be solved by *Dynamic Programming (DP)*. Let $\mathbf{Opt}(n, t)$ denote the minimum expected total cost for all remaining n tasks for the state (n, t), and $\mathbf{Price}(n, t)$ denote the corresponding optimal reward for each task. Then $\mathbf{Opt}(n, t)$ and $\mathbf{Price}(n, t)$ satisfy the following equations.

$$\mathbf{Opt}(n, t) = \min_c \sum_{s=0}^{n} [\mathbf{Opt}(n - s, t + 1) + sc] \\ \times \mathbf{Pr}\{(n, t) \to (n - s, t + 1)|c\}$$

$$\mathbf{Price}(n, t) = \arg\min_c \sum_{s=0}^{n} [\mathbf{Opt}(n - s, t + 1) + sc] \\ \times \mathbf{Pr}\{(n, t) \to (n - s, t + 1)|c\}$$

The values of $\mathbf{Opt}(n, t)$ and $\mathbf{Price}(n, t)$ can be sequentially determined. That is, we start at $(\cdot, N_T)$, and work our way backwards using the equations above. Once we have computed the optimal **Opt** and **Price** for all $(\cdot, t + 1)$, we can use the equations above to compute it for all $(\cdot, t)$ — the optimal $c_{n,t}$ can be found by considering all possible price values since it needs to be an integral multiples of a minimal unit of price (In Amazon Mechanical Turk it is 1 cent). The pseudocode for the DP algorithm can be found in the extended technical report [17].

3.2 Speed-up Techniques

The DP algorithm has a time complexity of $O(N^2 N_T C)$, where C is the number of price choices we want to consider, which is intractable when N is large or when N_T or C are fine-grained. Here we discuss some techniques to speed up the algorithm.

Poisson Distribution Truncation: Notice that while making pricing decisions, the DP algorithm enumerates all possible number of tasks s that can be picked up by workers during each time interval. However, for large s, the probability that more than s tasks are

completed in one time interval:

$$\mathbf{Pr}(\mathbf{Pois}(\cdot|\lambda) \geq s) = \sum_{k \geq s} e^{-\lambda} \frac{\lambda^k}{k!} \leq e^{-\lambda} \frac{\lambda^s}{s!} \frac{s}{s-\lambda}$$

becomes negligible, and thus the contribution of those terms in DP update formulas will also become negligible.

In practice, we could set a threshold ϵ for the probability $\mathbf{Pr}$ $(\mathbf{Pois}(\cdot|\lambda) \geq s)$. If for some s_0, $\mathbf{Pr}(\mathbf{Pois}(\cdot|\lambda) \geq s_0)$ is less than the threshold ϵ, all the terms $s > s_0$ can be ignored safely. Table 1 shows the value of s_0 for $\epsilon = 10^{-9}$ and different values of λ.

Threshold ϵ	Poisson mean λ	s_0
10^{-9}	10	35
10^{-9}	20	53
10^{-9}	50	99

Table 1: The value of s_0 for different thresholds ϵ and Poisson distribution means

The next theorem provides an upper bound of error produced by Poisson Distribution Truncation:

THEOREM 1. *The exact optimal total cost* $\mathbf{Opt}(n, t)$ *and estimated value of optimal total cost* $\mathbf{Est}_{trunc}(n, t)$ *using Poisson Distribution Truncation and the exact total cost* $\mathbf{Cost}_{trunc}(n, t)$ *based on the optimal policy obtained using Poisson Distribution Truncation satisfies the following inequality:*

$$\mathbf{Est}_{trunc}(n, t) \leq \mathbf{Opt}(n, t) \leq \mathbf{Cost}_{trunc}(n, t)$$
$$\leq \mathbf{Est}_{trunc}(n, t) + \epsilon n (N_T - t) C$$

where C is the upper bound of task reward in any state. In particular, $|\mathbf{Opt}(N, 0) - \mathbf{Cost}_{trunc}(N, 0)| \leq \epsilon N N_T C$

The proof of this and subsequent theorems are omitted due to space considerations and can be found in our technical report [17].

Monotonicity of Pricing Decision: Another speed-up technique relies on the following natural conjecture:

CONJECTURE 1. *The optimal reward* $\mathbf{Price}(n, t)$ *for each task is non-decreasing with respect to n for any fixed value of t.*

Intuitively, this conjecture says that with a fixed deadline, the more remaining tasks we have, the higher reward we should set for each task. Over repeated trials with many different values of λ, N, N_T, we tried generating optimal strategies (using the basic DP algorithm described in the previous section), and the optimal strategies never violate the preceding conjecture.

If we assume this conjecture to be correct, then the following can be used to speed up the DP process. The main idea is to reduce the search range of optimal reward c for each state: suppose $\mathbf{Price}(a, t)$ and $\mathbf{Price}(c, t)$ are already known, then for any $a < b < c$, $\mathbf{Price}(b, t)$ lies in range $[\mathbf{Price}(a, t), \mathbf{Price}(c, t)]$. For time interval t, we first search for the optimal reward for state $(\frac{N}{2}, t)$, then states $(\frac{N}{4}, t)$ and $(\frac{3N}{4}, t)$, then states $(\frac{kN}{8}, t)$ for $k = 1, 3, 5, 7$. This process continues until the optimal reward for every state has been found. Thus, the optimal reward searching process can be represented using a binary tree, where each node represents the optimal reward search range of certain state, and the search range of optimal reward is bounded by optimal reward already found in upper level nodes. Further, the search range of nodes in each level sum up to C, the pre-specified upper bound of task reward, while the number of levels is bounded by $O(\log n)$. Our technical report [17] has a diagram depicting these relationships. Therefore, the algorithm (which can be found in the extended technical report [17]) has a time complexity of $O(N_T N(N+C \log N))$.

Finally, although not improving time complexity, the monotonicity of task rewards $\mathbf{Price}(n, t)$ with respect to t for fixed n (i.e.,

when the number of remaining tasks are fixed, the rewards increase as we get closer to the deadline), can also be used to improve algorithm efficiency by reducing the optimal reward search range.

4. FIXED BUDGET PRICING STRATEGY

In this section we focus on another version of pricing problem: given a total monetary budget for all tasks, our objective is to minimize the expected total time when all tasks are completed.

Although, like in the previous section, we may still change the task reward dynamically, we will demonstrate that exercising this freedom does not help much in this scenario. In fact, we will prove that a static pricing strategy is nearly optimal.

4.1 Static Pricing Strategy

We first define what we mean to be a **Static Pricing Strategy**:

Definition 1. A *static pricing strategy* assigns a reward to each of the N tasks up-front (i.e., at the time the tasks are submitted to the marketplace), and then does not change this price subsequently. Note that the rewards need not be the same for all tasks.

Even though for a static pricing strategy tasks are submitted to the marketplace at the beginning with possibly different rewards, at any time, only the tasks with the highest reward will be picked up by workers. Thus, the rate at which tasks are picked up by workers will depend solely on the highest reward among all tasks (This property can be shown by *Utility Theory* in Section 2.2). Later on, when the tasks with the highest reward are exhausted, workers will start to pick up tasks with a lower reward; as a result the task acceptance rate will drop accordingly.

Note that static pricing strategies are a strict restriction of general dynamic pricing strategies. To see this, observe that for every static pricing strategy, there is an equivalent dynamic pricing strategy which changes the task reward for all tasks right after each task is completed. Therefore, the optimal static pricing strategy cannot have a lower total latency than the optimal dynamic pricing strategy. However we will show that in fact, the former can have as low expected total latency as the latter.

4.2 Optimality of Static Pricing Strategy

We now show that the optimal static pricing strategy has the minimum expected total latency for completing a given batch of tasks among all possible pricing strategies. Our main result will be Theorem 2, described in Section 4.2.1. Subsequent sections will focus on the proof and describe the algorithms.

4.2.1 Worker-Arrival Quantity

Recall that from Section 2, the workers arrive at the marketplace following a NHPP, and decide whether to work on our task following an independent Bernoulli process. Let T be the random variable denoting the total time elapsed before all tasks are completed, and W be the random variable denoting the total number of workers that have arrived at the marketplace before all the tasks are completed. Based on our model, the distribution of T conditioned on W depends only on the arrival-rate parameter $\lambda(t)$, and is independent of the pricing strategy. Suppose we use a pricing strategy S, then the expected value of T can be expressed as:

$$\mathbb{E}[T|S] = \int_W \mathbb{E}[T|W]\mathbf{Pr}(W|S)dW$$

Therefore our goal is to choose the optimal pricing strategy such that its induced distribution $\mathbf{Pr}(W|S)$ minimizes $\mathbb{E}[T|S]$.

Now if $E[T|W]$ is linear in W, then we have:

$$\mathbb{E}[T|S] = \int_W kW \mathbf{Pr}(W|S)dW = k\mathbb{E}[W|S]$$

which means that minimizing $\mathbb{E}[T|S]$ is equivalent to minimizing $\mathbb{E}[W|S]$. Minimizing the latter quantity is much more straightforward as we will show in next few sections. The justification for this linearity assumption is omitted due to space limitations, and can be found in the extended technical report [17].

The next theorem states that static pricing strategy is optimal in terms of minimizing the expected number of worker-arrivals $\mathbb{E}[W|S]$ and therefore expected latency $\mathbb{E}[T|S]$. We will prove the theorem in the next section.

THEOREM 2. *There exists a static pricing strategy S that minimizes the expected number of total worker-arrivals $\mathbb{E}[W|S]$, and therefore minimizes the expected total latency $\mathbb{E}[T|S]$ among all possible pricing strategies.*

4.2.2 Optimality of Static Pricing Strategy

The proof of Theorem 2 relies on another type of pricing strategy: *Semi-Static Pricing Strategy*. Semi-static pricing strategies serve as a bridge to connect static pricing strategies and dynamic pricing strategies in the proof of Theorem 2:

Definition 2. A *Semi-Static Pricing Strategy* generates a sequence of prices $c_1, c_2, \ldots, c_N$ at the time the tasks are posted to the marketplace. The strategy starts off by assigning c_1 to all tasks, and once one task is picked up by a worker, the price for all remaining tasks changes to c_2, and so on, until all the tasks are picked up by workers and completed. Unlike the static pricing strategy, the sequence of c_i's need not be monotonically decreasing.

We next show that the best dynamic pricing strategy is as good (i.e., has as low an expected completion time or latency) as the best semi-static pricing strategy.

THEOREM 3. *The optimal dynamic pricing strategy to minimize the expected number of worker-arrivals $\mathbb{E}[W]$ is in the form of a semi-static pricing strategy.*

The proof of this theorem can be found in the extended technical report [17]. Intuitively, the proof uses decision theory to demonstrate that, for a dynamic strategy, only the decisions made when a task gets completed matter — otherwise the state of the Markov process stays the same, and need not be changed.

The next theorem states that the effectiveness of any semi-static pricing strategy is not affected by the order of the c_i.

THEOREM 4. *For any semi-static pricing strategy S with price sequence $c_1, c_2, \ldots, c_N$, then the expected number of worker-arrivals $\mathbb{E}[W]$ is equal to $\sum_{i=1}^{N} \frac{1}{p(c_i)}$.*

The proof of this theorem can also be found in the extended technical report [17] as well. The proof sketch is as follows: We decompose W into sum of n independent random variables each representing the number of worker-arrivals between successive tasks. In doing so, we can rewrite the expected value of W as the sum of expected value of each of those n random variables. Finally the expected value of each individual random variable can be calculated directly.

Thus, by reordering the prices of a semi-static strategy (to ensure a descending order), we can change it into a static strategy with equal total expected completion time or latency. This result together with Theorem 3 demonstrates that the static pricing strategies are near-optimal.

4.3 Nearly Optimal Solution via LPs

In this section, we will address the problem of finding the optimal static pricing strategy. Suppose in the optimal static pricing strategy, the rewards for tasks are $c_1, c_2, \ldots, c_N$. Using Theorem 4, we know that the expected total number of worker-arrivals $\mathbb{E}[W]$ equals the sum of $\frac{1}{p(c_i)}$ (since any static pricing strategy is also a semi-static pricing strategy with the reward sequence monotonically non-increasing):

$$\mathbb{E}[W] = \sum_{i=1}^{N} \frac{1}{p(c_i)} \tag{8}$$

Let n_c be the number of tasks with reward c, i.e., $n_c = |\{i : c_i = c\}|$. Then, Equation (8) can be rewritten as: $\mathbb{E}[W] = \sum_c n_c \frac{1}{p(c)}$. The n_c values satisfy the following constraints:

$$\sum_c n_c = N; \ \ \sum_c n_c \times c \leq B; \ \ n_c \geq 0; \ \ n_c \in \mathbb{N} \tag{9}$$

where the first constraint is about the total number of tasks, the second constraint is about the total monetary budget (B denotes the total budget for all tasks).

Our objective is to find values of n_c that minimizes $\mathbb{E}[W]$ while simultaneously satisfying Constraints (9). For arbitrary functions $p(c)$, it is easy to show that the optimization problem is NP-HARD. Furthermore, we can show that the optimal static pricing strategy solution can be generated using a dynamic-programming based pseudo-polynomial time algorithm:

THEOREM 5. *The c_i for the optimal static pricing strategy can be discovered in* PTIME (B, N).

In short, the idea is to consider all optimal allocations of up to B to the first i tasks, for all $i \in 1 \ldots n$.

Our approach will instead be to approximately solve the optimization problem. We begin by casting the problem as an Integer Program (IP). Then, we will relax the IP to a Linear Program (LP) where the variables no longer have to be integers, i.e., the $n_c \in \mathbb{N}$ constraints are excluded. Lastly, we will round up variables in the solution to the LP to make them integers. The relaxed LP version of the problem is as follows:

Minimize $\sum_c n_c \frac{1}{p(c)}$ *s.t.* $\sum_c n_c = N; \sum_c n_c \times c \leq B; n_c \geq 0$

Instead of applying an LP solver and then performing rounding, next, we will describe an even faster approach, that leverages a special property of the LP above:

THEOREM 6. *There exists an optimal solution for the LP above which satisfies the following:*
- $\exists c_1 < c_2, \forall c \neq c_1, c \neq c_2, n_c = 0$
- $\forall c = tc_1 + (1-t)c_2, t \in \mathbb{R}: \frac{1}{p(c)} \geq t\frac{1}{p(c_1)} + (1-t)\frac{1}{p(c_2)}$

Theorem 6 can be intuitively explained using Figure 3. We first plot all the pairs $(c, \frac{1}{p(c)})$ in the plane. The first property of Theorem 6 states that there are at most two c_i's with non-zero n_{c_i}; i.e., there are at most two distinct prices c_1, c_2 that tasks are set at. Then, the second property of Theorem 6 states that for c_1 and c_2, there is no other point $(c, \frac{1}{p(c)})$ below the straight line connecting $(c_1, \frac{1}{p(c_1)})$ and $(c_2, \frac{1}{p(c_2)})$. In other words, $(c_1, \frac{1}{p(c_1)})$ and $(c_2, \frac{1}{p(c_2)})$ can only be segments on the *convex hull* of points $(c_i, \frac{1}{p(c_i)})$.

The proof of Theorem 6 can be found in the extended technical report [17]. The key idea is to show that given any optimal solution,
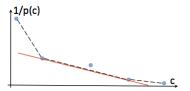
Figure 3: Illustration of Theorem 6, which implies that c_1 and c_2 can only be on the convex hull

it can be transformed to satisfy the first property while maintaining its optimality. The second property can be derived from the first property and Karush-Kuhn-Tucker conditions [10] of the LP.

Using Theorem 6, we can derive an algorithm (Algorithm 1) to find a nearly optimal pricing strategy. The algorithm generates the convex hull using all possible prices, and then picks the two most suitable prices to assign to tasks.

Theorem 7 provides an upper bound of the difference between rounded-LP solution (i.e., the solution provided by Algorithm 1) and optimal solution of original IP problem.

Algorithm 1 Find Optimal Static Pricing Strategy

---

function FINDOPTIMALSTATICSTRATEGY
    for $c = 0$ to C **do**
        Calculate the value of task acceptance probability $p(c)$.
    end for
    $CH \leftarrow$ **Convex hull of points** $(c, \frac{1}{p(c)})$
    $c_1 \leftarrow \max\{c \in CH : c \le \frac{B}{N}\}$
    $c_2 \leftarrow \min\{c \in CH : c > \frac{B}{N}\}$
    $n_1 \leftarrow \lceil \frac{c_2 N - B}{c_2 - c_1} \rceil, n_2 \leftarrow N - n_1$
    return n_1 tasks priced at reward c_1; n_2 tasks at reward c_2.
end function

THEOREM 7. *Let $\{n^*\}$ denote the optimal solution that minimizes $\mathbb{E}[W]$ under the Constraint (9), and $\{\hat{n}\}$ denote the rounded-LP solution from Algorithm 1, then the expected total latency difference between two solutions is bounded by:*

$$\sum_c \hat{n}_c \frac{1}{p(c)} \le \sum_c n_c^* \frac{1}{p(c)} + \left(\frac{1}{p(c_1)} - \frac{1}{p(c_2)}\right)$$

5. EXPERIMENTS

The goals of our experimental evaluation are two-fold: (a) to validate the pricing model assumptions we made in the previous sections, and (b) to compare our techniques versus others on simulations based on real crowdsourcing marketplace data, as well as real experiments deployed on a crowdsourcing marketplace. In Section 5.1, we examine the validity of the task acceptance probability equation (Equation (2)) and estimate the typical task acceptance probability values for real tasks. In Section 5.2, we examine the effectiveness (in terms of total monetary cost) of our techniques for the fixed deadline problem from Section 3 as compared to other schemes under simulations based on real workloads from Amazon's Mechanical Turk Marketplace via the mturk-tracker website [4]. We also study the sensitivity of our techniques with respect to (a) the algorithm parameters, (b) the estimation error of arrival-rate, and (c) the task acceptance probability mapping function, since many of these parameters may only be estimated approximately. In Section 5.3, we deploy our pricing technique for the fixed deadline problem from Section 3 on Mechanical Turk and report effectiveness in practice. (In the extended technical report, we present some data analysis of the data collected as a result.) In the extended technical report, we additionally examine the completion times of our techniques for the fixed budget problem from Section 4 under simulations based on real workloads.

5.1 Task Acceptance Probability

In Section 2.2, we used Equation (2) to map task rewards to task acceptance probabilities. In this section, we experimentally validate Equation (2) using utility theory (Section 5.1.1) and estimate the parameters in Equation (3) for real tasks (Section 5.1.2).

5.1.1 Utitity-based Simulation

As described in Section 2.2, workers choose tasks to work on by maximizing their gain in utility. In this section we simulate a specific workers' choice based on utility theory to justify the form of Equation (2).

The experiment settings are the following:

- The total number of tasks on the marketplace is set to be 100.
- The worker's utility estimate U_i for task $T_i (i > 1)$ follows a normal distribution $\mathcal{N}(\mu_i, \sigma_i^2)$, where μ_i are sampled independently from the normal distribution $\mathcal{N}(0, 1)$, and σ_i are sampled independently from the uniform distribution $U[0, 1]$.
- The worker's utility estimate U_1 for our target task T_1 follows a normal distribution $\mathcal{N}(\mu_1 = \frac{c}{50} - 1, \sigma_1^2)$ where c denotes the task reward of our task T_1 and σ_1 is sampled from the uniform distribution $U[0, 1]$.

For a given c (i.e., the reward for our task), we repeatedly sample the utility estimates for each of the 100 tasks as described above, and assume that the worker will choose our task if and only if our task has the highest utility among all the tasks in the marketplace. This sampling process gives us an estimate of the task acceptance probability p for a fixed reward c. We then repeat this process for different values of c, and plot the simulated acceptance probability p over different values of c in Figure 4. In the figure, we also depict the corresponding regression curve based on Equation (2) for comparison (the value of β is learned by fitting the simulated task acceptance probability value). As can be seen in Figure 4, the simulated acceptance probability p is well predicted by Equation (2). This justifies the model assumption that p is proportional to the exponential of the task utility U_i.
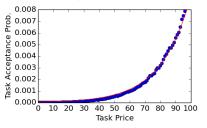

Figure 4: Simulated task acceptance probability p with reward c ranging from 0 to 100. Blue dots are simulation results and red curve is the regression function based on Equation (2) with $z_i = \mu_i$ and $\beta = 2.6$.

5.1.2 Real World Data

In Section 2.2, we use Equation (3) as parametric form of task acceptance probability function. In this section, we aim to estimate the typical values of parameters s, b, M in Equation (3) for tasks on a real marketplace.

We retrieved the snapshots of Amazon Mechanical Turk [3] from mturk-tracker [4]. The snapshots of the marketplace are taken every 20 minutes; we estimate the number of tasks that are completed every 20 minutes by subtracting the number of remaining tasks in each task group (note that in Mechanical Turk a task is called a HIT and a group or batch of tasks is called a HIT group). If the number of remaining tasks increased during the 20 minute window(i.e., the task requester added new tasks to this task group), we simply assume no tasks was completed during that 20 minutes.

We sampled 100 task groups that had at least 50 tasks completed (we enforced this threshold to filter out spam tasks) from

1/1/2014—1/28/2014, and for each task group we manually estimated the approximate average time usage for completing one task. Figure 5 shows for the two most popular task types, the *wage per second* versus average completed *workload per hour*, defined as:

$$\textbf{workload per hour} = \textbf{average \# of completed tasks per hour}$$
$$\times \textbf{ average time usage of each task}$$

We use these two values as axes because we want to make sure that our figure is invariant under task bundling. (In Mechanical Turk, requesters often group several tasks into one larger task.)
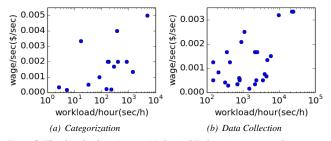

(a) Categorization   *(b) Data Collection*

Figure 5: The plot of tasks in Amazon Mechanical Turk, x axis represents the wage per second($/sec), y axis represents the average completed workload per hour(sec/h)

In order to estimate the value of parameters s, b, M in Equation (3), we assume that the utility of each task equals the logarithm of *workload per hour*, as implied by Equation (2) if we assume the sum of the exponential of the utilities of all tasks is a constant. We further assume that the utility of each task is linearly correlated with the *wage per second* attribute:

$$\log \textbf{workload/hour} = \textbf{utility} = \alpha \times \textbf{wage/sec} + b + \epsilon$$

where b is task-type bias term and ϵ accounts for all other factors affecting utility. We then apply Least Square Regression to estimate the linear coefficient α and the bias term b for each task type.

Table 2 shows the result of Least Square Regression. The two linear coefficients are approximately the same, implying that the linear coefficient of the *wage per second* attribute is the same for all task types. The bias term of Data Collection tasks is significantly higher than Categorization tasks, implying that workers in Mechanical Turk prefer Data Collection tasks to Categorization tasks.

	Linear coefficient	Bias
Categorization	748	3.66
Data Collection	809	6.28

Table 2: Linear coefficients and bias terms generated using Least Square Regression

Using the results in Table 2, we can then estimate the parameters s, b, M in Equation (3). Say our task is a Data Collection task and the average completion time of our task is 120 seconds, then based on Table 2, we have (task reward c is in cents):

$$\textbf{workload per hr.} = \exp\{809 \times \frac{c}{100} \times \frac{1}{120} + 6.28\} = \textbf{total} \times p(c) \times 120$$

where **total** denotes the total number of tasks completed per hour in the crowdsourcing marketplace (including all other tasks). In Mechanical Turk we have $\textbf{total} \approx 6000$ (as seen in mturk-tracker data). Using this fact we derive the following expression for $p(c)$:

$$p(c) \approx \frac{\exp\{\frac{c}{15} + 0.39\}}{\exp\{\frac{c}{15} + 0.39\} + 2000} \qquad (10)$$

5.2 Fixed Deadline Pricing Simulation

In this section, we examine the effectiveness of the dynamic pricing strategy in Section 3. We compare our dynamic pricing strategy against the binary-search-based fixed pricing strategy in Faridani's work [15]. We first compare the two pricing strategies under a realistic crowdsourcing workload in Section 5.2.1. We then study the

trend of relative reduction in cost of our dynamic pricing strategy compared to Faridani's fixed pricing strategy under different problem settings in Section 5.2.2. We further examine the sensitivity of dynamic pricing strategy to the granularity of time discretization in Section 5.2.3. The sensitivity of both pricing strategies to the estimations of task acceptance probability function and future arrival-rates are examined in Section 5.2.4 and Section 5.2.5 respectively.

In the following experiments, we assume the following default settings unless explicitly stated:

- The total number of tasks $N = 200$.
- The total time before deadline $T = 24$ hours.
- We retrieved the number of tasks completed during every 20 minutes interval for the time period from 1/1/2014 to 1/28/2014 from mturk-tracker as described in Section 5.1.2. The worker arrival rate $\lambda(t)$ is set to be piecewise constant on every such 20 minute time interval, i.e., for each time interval, $\lambda(t)$ is set to match the retrieved arrival data.
- Our target task is assumed to be a Data Collection task with an average completion time of 2 minutes. The mapping function between task reward c and task acceptance probability p can be derived as in Equation 10 in Section 5.1.2.
- The dynamic pricing model is trained using a time interval of length 20-minutes.

5.2.1 Effectiveness under a Realistic Workload

In this section we examine the effectiveness of our dynamic pricing strategy under a realistic workload derived from mturk-tracker.

We compare our dynamic pricing strategy with fixed price strategies which assigns a fixed reward to all tasks in advance, determined using binary search, and does not change the reward afterwards. Figure 6(a) shows the results of our experiment: for various values of the average reward (y axis), we plotted the expected number of tasks that remain unsolved at the deadline (x axis)—the total reward can be estimated by multiplying the average reward with the number of tasks. The **Penalty** parameter (see Section 3.1) is set in our dynamic pricing strategy such that the expected number of remaining tasks matches those of the fixed pricing strategy.
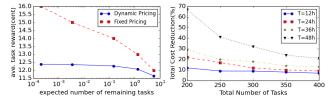

Figure 6: (a) Simulated average task reward c of our dynamic pricing strategy with respect to different threshold for the expected number of remaining tasks after deadline. (b) Percentage Cost Reduction with various settings of N and T

From Figure 6(a), we see that with low expected number of remaining tasks after deadline (less than 1 remaining tasks on average), the dynamic pricing strategy achieves an average task reward between 12 and 12.5. In fact, we can show that this average reward is very close to the theoretical lower bound of average task reward c_0 for any pricing strategy, which satisfies the following equation: $p(c_0) = \frac{N}{\int_0^T \lambda(t)dt}$. In our experiment, $c_0 \approx 12$.

The task reward c_0 has the following intuitive meaning: Suppose that we have an infinite number of tasks that can be picked up by workers. Let X denotes the number of tasks completed before deadline. Then c_0 is the minimum task reward such that: $\mathbb{E}[X] \geq N$. However, in practice we want to complete all N tasks before deadline, which is equivalent to: $\textbf{Pr}(X \geq N) \approx 1$. Note

that any pricing strategy satisfying the second constraint will automatically satisfy the first constraint. Therefore, in order to achieve a high probability guarantee, the average task reward will necessarily be higher than c_0 (since c_0 is the minimal possible task reward to satisfy the first constraint of $\mathbb{E}[X] \geq N$).

Figure 6(a) shows that our dynamic pricing strategy can finish all tasks by the deadline with very high probability (99.9%) and only 3% overhead (as compared to c_0). On the other hand, for the fixed reward pricing strategy from [15], the task reward needs to be set at 16 to achieve the same guarantee, resulting in a 33% increase over our dynamic pricing strategy, a significant difference in cost.

5.2.2 Trends of Effectiveness

In this section, we examine the relative gain of the dynamic pricing strategy compared to a fixed pricing strategy under various settings. We compute the cost reduction achieved by using the dynamic pricing strategy instead of the fixed pricing strategy (in percentage), and study how the reduction changes when the parameters are varied. The experiment settings are listed below:

- We study the relative effectiveness of dynamic pricing strategy with respect to the above five parameters of the experiment: N, T, and three parameters s, b, M in Equation (3).
- Each time we vary only one experiment parameter (N, T, s, b, M) while keeping other parameters fixed. The default value of these experiment parameters are: $N = 200$, $T = 24$ hours, $s = 15$, $b = -0.39$, $M = 2000$ (same as before).
- We will compute the total cost of all tasks using both pricing strategies. Let c_d and c_f be the total cost of the dynamic pricing strategy and the fixed pricing strategy respectively, the percentage cost reduction r is defined as: $r = \frac{c_f - c_d}{c_f}$. The percentage cost reduction r serves as a measure of the effectiveness of the dynamic pricing strategy as compared to the fixed pricing one.
- For both the dynamic pricing strategy and the fixed pricing strategy, the task reward is chosen such that all tasks are finished by the deadline with 99.9% confidence. This will be the default setting for the following experiments.

Figure 6(b) shows the percentage cost reduction under various settings of N and T. The experiment shows that the percentage cost reduction decreases as N increases and increases as T increases. Therefore, if we have less number of tasks and time before deadline is longer, then the gain of the dynamic pricing strategy is higher. On the other hand, the gain of the dynamic pricing strategy is lower if we want to complete more tasks in a shorter period of time. The intuitive explanation for this behavior is that with a longer period of time we have the ability to plan ahead and vary the price over time to get additional monetary cost savings.

Figure 7(a)–(c) shows the trend of percentage cost reduction when the parameter value of s, b, M changes. The implications can be summarized as follows:

- The gain of the dynamic pricing strategy is stable no matter how much the task acceptance probability p is sensitive to task reward c (Figure 7(a));
- The gain is lower if the task content is intrinsically more attractive compared to other tasks (Figure 7(b));
- The gain is higher if there are less tasks in the crowdsourcing marketplace than average (Figure 7(c)).

5.2.3 Granularity of Time Interval

In this section, we examine the effects of different time interval granularities. We train our dynamic pricing strategy using different time interval lengths, and examine the corresponding trade-off between effectiveness of pricing strategy and training time. The
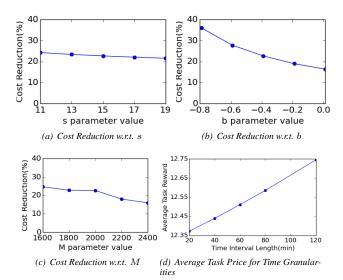

(a) Cost Reduction w.r.t. s

(b) Cost Reduction w.r.t. b

(c) Cost Reduction w.r.t. M

(d) Average Task Price for Time Granularities

Figure 7: $(a - c)$ Percentage Cost Reduction on varying s, b, M (d) Task price variation with Granularity

length of time interval used for training the dynamic pricing strategy ranges from 20 minutes to 2 hours.

Intuitively, the average task price should increase as the length of time interval increases since the strategy space is reduced; our experiment results in Figure 7(d) depict the expected behavior: The average task price increases steadily (but not by too much) as the length of time interval increases. On the other hand, the algorithm running time is rather stable and is not affected by the length of time interval (the algorithm running time is between 4 seconds and 5 seconds for all experiments, by executing Python code on a laptop with an Intel i7 processor). The stable behavior of running time is probably because of the Poisson truncation technique in Section 3.2: the expected number of workers arriving into the marketplace during each time interval will decrease as the length of time interval decreases, and the corresponding Poisson truncation threshold will also decrease. These results argue for using as small a time interval for which we can reliably obtain $\lambda(t)$ data.

5.2.4 Sensitivity of Parameter Estimation

Our dynamic pricing strategy (as well as Faridani's fixed pricing strategy [15]) requires estimation about the task acceptance probability mapping function $p(c)$ as input. However, these estimates may sometimes not be perfectly accurate. In this section, we examine the sensitivity of our pricing strategy to the estimation accuracy.

We train our dynamic pricing strategy under an inaccurate estimate of $p(c)$, and test it using the real value of $p(c)$. The task acceptance probability function is as Equation 10. For each experiment, we vary one parameter of the real $p(c)$ to examine the robustness of our dynamic pricing strategy. The estimation of other parameters are assumed to be accurate.

Figure 8 shows the average number of remaining tasks (left figures) and average task reward (right figures) for our dynamic pricing strategy with respect to different values of parameters s, b, M for the real $p(c)$. We focus first on the left figures, indicating the average number of remaining tasks. The data for the fixed pricing strategy (for various values of fixed price — 12 . . . 16) is also added for comparison. Here, unlike the fixed price strategies that all have non-zero remaining tasks, the dynamic pricing strategy curve is not visible because the number of remaining tasks is very close to zero. *Thus, we see that our dynamic pricing strategy is much more robust under inaccurate parameter estimation compared to fixed pricing strategy: it returns 0 remaining tasks with very high*
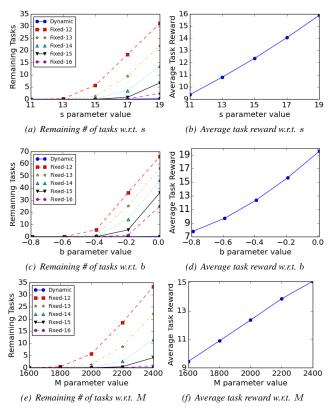
(a) Remaining # of tasks w.r.t. s

(b) Average task reward w.r.t. s

(c) Remaining # of tasks w.r.t. b

(d) Average task reward w.r.t. b

(e) Remaining # of tasks w.r.t. M

(f) Average task reward w.r.t. M

Figure 8: Simulated average number of remaining tasks under inaccurate parameter estimation of $p(c)$ for dynamic pricing strategy and fixed pricing strategy(left) and average task reward for dynamic pricing strategy(right)

probability, while the fixed pricing strategy completely fails to finish all the tasks on time. The right figures (depicting only the dynamic pricing strategy) show how the dynamic pricing stays robust: as the parameters are increased, even though the dynamic pricing strategy has been learned on incorrect parameters, it automatically increases the task reward as necessary.

5.2.5 *Sensitivity of Arrival-Rate Prediction*

Our dynamic pricing strategy and the fixed pricing strategy [15] both require the prediction of future worker arrival rate. There will be some inevitable discrepancy between the predicted and actual arrival-rates because of the intrinsic variations in arrival-rate. In this section, we examine the stability of our dynamic pricing strategy against such discrepancies.

We divide the historical arrival-rate data retrieved from mturk-tracker into two separate parts: one for training and the other for testing. We train our pricing strategy using the training arrival-rate data, but apply it on the test arrival-rate data. This way, we allow the algorithms to predict the general trend of the arrival-rate; however the algorithms will not be aware of the actual arrival-rate. We test our pricing strategies on 4 different days in year 2014: 1/1, 1/8, 1/15, 1/22. The training arrival-rate is the average arrival-rate of the other 3 days.

Figure 9(a) and 9(b) show the experiment results of the average number of remaining tasks and the average task reward respectively. As can be seen in the figures, both pricing strategies are relatively stable except for 1/1.

The surprising results for 1/1 can be explained by comparing Figure 9(c) and 9(d). Figure 9(d) shows the training arrival-rate and testing arrival-rate for 1/22: the training data is mostly in accordance with testing data, except that there are several random spikes in the testing data. The experiment results show that both pricing

(a) Average remaining # of tasks for different testing days

(b) Average task reward for different testing days

(c) The actual arrival rate and training arrival rate for 1/1/2014

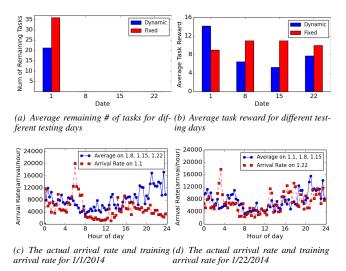(d) The actual arrival rate and training arrival rate for 1/22/2014

Figure 9: Testing Sensitivity of Arrival Rates

strategies are relatively stable to this kind of prediction error. On the other hand, Figure 9(c) depicts a consistent deviation between training data and testing data on 1/1, in which case both pricing strategies performs poorly. Such a consistent deviation is probably due to the special date of 1/1, so these deviations shouldn't occur very frequently in practice. Naturally, the prediction of arrival-rate on special days is hard to do because it does not follow a normal weekday pattern. As a result, adaptive prediction techniques such as predicting the arrival-rate in next few hours based on arrival-rate in last few hours could be useful in such cases. We leave exploration of such adaptive schemes for future work.

5.3   Live Experiments on Mechanical Turk

In this section, we conduct experiments on Mechanical Turk to examine the effectiveness of our dynamic pricing strategy from Section 3 in practice. In Section 5.3.1 we first deploy the fixed pricing strategy on Mechanical Turk to collect data about worker arrival rate $\lambda(t)$ and task acceptance probability function $p(c)$. In Section 5.3.2 we deploy our dynamic pricing strategy based on collected data, and experiment results are compared to the fixed pricing strategies. Common experiment settings are listed below:

- We use an entity resolution task dataset from Joglekar et al. [22]. Each task in the dataset consists of two photos (each with one athlete), and the worker is asked whether they contain the same person. In all experiments, we have 5,000 pairs of photos that we want workers to label.
- In all experiments, we post tasks on weekdays at 8 a.m. PST, with the deadline as 14 hours after start time (i.e., 10 p.m. PST).
- The worker qualifications are: worker must have at least 90% approval rate in history and live in United States.

Note that in Mechanical Turk, HITs (i.e., the unit of work in Mechanical Turk) with different price are grouped differently, even if they are issued by the same requester. Thus, workers looking for our specific HITs may not be able to know how many there are in total. So, in our experiments, we considered two options to vary price: (a) per HIT, keep the base price and number of tasks fixed, and vary the amount of bonus provided to the worker, or (b) per HIT, keep the base price fixed, and vary the number of tasks. We decided to go with the second option. In our experiments, the price of each task group (i.e., HIT in Mechanical Turk) is fixed at $0.02, and the price difference is expressed by the number of tasks (i.e., number of photo pairs to be labeled) in each HIT.

5.3.1 Fixed Pricing Experiment

The fixed pricing experiment consists of five trials, where each HIT contains 10/20/30/40/50 tasks respectively. Given the total number of tasks in the trials is fixed at 5,000, the actual number of HITs posted to the marketplace is 500/250/167/125/100 respectively. In other words, in the five trials, the price for each task is implicitly \$0.002/0.001/0.00066/0.0005/0.0004 respectively. We stopped at 50 tasks per HIT, to limit worker fatigue. Figure 10(a) shows the number of HITs completed during the whole time period.

As can be seen from the figure, the HIT completion rate is positively correlated with the price of each task in general: for instance, when the elapsed time is 6 hours, the trial with 10 tasks per HIT has more than double the number of HITs completed than that with 20 tasks, and more than four times the number of HITs completed than that with 30, 40 or 50 tasks. When the number of tasks in each HIT is below 20, the task completion rate becomes high enough to have all tasks completed before the deadline (i.e., 14 hours). The task completion rates are very close for trials with grouping size 30/40/50, which can be explained by the small difference between unit task prices (\$0.00066 / \$0.0005 / \$0.0004).

However, the actual work completion rates (in terms of percentage of total work completed) are quite different when we take the difference of number of tasks in each HIT into account, as shown in Figure 10(b). Perhaps somewhat surprisingly, we see that after multiplying the number of tasks in each HIT to the quantities in Figure 10(a), the curve of the trial with grouping size 50 becomes significantly higher than the curves of the trials with grouping size 30 and 40. This phenomenon suggest that grouping size per HIT has considerable effect on work completion rate: while workers choose HITs based on unit time wage, grouping more tasks into single HIT tends to force workers to stay on the same HIT for a longer time. (Note that in Mechanical Turk, workers do not earn any reward until they have completed all tasks in a HIT.)

5.3.2 Dynamic Pricing Experiment

To make experiment results comparable, the basic settings settings of the dynamic pricing experiment are the same as Section 5.3.1 (i.e., start time, deadline, total number of tasks), except that the grouping size is changed every hour based on our dynamic pricing strategy. The grouping size is chosen from 10/20/30/40/50, and the corresponding HIT acceptance rates are estimated from the fixed pricing experiment in the previous section. The worker arrival rates are estimated by averaging normalized worker arrival data in the five fixed pricing trials.

Figure 10(c) depicts work completion rate of the five trials (one on each day) in our experiment. As can be seen in the figure, the dynamic pricing strategy ends up completing all the tasks well before the deadline (6 hours instead of 14 hours). Furthermore, we find that the average total cost for the five trials is \$3.2, which is much less ($\approx 36\%$ less) than the total cost of \$5 for the fixed pricing strategy with grouping size 20; in fact, note that the fixed pricing strategy with grouping size 20 had an elapsed time of 8, two hours more than the elapsed time of any of the trials for our strategies.

In our extended technical report [17], we analyze the data collected via both experiments to shed light on two aspects of worker behavior: the accuracy of submitted answers and the number of tasks completed by each worker under different pricing settings. Our experimental results suggest that the quality of answers are reasonably good (with average accuracy > 90%) regardless of task price. While we refer the reader to the technical report for the details, Table 3 and 4 depict the average accuracy for different group sizes for the fixed pricing experiment, and the average accuracy for different trials in the dynamic pricing experiment. Overall, there

doesn't seem to be a distinct pattern governing how pricing affects accuracy, and we suspect the differences are all due to random variations. Our results also show that the average number of tasks completed by each worker growing approximately exponentially with respect to task price. However, note that these results are preliminary, and that comprehensive experiments are necessary to draw further conclusions.

Group Size	10	20	30	40	50
Average Accuracy	92.7	90.4	91.6	90.0	89.5

Table 3: The average accuracy of answers in the fixed pricing experiment

Trial	1	2	3	4	5
Overall Ave. Accuracy	90.7	91.7	88.2	95.0	90.9

Table 4: The average accuracy of answers in dynamic pricing experiment

6. RELATED WORK

The prior work related to ours can be placed in a few categories; we describe each of them in turn:

Pricing Schemes: Faridani [15] develop models for marketplace dynamics that we leverage in this paper. They also develop static pricing strategies that we compare against. To the best of our knowledge, there has been no work on optimizing price apart from [15].

Control Theory: Recent work has leveraged decision theory for improving cost and quality in simple crowdsourcing workflows, typically using POMDPs (Partially Observable MDPs): Dan Weld's group has designed strategies to dynamically choose the best decision to make at any step in the workflow (refine, improve, vote, or stop), and also to dynamically switch between workflows to improve the overall "utility" [11, 12, 26, 27]. Kamar et al. [23] use POMDPs to study how to best utilize participation in voluntary crowdsourcing systems, specifically, Galaxy Zoo, an astronomical data set verified by human workers. The papers mentioned above do not provide theoretical guarantees. Our prior work also uses decision theory for getting guarantees on cost and accuracy for filtering [35, 36]. None of these prior papers study the problem of determining optimal pricing for tasks over time: all of them assume that each task has a fixed price or reward, and optimize the set of tasks to meet accuracy guarantees.

Crowd Algorithms: There has been a lot of recent activity centered around designing data processing algorithms where the unit operations are performed by human workers, such as filtering [36], sorting and joins [20, 30], top-k [14], deduplication and clustering [19, 47] and categorization [39]. None of these papers explore the problem of pricing tasks to complete on time.

Of these papers, just categorization [39] and filtering [35, 36] consider the aspect of latency, and there too, they use number of round-trips as a proxy for latency rather than the true elapsed time.

Error Estimation: There has been significant work on simultaneous estimation of answers to tasks and errors of workers using the EM algorithm or other local optimization techniques. There have been a number of papers studying increasingly expressive models for this problem, including difficulty of tasks and worker expertise [28, 44, 49], adversarial behavior [42], and online evaluation of workers [29, 41, 48]. There has also been work on choosing workers for evaluating different items so as to reduce overall error rate [34, 45]. Recent work has also tried to obtain theoretical guarantees for both worker error estimates as well as correct labels for items [13, 18, 21, 24]. Our work on pricing tasks is orthogonal to this line of work, and can be combined with any of these schemes to better price a batch of tasks to complete by a given deadline.

Applications: There are a number of useful applications of crowdsourcing, such as sentiment analysis [40], identifying spam [33], determining search relevance [8], and translation [50].
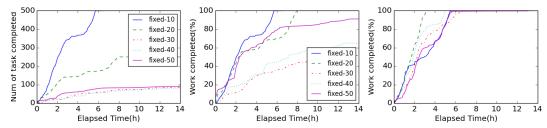
Figure 10: Experiments on Mechanical Turk (a) The HIT completion rate for fixed pricing strategy (b) The percentage work completion rate for fixed pricing strategy (c) The percentage work completion rate for dynamic pricing strategy

7. CONCLUSIONS

In this paper, we developed algorithms to optimally set and vary the price for human computation tasks in a crowdsourcing marketplace to meet latency and cost constraints. For a monetary budget scenario, we demonstrated that static pricing strategies lead to optimal completion times, and developed efficient algorithms to find near-optimal pricing strategies. For a fixed deadline scenario, we demonstrated that our techniques, based on MDPs, outperform fixed pricing strategies by up to 30% on simulations based on real-world crowdsourcing marketplace data and live experiments, and are more robust to errors in estimates of marketplace parameters and predictions of future trends. Our techniques can be profitably employed in scenarios demanding the repeated use of crowdsourcing on a large scale, wherein the cost savings will be massive.

8. REFERENCES

[1] Crowdsourced data analysis with Clockwork Raven. *https://blog.twitter.com/2012/crowdsourced-data-analysis-with-clockwork-raven*.

[2] Crowdsourcing Insights from eBay (Retrieved 10 January 2014). *http://crowdopolis.info/james-rubinstein.pdf*.

[3] Mechanical Turk (Retrieved 22 July 2013). *http://www.mturk.com*.

[4] Mechanical Turk Tracker (Retrieved 20 February 2014). *http://mturk-tracker.com*.

[5] Microsoft Wants to Turn Crowdsourcing from an Art to a Science (Retrieved 10 January 2014). *http://www.crowdsourcing.org/editorial/microsoft-wants-to-turn-crowdsourcing-from-an-art-into-a-science/21233*.

[6] Samasource Annual Report (Retrieved 10 January 2014). *http://www.slideshare.net/leila_c/samasource-2011-annual-report*.

[7] Translators Wanted at LinkedIn. The Pay? 0 an Hour. The New York Times (Retrieved 10 January 2014). *http://www.nytimes.com/2009/06/29/technology/start-ups/29linkedin.html*.

[8] O. Alonso, D. E. Rose, and B. Stewart. Crowdsourcing for relevance evaluation. *SIGIR Forum*, 42(2):9–15, 2008.

[9] P. Bohannon, S. Merugu, C. Yu, V. Agarwal, P. DeRose, A. S. Iyer, A. Jain, V. Kakade, M. Muralidharan, R. Ramakrishnan, and W. Shen. Purple sox extraction management system. *SIGMOD Record*, 37(4):21–27, 2008.

[10] S. P. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[11] N. Bruno. Minimizing database repros using language grammars. In *EDBT*, pages 382–393, 2010.

[12] P. Dai, Mausam, and D. S. Weld. Decision-theoretic control of crowd-sourced workflows. In *AAAI*, 2010.

[13] N. Dalvi, A. Dasgupta, R. Kumar, and V. Rastogi. Aggregating crowdsourced binary ratings. In *WWW*, pages 285–294, 2013.

[14] S. B. Davidson, S. Khanna, T. Milo, and S. Roy. Using the crowd for top-k and group-by queries. ICDT '13, pages 225–236, 2013.

[15] S. Faridani, B. Hartmann, and P. G. Ipeirotis. What's the right price? pricing tasks for finishing on time. In *Proceedings of HCOMP11: The 3rd Workshop on Human Computation*, 2011.

[16] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. Crowddb: answering queries with crowdsourcing. In *SIGMOD*, 2011.

[17] Y. Gao and A. Parameswaran. Finish them!: Pricing Algorithms for Human Computation. Technical report, U. Illinois (UIUC), `http://arxiv.org/pdf/1408.6292.pdf`, 2014.

[18] A. Ghosh, S. Kale, and P. McAfee. Who moderates the moderators? crowdsourcing abuse detection in user-generated content. In *EC*, pages 167–176, 2011.

[19] R. Gomes, P. Welinder, A. Krause, and P. Perona. Crowdclustering. In *NIPS*, pages 558–566, 2011.

[20] S. Guo, A. Parameswaran, and H. Garcia-Molina. So who won?: dynamic max discovery with the crowd. In *SIGMOD Conference*, pages 385–396, 2012.

[21] M. Joglekar, H. Garcia-Molina, and A. Parameswaran. Evaluating the crowd with confidence. In *SIGKDD*, 2013.

[22] M. Joglekar, H. Garcia-Molina, and A. Parameswaran. Evaluating the Crowd with Confidence . In *KDD*, 2013.

[23] E. Kamar, S. Hacker, and E. Horvitz. Combining human and machine intelligence in large-scale crowdsourcing. In *AAMAS*, pages 467–474, 2012.

[24] D. Karger, S. Oh, and D. Shah. Effcient crowdsourcing for multi-class labeling. In *SIGMETRICS*, pages 81–92, 2013.

[25] D. R. Karger, S. Oh, and D. Shah. Budget-optimal task allocation for reliable crowdsourcing systems. *CoRR*, abs/1110.3564, 2011.

[26] C. H. Lin, Mausam, and D. S. Weld. Crowdsourcing control: Moving beyond multiple choice. In *UAI*, pages 491–500, 2012.

[27] C. H. Lin, Mausam, and D. S. Weld. Dynamically switching between synergistic workflows for crowdsourcing. In *AAAI*, 2012.

[28] Q. Liu, J. Peng, and A. Ihler. Variational inference for crowdsourcing. In *NIPS*, pages 701–709, 2012.

[29] X. Liu, M. Lu, B. C. Ooi, Y. Shen, S. Wu, and M. Zhang. Cdas: a crowdsourcing data analytics system. *Proc. VLDB Endow.*, 5(10):1040–1051, June 2012.

[30] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller. Human-powered sorts and joins. In *VLDB*, 2012.

[31] W. A. Massey, G. A. Parker, and W. Whitt. Estimating the parameters of a nonhomogeneous poisson process with linear rate. *Telecommunication Systems*, 5(2):361–388, 1996.

[32] D. McFadden. Conditional logit analysis of qualitative choice behavior. 1973.

[33] M. Motoyama, K. Levchenko, C. Kanich, D. McCoy, G. M. Voelker, and S. Savage. Re: Captchas-understanding captcha-solving services in an economic context. In *USENIX Security Symposium*, pages 435–462, 2010.

[34] P. Donmez et al. Efficiently learning the accuracy of labeling sources for selective sampling. In *KDD*, pages 259–268, 2009.

[35] A. Parameswaran, S. Boyd, H. Garcia-Molina, A. Gupta, N. Polyzotis, and J. Widom. Optimal Crowd-Powered Rating and Filtering Algorithms. Technical report, Stanford University, 2013.

[36] A. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom. CrowdScreen: algorithms for filtering data with humans. In *SIGMOD Conference*, pages 361–372, 2012.

[37] A. Parameswaran, H. Park, H. Garcia-Molina, N. Polyzotis, and J. Widom. Deco: declarative crowdsourcing. In *CIKM*, pages 1203–1212, 2012.

[38] A. Parameswaran and N. Polyzotis. Answering Queries using Humans, Algorithms and Databases. In *CIDR*, pages 160–166, 2011.

[39] A. Parameswaran, A. D. Sarma, H. Garcia-Molina, N. Polyzotis, and J. Widom. Human-assisted graph search: it's okay to ask questions. *PVLDB*, 4(5):267–278, 2011.

[40] R. Snow et al. Cheap and fast - but is it good? evaluating non-expert annotations for natural language tasks. In *EMNLP*, pages 254–263, 2008.

[41] A. Ramesh, A. Parameswaran, H. Garcia-Molina, and N. Polyzotis. Identifying reliable workers swiftly. Technical report, Stanford University, September 2012.

[42] V. C. Raykar and S. Yu. Eliminating spammers and ranking annotators for crowdsourced labeling tasks. *Journal of Machine Learning Research*, 13:491–518, 2012.

[43] S. M. Ross. *Stochastic processes*, volume 2. John Wiley New York, 1996.

[44] V. C. Raykar et al. Supervised learning from multiple experts: whom to trust when everyone lies a bit. In *ICML*, page 112, 2009.

[45] V. S. Sheng et al. Get another label? improving data quality and data mining using multiple, noisy labelers. In *KDD*, pages 614–622, 2008.

[46] P. Wais, S. Lingamneni, D. Cook, J. Fennell, B. Goldenberg, D. Lubarov, D. Marin, and H. Simons. Towards building a high-quality workforce with mechanical turk. In *Computational Social Science and the Wisdom of Crowds, NIPS Workshop*, 2010.

[47] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *PVLDB*, 5(11):1483–1494, 2012.

[48] P. Welinder and P. Perona. Online crowdsourcing: rating annotators and obtaining cost-effective labels. In *CVPR*, 2010.

[49] J. Whitehill, P. Ruvolo, T. Wu, J. Bergsma, and J. R. Movellan. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *NIPS*, pages 2035–2043, 2009.

[50] O. Zaidan and C. Callison-Burch. Feasibility of human-in-the-loop minimum error rate training. In *EMNLP*, pages 52–61, 2009.