Optimal Crowd-Powered Rating and Filtering Algorithms

Aditya Parameswaran
Stanford University & UIUC
adityagp@illinois.edu

Stephen Boyd
Stanford University
boyd@stanford.edu

Hector Garcia-Molina
Stanford University
hector@cs.stanford.edu

Ashish Gupta
Stanford University
ashgup@stanford.edu

Neoklis Polyzotis
UCSC & Google
alkis@cs.ucsc.edu

Jennifer Widom
Stanford University
widom@cs.stanford.edu

ABSTRACT

We focus on crowd-powered filtering, i.e., filtering a large set of items using humans. Filtering is one of the most commonly used building blocks in crowdsourcing applications and systems. While solutions for crowd-powered filtering exist, they make a range of implicit assumptions and restrictions, ultimately rendering them not powerful enough for real-world applications. We describe two approaches to discard these implicit assumptions and restrictions: one, that carefully generalizes prior work, leading to an optimal, but oftentimes intractable solution, and another, that provides a novel way of reasoning about filtering strategies, leading to a sometimes suboptimal, but efficiently computable solution (that is asymptotically close to optimal). We demonstrate that our techniques lead to significant reductions in error of up to 30% for fixed cost over prior work in a novel crowdsourcing application: peer evaluation in online courses.

1. INTRODUCTION

Crowdsourcing, i.e., using input from human workers to solve problems, has emerged as a powerful mechanism for processing data, especially unstructured data, such as images, videos, and text. Over the past few years, there has been a development of algorithms that use humans for a range of data processing tasks, including sorting [30], clustering [20], maximum [21], finding [40], filtering [36], and deduplication [46]. These algorithms typically offer guarantees on some combination of accuracy, latency, and monetary cost (i.e., total compensation to human workers).

Of these tasks, perhaps the most commonly used data processing task is filtering, i.e., using humans to evaluate or rate items such as images, videos, or text. For instance, filtering is used for content moderation [2], i.e., deciding if each image in a dataset is appropriate to be viewed by a general audience, by having multiple humans evaluate each image. In addition to content moderation, filtering is used in spam identification [32], relevance estimation [7], text classification [9,41], and video analysis [15]. Filtering also forms a fundamental relational operator in declarative crowdsourcing systems [19,38].

Our previous work on filtering, titled CrowdScreen [36], provides a state representation that enables us to reason about and store filtering strategies for a simple setting: filtering a set of *equally difficult* items *(with no prior knowledge on items)* using a *infinite pool* of *independent and identical* workers with *equal costs* (i.e., the workers need to be paid the same amount). Under those assumptions, the designed strategies have guarantees on expected cost and accuracy (and *no guarantees on latency*).

While CrowdScreen provides a good starting point for filtering, due to the many implicit assumptions and restrictions, it is ultimately *not powerful enough for real-world filtering applications*. In this paper, we describe techniques that enable us to remove many of the assumptions and restrictions in CrowdScreen. Overall, our techniques can inform the design of a general and powerful filtering operator that we believe can be applied to any real-world scenario.

Our approach is the following: we first recast the new variants of filtering (on removing simplifying assumptions and restrictions one by one) into generalizations of the representation in CrowdScreen. While this step in itself is not hard, our main challenge here is to ensure that we preserve the desirable linearity properties that allow us to use efficient optimization techniques.

However, an even bigger challenge arises while incorporating many of the new variations: the representation itself becomes intractable to store. For instance, when incorporating distinct worker abilities (instead of assuming that all workers have identical accuracies), the representation scales exponentially in the number of workers. Thus, the optimization problems becomes even more complex and time consuming. To combat this challenge for the troublesome variations, we devise a novel state representation that reduces the amount of information stored and enables us to find strategies that are approximate, but provably close to optimal asymptotically.

To describe both these representations (the recasting of the CrowdScreen representation, as well as the entirely new compressed representation), we focus on two important variations not considered by CrowdScreen: distinct worker abilities and prior information.

- *Distinct worker abilities:* CrowdScreen assumes that all human workers have the same error rates — all workers are equally capable of answering questions. This assumption is certainly not true in practice: there are some workers that are much better than others, possibly because they do a more careful job or are more competent.
- *Prior information:* The CrowdScreen algorithms assume that we have no information about any of the items to begin with; however, there may be cases where we have "prior information"—that is, knowledge that some items are more likely to pass the filter than others. This prior information may originate from an automated algorithm, such as a machine learning algorithm, that outputs probabilities for whether each item is likely to pass the filter or not. We therefore come up with a principled way to combine machine- and human-computation, so that we expense human cycles on the items that are hard to judge automatically.

The entire list of variations or aspects we consider can be found in Table 1 in Section 5 (we do not expect the reader to fully understand the table at this point), along with brief explanations.

Novel Application: To ground our discussion, we evaluate our techniques on a novel real-world application that, to our knowledge, has not been studied in the context of crowd algorithms, *peer evaluation in online courses*.

MOOCs (Massive Open Online Courses) [10] are revolutionizing education. There are hundreds of courses being offered by organizations such as Coursera [1], Udacity [6], and EdX [3], and each of these courses are being taken by thousands of students worldwide [4]. Evaluating students in many of these MOOCs (especially courses in the humanities) requires human expertise: for instance, it is impossible to grade an essay, a software project, or a mathematical proof completely automatically.

Given that thousands of students are taking these courses, the MOOC providers have turned to peer evaluation, i.e., having students evaluate each other's work, as the primary mechanism for grading in these problematic courses. Peer evaluation is a large scale application of crowd-powered filtering: for each student submission, the peer evaluation system needs to decide how many graders would need to evaluate that submission in order to correctly determine the true grade. Student graders may make mistakes while evaluating submissions, and therefore, we may need multiple student graders to evaluate each submission. Since student grader time is a limited resource, we would rather have graders evaluate student submissions for which there is more uncertainty regarding the true grade, instead of submissions for which the true grade is fairly certain.

We demonstrate that the techniques described in this paper are useful for peer evaluation: they provide significantly higher quality results than standard heuristics currently in use in the peer evaluation system in the MOOC platform Coursera [1], as well as algorithms in CrowdScreen, which ignore a number of key factors, and are therefore not as useful. Overall, we get reductions of upto 30% in error for fixed cost, a sizable improvement in performance.

Contributions: Here is the outline for the paper:

- We describe the *answer-record* representation, a way to represent and reason about filtering strategies. This representation is a straightforward extension of the representation in CrowdScreen. This representation provides the optimal solution to all the aspects we consider, but can be expensive to compute in some cases (Section 3).
  - We describe our solution for distinct worker abilities (Section 3.1).
  - We provide the key ideas for incorporating prior information (Section 3.2).
- We describe the *posterior-based* representation, a new way of representing and reasoning about strategies. This representation leads to an efficient but approximate solution to all the aspects we consider (Section 4).
  - We describe the representation for the basic setting considered in CrowdScreen, and show that the expected cost of the optimal strategy in this representation converges (on increasing a user-specified parameter without bound) to the cost of the optimal strategy in the answer-record representation (Section 4.1).
  - We describe our solution using this representation for distinct worker abilities (Section 4.2).
  - We provide the key ideas for using the posterior-based representation when incorporating prior information (Section 4.3).
- We discuss other aspects and demonstrate that our representations are general enough to incorporate all the new variations (Section 5).
- We evaluate our algorithms on real MOOC data, demonstrating that we get a significant reduction of up to 30% in error (for same cost) by using our techniques over techniques in CrowdScreen, as well as other techniques and heuristics currently used in the peer evaluation system (Section 6).
- We describe related work (Section 7), and conclude (Section 8).

2. PRELIMINARIES

We begin by describing the basic setup from CrowdScreen, but when taking into consideration distinct worker abilities. The other aspects mentioned in the introduction will be described later on.

We are given a set of items $\mathcal{I}$, where $|\mathcal{I}| = n$. A random variable V controls whether an input item satisfies the filter ($V = 1$) or not ($V = 0$). The selectivity of our filter, s, gives us the probability that $V = 1$ (over all possible items).

We assume that there is no automated mechanism to examine an item and determine for certain whether that item satisfies the filter or not. The only type of action we can perform on an item is to ask a specific human worker w_i, $i \in 1 \dots r$ a *question*. The worker can tell us YES (meaning that they think the item satisfies the filter) or NO. The worker w_i can make mistakes, and in particular:

- The false positive rate is: $Pr[w_i\text{'s answer is YES}|V = 0] = e_0(w_i)$
- The false negative rate is: $Pr[w_i\text{'s answer is NO}|V = 1] = e_1(w_i)$

The error rates $e_0(w_i), e_1(w_i)$ are estimated either by evaluating worker w_i on questions with known correct answers, or by using prior history on worker performance. In peer evaluation, error rates are estimated by having workers (in this case, graders) evaluate a few "test" submissions that the course instructors have also graded. The selectivity s is estimated by having course instructors evaluate a small sample of submissions.

Overall, there may be some workers who are less error-prone at answering questions than others, possibly because they are more diligent or more capable. We can ask different humans the same question to get better accuracy, and we assume that their errors are independent. (We relax this restriction in Section 5.) However, if we ask the same human the question on the same item, we will get the same answer. Therefore, we will ask the question on a given item to a given human at most once.

A *strategy* $\mathcal{F}$ is a computer procedure that takes as input one item, asks one or more humans questions on that item, and eventually outputs either Pass or Fail for that item. A Pass output represents a belief that the item satisfies the filter, while Fail represents the opposite. We define an *algorithm* to be a procedure that, given parameters and constraints, generates a strategy.

3. ANSWER-RECORD REPRESENTATION

In this section, we describe our first representation for worker abilities. We will then discuss how we may incorporate prior information. For the latter, we may have a machine learning algorithm (say a classifier) that analyzes items and assigns probabilities of passing the filter to each item. As an example, if we were doing content moderation of images, there are automated algorithms that analyze each image (perhaps by using the distribution of colors or by looking for specific patterns) and provide a probability for whether the image is likely to be inappropriate for a general audience.

3.1 Setting with Worker Abilities

Representation: Since humans may have different error rates, the state of processing for an item after some questions are asked can be completely represented using a state variable $S = (x_1, y_1, x_2, y_2, \dots, x_r, y_r)$, where x_i is an indicator variable indicating whether worker w_i answered YES for the question on the item, and y_i indicates if worker w_i answered NO for the question on the item. If $x_i = y_i = 0$, then w_i has not been asked a question yet on the item; if $x_i = 1, y_i = 0$, then w_i has been asked a question and has answered YES; and if

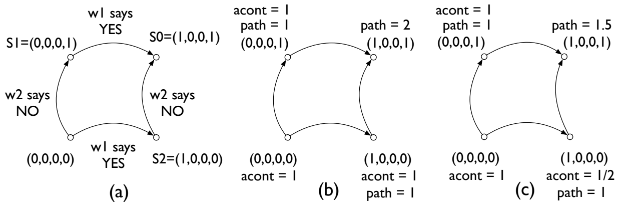

Figure 1: Path-based Reasoning

$x_i = 0, y_i = 1$, then w_i has been asked a question on the item and has answered NO. The case where $x_i = y_i = 1$ can never arise.

Note that the reason why the state variable S is a complete description of the state of processing is that the order in which the answers are provided by humans is not critical for making a decision on an item; only the set of YES/NO answers is important, and the identity of the workers who provided the answers. Therefore, the state variable S captures all the relevant information about an item that is necessary for a strategy to maintain. In CrowdScreen, S was simply the count of the YES and NO answers. This information was sufficient because all workers were assumed to be equally error-prone in that setting, and therefore, the identity of the worker who gave a specific answer was not important.

We therefore call this representation the *Answer Record Representation* (i.e., the state or processing is the complete set of answers).

At a given state S, a strategy $\mathcal{F}$ probabilistically does one of the following: (a) stop executing, and return Pass on the item, (b) stop executing, and return Fail on the item, or (c) continue executing (Cont), i.e., ask an additional question for that item. Notice that, we are overloading our use of the term "strategy" to mean two things: (a) the code that operates on an item, and eventually returns Pass or Fail, (b) a decision function, defined over all reachable states, that takes as input a state S, and outputs a decision for that state (Pass, Fail, Cont). The use will be clear from the context.

Thus, at a given state S, the strategy returns Pass, i.e., $\mathcal{F}(S) =$ Pass, with probability $a_{pass}(S)$, returns Fail, i.e., $\mathcal{F}(S) =$ Fail with probability $a_{fail}(S)$, and will ask another human a question on that item, $\mathcal{F}(S) =$ Cont with probability $a_{cont}(S) = 1 - a_{pass}(S) - a_{fail}(S)$. If either Pass or Fail returned at a state, we say that the strategy terminates. If Cont is returned at a state, then an answer is requested from one of the unasked human workers—those for whom $x_i = y_i = 0$, all with equal probability. Thus, in this scenario, we do not control which human worker answers our question—this scenario is relevant in marketplaces like Mechanical Turk, or even in peer evaluation, where we do not control who is online and available for grading at a given time. In some real world applications, we may be able to control which worker is asked to answer the question; this aspect is covered in Section 5.

Due to the properties described above, strategies are instances of the well-studied discrete Markov Decision Processes (MDPs) [42]. MDPs are represented by a set of states (here, all possible S), possible decisions for each state (here, Pass, Fail, Cont), and a probability distribution over next states when a decision is taken at a given state. MDPs have a single reward function (or metric) associated with each state. In our case, since we are considering multiple metrics—cost and accuracy—we cannot use the standard value and policy iteration techniques that are meant for optimizing single metrics in MDPs. Instead, we must rely on linear programming. We emphasize that our key contribution is not in the solution but in reducing our scenario to the MDP formalism, while maintaining computational efficiency. To enable our paper to be self-contained for a database audience, we describe our approach without using the MDP formalism.

Metrics: To determine which strategy is best, we study the metrics of error and cost. We start by defining two quantities, given a strategy:

- $p_1(S)$ is the probability that the strategy reaches state S and the item satisfies the filter ($V = 1$); and
- $p_0(S)$ is the probability that the strategy reaches state S and the item does not satisfy the filter ($V = 0$).

We can now define the following metrics:

- E is the sum of expected errors across all states. The expected error at a state S is simply the probability that the strategy terminated at S with an error being made.

$$E = \sum_S a_{pass}(S) \cdot p_0(S) + a_{fail}(S) \cdot p_1(S) \tag{1}$$

- C is the sum of expected cost across all states. The cost at a state S is the probability that the state was reached, i.e., $(p_0(S)+p_1(S))$, multiplied by the probability that a termination decision was taken, i.e., $(a_{pass}(S) + a_{fail}(S))$, multiplied by the cost, i.e., the total number of answers so far: $\sum_{i\in 1\ldots r}[x_i + y_i]$.

$$C = \sum_S [p_0(S) + p_1(S)] \cdot (a_{pass}(S) + a_{fail}(S)) \cdot (\sum_{i\in 1\ldots r} [x_i + y_i]) \tag{2}$$

The probabilities p_0 and p_1 can be iteratively computed; we illustrate this using a simple example with $r = 2$ workers for p_0 (also displayed in Figure 1(a)). Consider state $S_0 = (1, 0, 0, 1)$ (i.e., worker w_1 answered YES and w_2 answered NO). The figure shows that we can get to state S_0 from either $S_1 = (0, 0, 0, 1)$ by receiving a YES from w_1, or from $S_2 = (1, 0, 0, 0)$ by receiving a NO from w_2. Then, $p_0(S_0)$ is the sum of two quantities: the first quantity is the probability that the strategy reached S_0 via S_1, and the second quantity is the probability that the strategy reached S_0 via S_2, both when the item does not satisfy the filter. There are no other ways to get to S_0. The first quantity is equal to $p_0(S_1) \cdot a_{cont}(S_1) \cdot e_0(w_1)$, where

- $p_0(S_1)$ is the probability that the strategy reached S_1 and the item does not satisfy the filter ($V = 0$),
- $a_{cont}(S_1)$ is the probability that an additional human worker was asked (the only unasked worker at S_1 is w_1, so w_1 was asked to answer),
- and $e_0(w_1)$ is the probability that w_1 answered incorrectly.

The second quantity is similar. Thus,

$$p_0(S_0) = p_0(S_1)a_{cont}(S_1)e_0(w_1) + p_0(S_2)a_{cont}(S_2)(1 - e_0(w_2))$$

In the general case, we have the following equations:

$$\begin{aligned} p_0(x_1, y_1, \ldots, x_r, y_r) \quad &= \sum_{\substack{1\leq i\leq r;\ x_i=1; \\ R=(x_1,y_1,\ldots,x_{i-1},y_{i-1},0,0,\ldots,x_r,y_r)}} \frac{e_0(w_i) \cdot p_0(R) \cdot a_{cont}(R)}{b} \quad + \\ &\sum_{\substack{1\leq i\leq r;\ y_i=1; \\ R=(x_1,y_1,\ldots,x_{i-1},y_{i-1},0,0,\ldots,x_r,y_r)}} \frac{(1 - e_0(w_i)) \cdot p_0(R) \cdot a_{cont}(R)}{b} \cdot \\ p_0(0, 0, \ldots, 0, 0) \quad &= \quad (1 - s) \end{aligned}$$

where b is the number of x_i or y_i that are 0, i.e., the number of workers who have not been asked yet at state R. Thus, we simply sum up the probabilities that the strategy reaches $(x_1, y_1, \ldots, x_{i-1}, y_{i-1}, 0, 0, \ldots, x_n, y_n)$ and gets a YES from a specific worker w_i (out of b unasked workers) who has not been asked before, for all i. And we add this sum to the probabilities that the strategy reaches $(x_1, y_1, \ldots, x_{i-1}, y_{i-1}, 0, 0, \ldots, x_n, y_n)$ and gets a NO from a specific worker w_i (out of b) who has not been asked before, for all i. Therefore, the probability of getting to a state S (and the item not satisfying the filter) is the sum of the probabilities of getting to one of the previous states R that is identical to S but has one x_i or y_i diminished by 1 (with the item not satisfying the filter), and getting the appropriate answer (YES/NO) from the appropriate worker i. (Naturally, the states R that are invalid are omitted from the summation.)

For a strategy, we define the states S for which p_0 or p_1 are non-zero as *reachable states* (that is, there is a non-zero probability of reaching them while executing a strategy.)

Problem: Given parameters selectivity s, false negative rates $e_1(w_i)$, and false positive rates $e_0(w_i)$, we consider the following problem:

Problem 1 (Abilities). *Given an error threshold τ and a budget threshold per item m, find a strategy that minimizes C such that $E < \tau$ and $\forall$ reachable $(x_1, y_1, \ldots, x_r, y_r) : \sum_{i \in 1 \ldots r}(x_i + y_i) \leq m$*

Since we want to ensure that the strategy terminates, we enforce a threshold m on the maximum number of questions we can ask for any item (which is nothing but a maximum budget for any item that we want to filter). In our peer evaluation setting, the problem above translates to a minimization of the average number of evaluations assuming a threshold on error rate, a maximum number of graders per submission and knowledge of per-worker error rates.

Solution Intuition: We present a solution to Problem 1 that generalizes the solution in CrowdScreen, but is simpler to understand. Our solution leverages linear programming.

Even though the equations describing relationships between variables are highly non-linear (ref. Equation 1, 2), we can convert these into linear equations by considering the flow of *paths*. A path is a specific sequence of answers that can be used to get to a given state S. Returning to our previous example (Figure 1), one path to get to S_0 may be w_1 answering YES first, followed by w_2 answering NO. The only other possible path is for w_2 to answer NO followed by w_1 answering YES. These paths are depicted in Figure 1(a); the only way to get from $(0, 0, 0, 0)$ to $(1, 0, 0, 1)$ is via one of these two paths. Of course, not all paths may be possible in a strategy. That is, it may be possible that a strategy stops and returns Fail when w_2 answers NO, and therefore, the latter path is not feasible.

We define a new variable called $path(S)$ to denote the number of paths from the origin $(0, \ldots, 0)$ to S within a strategy. In our example above, assume the strategy always continues asking questions at $(0, 0, 0, 0)$, $(1, 0, 0, 0)$, and $(0, 0, 0, 1)$ (that is, $a_{cont} = 1$ for all of these points). Then $path(S_0)$ is 2, since both paths are feasible. This case is depicted in Figure 1(b).

Now, assume that from $(1, 0, 0, 0)$ the strategy continues only with 0.5 probability (i.e., $a_{cont} = 0.5$). In this case $path(S_0)$ is equal to 1.5 — one path via $(0, 0, 0, 1)$, and half a path via $(1, 0, 0, 0)$. This case is depicted in Figure 1(c).

In fact, $path(S)$ can be computed recursively. Continuing with our example, observe that the number of ways of getting to S_0, i.e., $path(S_0)$, is equal to the sum of the number of ways of reaching S_0 via $S_1 = (0, 0, 0, 1)$, which we denote $path_{cont}(S_1)$, and the number of ways of reaching S_0 via $S_2 = (1, 0, 0, 0)$, which we denote $path_{cont}(S_2)$. If, for instance, $a_{cont}(S_1) = 0$, that is, the strategy always terminates at S_1, then, the former number (i.e., $path_{cont}(S_1)$) is 0. If, on the other hand, $a_{cont}(S_1) = 0.5$, that is, with probability half, the strategy asks an additional question at S_1, then the former number $path_{cont}(S_1) = 0.5 \times path(S_1)$, i.e., the number of paths leaving S_1 is half the number of paths reaching S_1 (alternatively, half the number of ways to reach S_1).

Solution Details: We now present the details. From each state S, the incoming paths, $path(S)$, flow onward to other states in the following manner: $path_{pass}(S)$ is the fraction of paths that stop at S with the strategy returning Pass; $path_{fail}(S)$ is the fraction of paths that stop at S with the strategy returning Fail; and $path_{cont}(S)$ is the fraction of paths that continue onward to other states.

We therefore have:

$$\begin{aligned} path_{pass}(S) &= a_{pass}(S) \times path(S) \\ path_{fail}(S) &= a_{fail}(S) \times path(S) \\ path(S) &= path_{cont}(S) + path_{pass}(S) + path_{fail}(S) \end{aligned}$$

That is, a_{pass} and a_{fail} alternatively represent the fraction of the path at a state that is lost by returning a Pass or Fail decision respectively. The remaining (fractional) number of paths, $path_{cont}$, continue onward to other states. Via conservation of paths, we have:

$$\begin{aligned} & path(x_1, y_1, x_2, y_2, \ldots, x_r, y_r) = \\ & \sum_{1 \leq i \leq r;\ x_i = 1} path_{cont}(x_1, y_1, x_2, y_2, \ldots, x_{i-1}, y_{i-1}, 0, 0, \ldots, x_r, y_r) + \\ & \sum_{1 \leq i \leq r;\ y_i = 1} path_{cont}(x_1, y_1, x_2, y_2, \ldots, x_{i-1}, y_{i-1}, 0, 0, \ldots, x_r, y_r) \end{aligned} \tag{3}$$

In other words, path flow can come to a state by asking any one of the r workers and getting one of the two possible answers (YES/NO).

It can be shown that the rest of the variables are linear equalities on the $path$ variables. For some constants $const(S)$, $const'(S)$ (independent of the strategy and only dependent on S) we have:

$$\begin{aligned} p_0(S) &= const(S) \times path(S) \\ p_1(S) &= const(S) \times path(S) \\ E = \textstyle\sum_S &\ const(S) \cdot path_{pass}(S) + const'(S) \cdot path_{fail}(S) \\ C = \textstyle\sum_S &\ const(S) \cdot path_{pass}(S) + const'(S) \cdot path_{fail}(S) \end{aligned}$$

We prove the first equation using induction. The proofs of the remaining identities are similar.

Proof. First, for $S = (0, \ldots, 0)$, $path(S) = 1$ and $p_0(S) = (1 - s)$, so the identity is trivially true for the start state, independent of the strategy. We now assume that the strategy is fixed, and that we have proved that the identity holds for all S' such that $|S'| < |S|$, where $|S|$ denotes the number of non-zero entries in the state S (i.e., the number of workers that have answered when the strategy reaches S). We show that this identity holds true for the $const(S)$ to be set as follows:

$$const(S) = (1-s) \times \prod_{r \geq y \geq r-|S|+1} \frac{1}{y} \times \prod_{w_i \text{ said YES } \in S} e_0(w_i) \times \prod_{w_i \text{ said NO } \in S} (1 - e_0(w_i))$$

For convenience, we denote $prev^+(S)$ to be the set of all states R, for whom $|R| = |S| - 1$, and R's non-zero entries are a subset of those in S with R missing precisely one YES from one of the workers, while we denote $prev^-(S)$ to be the set of all states R, for whom $|R| = |S| - 1$, and R's non-zero entries are a subset of those in S with R missing precisely one NO from one of the workers. (Naturally, if the state S does not comprise any workers saying either YES or any workers saying NO, one of these sets may be empty.) Now,

$$\begin{aligned} p_0(S) \quad = & \sum_{\substack{R \in prev^+(S); \\ w_j \text{is the missing worker}}} \frac{1}{r - |S| + 1} \times p_0(R) \times a_{cont}(R) \times e_0(w_j) \\ + & \sum_{\substack{R \in prev^-(S); \\ w_j \text{is the missing worker}}} \frac{1}{r - |S| + 1} \times p_0(R) \times a_{cont}(R) \times (1 - e_0(w_j)) \end{aligned}$$

Using the induction hypothesis, we have each of the terms in the first summation to be equal to the following:

$$\begin{aligned} & \frac{1}{r - |S| + 1} \times p_0(R) \times a_{cont}(R) \times e_0(w_j) \\ = & \frac{1}{r - |S| + 1} \times const(R) \times path(R) \times a_{cont}(R) \times e_0(w_j) \\ = & \ const(S) \times a_{cont}(R) \times path(R) \end{aligned}$$

Similar identities hold for the second summation. Thus, we have:

$$\begin{aligned} p_0(S) &= \sum_{R \in prev^+(S) \cup prev^-(S)} const(S) \times a_{cont}(R) \times path(R) \\ &= const(S) \times \sum_{R \in prev^+(S) \cup prev^-(S)} a_{cont}(R) \times path(R) \\ &= const(S) \times path(S) \end{aligned}$$

Hence proved. □

Thus, we have linear equations relating all the variables of interest, along with corner cases:

$$\begin{aligned} path(0, \ldots, 0) &= 1 \\ \forall S = (x_1, y_1, \ldots, x_n, y_n);\ \sum_{i \in 1 \ldots r} [x_i + y_i] = m + 1 :\ path(S) &= 0 \end{aligned}$$

The objectives are linear as well, enabling a linear programming solution. Recall that the complexity of linear programming is polynomial in the number of variables (with an exponent of 3.5), and logarithmic in the problem encoding. Our Linear Program (LP) has a total of $O(m^{2r})$ variables, with total size of the LP encoding being $O(poly(m^r))$. As a result, the complexity of the solution is: $O((m^{2r})^{3.5} \times \log(poly(m^r))$, i.e., $O(m^{7r}\, r \log m)$. Thus, we have:

Theorem 3.1 (Abilities). *We can find the optimal strategy for Problem 1 in $O(m^{7r} r \log m)$.*

Note that our techniques are easily adapted to the somewhat simpler setting when there are different worker classes, each with an infinite number of workers. In this setting, there are r classes of workers with error rates $e_0(w_i), e_1(w_i), i \in 1 \dots r$, such that there are infinitely many workers in each class (and each worker in a given class has the same error rate). Here, even after receiving an answer from a worker in one class, we may still get additional (possibly different) answers from workers in the same class (with the same error rate). We return to this simpler setting in the posterior-based representation section (Section 4).

Discussion: The astute reader may have noticed that the complexity, especially when r is large, can be rather high, due to r being present in the exponent of the complexity expression. Therefore, the linear programming approach will not scale if r is large. Instead, we will need to resort to an approximate approach, presented in Section 4.

3.2 Incorporating Prior Information

We now generalize our algorithms for when we have prior information about items.

Modified Setting: Instead of having a single prior probability or selectivity s for all items, we now have prior probabilities s_i for each item i, representing the probability of the item satisfying the filter. We let $s'_1, s'_2, \dots, s'_l$ be the *distinct* set of prior probabilities. The number l may be typically much smaller than n, the total number of items. For instance, if $s_1 = s_2 = s_5 = 0.8, s_3 = s_4 = s_6 = 0.3$, then $s'_1 = 0.8, s'_2 = 0.3$, i.e., $l = 2$.

Solution: Our algorithm will effectively design a distinct strategy for each distinct probability s'_j. We characterize the new state space as: $(x_1, y_1, \dots, x_r, y_r, j), j \in 1 \dots l$. The last coordinate in this state space encodes the a-priori probability of the item we operate on.

Each item with priori probability $s_i = s'_j$ will then begin filtering at state $(0, \dots, 0, j)$, i.e., the start state for the strategy corresponding to s'_j. On asking a question and getting an answer from a worker, we transition from $(x_1, y_1, \dots, x_r, y_r, j)$ to $(x_1, y_1, \dots, x_i + 1, y_i, \dots, x_r, y_r, j)$ or $(x_1, y_1, \dots, x_i, y_i + 1, \dots, x_r, y_r, j)$ — that is, the last coordinate remains fixed, while one of the other coordinates is incremented based on the given worker answer.

When computing the strategies in the previous section, we had set $p_0(0, 0, \dots, 0) = 1 - s$. Here, we have a different probability p_0 depending on the last coordinate. We have,

$$p_0(0, 0, \dots, 0, j) = frac(s'_j) \times (1 - s'_j)$$

The probability above is a product of two factors: The first factor is the fraction of items that begin at $(0, \dots, 0, j)$, i.e., the probability that any item has prior probability s'_j. The second factor is the probability that an item does not satisfy the filter, given that it begins processing at $(0, \dots, 0, j)$—i.e., $(1 - s'_j)$

Note that by adding another index to our state space, we are effectively constructing one strategy for each s'_j: intuitively, for s'_j close to 1 or 0, we expect the strategies to be "small" (i.e., have a small number of reachable states), while for s'_j close to 0.5, the strategy may be larger, since that case is more uncertain.

Given these modifications, once again, the path flow property can be leveraged to derive a linear program, giving us:

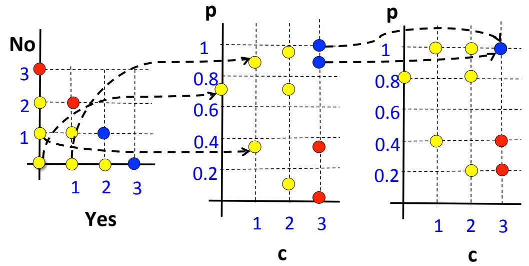

Figure 2: Mapping Between Representations: Left: Answer-record Representation; Middle: Continuous Posterior-based Representation; Right: Discretized Posterior-based Representation

Theorem 3.2 (Problem 1+Priors). *We can find the optimal strategy for Problem 1 with priors in $O(m^{7r} r l^{3.5} \log(ml) + m^{2r} l)$, where l is the number of distinct s_i.*

The proof of complexity is a straightforward extension of the argument in the previous section.

Discussion: Notice that even if $r = 1$, l can be as large as n (the total number of items), and as a result, this approach can be rather inefficient when l is large. In particular, the fine-grained probabilities output by machine learning algorithms may give us many distinct prior probabilities. Instead, we will need to resort to an approximate approach, presented in the next section.

Note also that prior information may not always be completely accurate for individual items. However, we do expect that over all items, the statistics may indeed be correct, and in such a case, our guarantees still continue to hold.

4. POSTERIOR-BASED REPRESENTATION

We now describe the posterior-based representation. This representation, unlike the previous representation, is approximate, that is, the representation does not comprise a complete record of the state of processing of an item—some information is lost. However, our technique exposes a knob that allows the calling application to control the amount of lost information (at the cost of added computational complexity): as the amount of information recorded is increased, the cost of the output strategy tends towards reaching optimal cost. In addition, unlike the answer-record representation whose state space scales rapidly (with some parameter dependent on the aspect under consideration), the dimensionality of the posterior-based representation stays constant on adding additional aspects.

We first describe the representation for the basic setting considered in CrowdScreen—when all workers are assumed to be equally error-prone. Considering the basic setting allows us to demonstrate, for a simple case, how the states in the answer-record representation, discussed previously, map to those the posterior-based representation. Then, in the next subsection, we will provide our solution for worker abilities (that we considered in Section 3.1), and then for incorporating prior information (that we considered in Section 3.2).

4.1 Basic Setting

Preliminaries: In the basic setting, when all workers have identical error rates, the answer-record representation has a collection of states S represented using a pair (x, y), where x is the total number of YES answers so far, while y is the total number of NO answers so far, as shown in Figure 2 (left). The figure shows a strategy that has a Cont decision (Yellow) for all states (x, y) such that $x + y \leq 2$, with Pass (blue) for $(3, 0)$ and $(2, 1)$, and Fail (red) for $(0, 3)$, and $(1, 2)$.

Instead, the posterior-based representation has a new state space represented using two components (p, c), encoding a probability p, which represents the probability that an item has $V = 1$ given the answers obtained so far, denoted: $p = \Pr[V = 1|(x, y)]$ and cost c, which represents the total cost incurred so far, i.e., $(x + y)$.

As in the answer-record representation, in this representation, a strategy takes as input a state (p, c), and outputs a probabilistic decision: "Fail", "Pass", or "Continue".

Mapping and Optimality: We show the correspondence by mapping states in the strategy in the answer-record (left) representation to the posterior-based (middle) representation. The mapping is shown in Figure 2 for $s = 0.7, e_0 = e_1 = 0.2$ with dashed lines for three states (other mappings are omitted for clarity).

As can be seen in the figure, the state $(0, 0)$ maps to precisely $(s, 0) = (0.7, 0)$, since $\Pr[V = 1|(0, 0)] = s = 0.7$, and since $x + y = 0 + 0 = 0$. The state $(1, 0)$ maps to precisely $(0.903, 1)$ since $\Pr[V = 1|(1, 0)] = \frac{se_1}{se_1+(1-s)(1-e_0)} = 0.903$, and since $x + y = 1 + 0 = 1$, while state $(0, 1)$ maps to precisely $(0.368, 1)$ since $\Pr[V = 1|(0, 1)] = 0.368$, and since $x + y = 0 + 1 = 1$.

Thus, each state (x, y) (in the left in the figure) maps to precisely one state in the new representation (in the middle in the figure). It is also easy to see that each state in the new representation can correspond to at most one state (x, y). To see this, let there be two distinct states (x', y') and (x, y) in the answer-record representation that map to the same state in the posterior-based representation. Thus, $x' + y' = x + y$. Now, without loss of generality, let $x' < x$. Then, we can show that the p value associated with (x', y') must be smaller (since there are fewer YES answers from workers with the same error rates). Thus, strategies in the answer-record representation may be represented in the posterior-based representation without loss in information.

Furthermore, there are no better strategies in the posterior-based representation, that is, the best strategy in the posterior-based representation is no better than the best strategy in the answer-record representation. This is because the only states in the posterior-based representation that matter are the ones that are reachable in any answer-record strategy; those are precisely the ones that have a corresponding state in the answer-record representation. Thus:

Theorem 4.1 (Optimality). *For the basic setting (with all workers having identical error rates), the optimal strategy for Problem 1 in the posterior-based representation would have same expected monetary cost as the optimal strategy in the answer-record representation.*

Approximation: The theorem above states that the optimal strategy in the posterior-based representation would have same expected cost as the optimal strategy in the answer-record representation. Instead of storing the set of reachable states and computing a strategy using those states, we instead compute strategies on an approximate discretized version of the entire posterior-based representation states; when we consider worker abilities, approximations will become more necessary. As we will see in Section 6, even our approximate solutions have very good performance (i.e., very low cost for same error threshold).

In particular, we approximate the state (p, c) by using discretization. We discretize the probability p into intervals. (Note that c is already discrete.) We use a discretization factor δ, and we divide the $[0, 1]$ interval for p into δ intervals of size $1/\delta$ each. For instance, if $\delta = 2$, then $[0, 1]$ will be divided into two parts: $[0, 0.5], (0.5, 1]$. The larger the discretization factor δ, the smaller will be our intervals, and our intervals will be more in number.

Our state space S is now restricted to (p, c) where p is an integer multiple of $1/\delta$. The discretized state space is depicted in Figure 2 (Right) for $\delta = 5$. As can be seen in the figure, the two blue states in the full posterior-based representation (middle) map to the same state in the approximate posterior-based representation (right); there are infinitely many mappings from states in the full posterior-based representation to the approximate discrete one, but we omit them from the figure for clarity. For instance, any (p, c), where $a/\delta \le p \le (a + 1)/\delta$, where a is an integer, maps to $(a/\delta, c)$.

Now, if on getting an answer *ans* for a question asked when at state (p, c), (where p is an integer multiple of $1/\delta$), our updated posterior probability value is p', we round p' up to p'', the nearest integer multiple of $1/\delta$, and the new state will be $(p'', c + 1)$.

Approximate Solution: Now that we have a discrete set of states, we can once again use path-based reasoning, and linear programming on paths to find the best strategy, as in the answer-record representation. We now briefly describe how path-based reasoning works in this case: The value $path(p, c)$ is the sum of $path_{cont}$ from all $(p', c - 1)$ such that receiving either a YES or a NO answer from any worker leads to the point (p, c). Essentially, we can define a recurrence expression that allows us to compute $path(p, c)$ in a manner similar to Equation 3. (Our approximate posterior-based representation is once again a Markov Decision Process: we have a set of states, decisions to be made at each state, and probabilistic transitions from each state based on the decision that is made.)

We therefore have the following theorem, using a straightforward extension of the complexity argument of Theorem 3.1:

Theorem 4.2. *We can find a strategy using the approximate posterior-based representation for a variant of Problem 1 with all workers having identical error rates, in $O(\delta^{3.5} m^{3.5} \log(m\delta))$, where δ is the discretization factor.*

Thus, by adjusting δ, the user can control how much time they are willing to invest in finding a strategy. The more they invest, the lower the monetary cost of the strategy will be, as we will see next.

Convergence: As we saw in Figure 2, multiple states in the answer-record representation may map to the same state in the approximate posterior-based representation. For example, the two blue states in the answer-record representation (left) map to two separate states in the full posterior-based representation (middle), both of which map to a single discrete blue state in the approximate posterior-based representation (right). However, as we increase the discretization factor (and therefore the number of intervals), the likelihood that multiple states in the answer-record representation will map to the same discrete state in the approximate posterior-based representation will go down; as a result, the cost of the optimal strategy in the approximate discrete posterior-based representation tends towards optimal cost. Formally,

Theorem 4.3 (Asymptotic Optimality). *As $\delta \to \infty$, the cost of the optimal strategy in the approximate posterior-based representation will tend to the cost of the optimal strategy in the answer-based representation.*

Discussion: In this subsection, we demonstrated that even though the strategies computed using the approximate posterior-based representation do not achieve the same low monetary cost of the exact answer-record representation, we can get as close as we want to that cost by varying the user-controlled discretization factor δ.

This guarantee seems to not be that useful for the basic setting with an infinite pool of identical workers, where the answer-record representation leads to a tractable solution. However, we will find that similar guarantees hold for the aspects considered next. While tractable solutions are not possible for those aspects using the answer-record representation, they are indeed possible with the posterior-based representation.

4.2 Worker Abilities

Recall that in Section 3.1, for the answer-record representation, we found that representing the answers from each worker individually led to an explosion in the state space. In this section, we describe how we may leverage the posterior-based representation when we have worker abilities without a similar explosion.

Also in Section 3.1, we had briefly mentioned that our techniques would directly apply to the simpler setting of infinite worker classes, where instead of having r distinct workers, we had r infinite worker classes, with each class having a distinct error rate. The key difference is that if one of the r workers answers a question on an item, that worker will not be asked from that point on; while in the r infinite worker classes case, the same worker class may be used multiple times on the same item.

Here, we revisit that setting first: our guarantees for asymptotic optimality only hold for the simpler setting of r worker classes, and do not hold for the setting of r distinct workers. We will return to this point once we finish our treatment of r worker classes.

We begin by describing the changes in representation that apply to the entire section. We then discuss the infinite worker classes case (along with the associated optimality guarantees), and the r distinct workers case.

Changes in Representation: Unlike in the answer-record representation, where we had $2r$ coordinates in the representation corresponding to the r workers, here, the posterior-based representation continues to use two coordinates (p, c). Thus, the size of the posterior-based state space does not change when we have many workers with different abilities — but, as we will see later, the cost of computing the strategy does change.

We briefly discuss how transitions happen in this new representation; in short, the only change is how the probability p is updated on receiving an answer. At a state (p, c), if a Cont decision is made (instead of a Pass or a Fail), then a worker is asked the question on the item, and the probability p (of $V = 1$ given the set of answers, including the new answer), is updated to p', and c is updated to $c+1$. We then transition to $(p'', c + 1)$, where p'' is p' rounded up to the nearest multiple of $1/\delta$.

Infinite Worker Classes: We first consider the full posterior-based representation before discretization, and then discuss discretization. Recall that in the infinite worker classes case, instead of having r workers with different abilities, we have r infinite worker classes. That is, there are r classes of workers, such that, at any state, with probability $1/r$, our question is answered by a worker with error rate $e_0(w_1), e_1(w_1)$, with probability $1/r$, by a worker with error rate $e_0(w_2), e_1(w_2)$, and so on. These classes are infinite; that is, if we sample a worker from class 1, the probability of getting a worker from class 1 does not change in the future. In this scenario, the answer-record representation is the same $S = (x_1, y_1, \ldots, x_r, y_r)$: but with one difference; in the previous setting, at most one of x_i or y_i is 1; here x_i or y_i can both be as large as m (because there may be as many as m YES or NO answers from a given worker class — recall that our cost bound per item is m.)

We state the following lemma without proof.

Lemma 4.4 (No Loss of Information). *With infinite worker classes, all states $s_1, s_2, \ldots, s_a$ in the answer-record representation that map to the same state $s = (p, c)$ in the full posterior-based representation have the following properties:*

- *For the same answer obtained at $s_1, \ldots, s_a$ (YES/NO from any worker class), the resulting states $s_1', s_2', \ldots, s_a'$ all map to the same state $s' = (p', c')$.*
- *The probability of getting a YES or a NO from a specific worker class at $s_1, \ldots, s_a$, given that a question is asked, is identical for each of $s_1, \ldots, s_a$.*

We can now state the following theorem:

Theorem 4.5 (Optimality with Worker Classes). *With infinite worker classes, the optimal strategy for Problem 1 in the full posterior-based representation has the same cost as the optimal strategy in the answer-record representation.*

The proof of the above theorem is not as straightforward as the proof of Theorem 4.1, where we could simply show a one-to-one correspondence between states in the answer-record and posterior-based representations.

Proof. For convenience, we prove the above theorem when there are two worker classes, but our technique will work for the r-worker class case. Let $\mathcal{S}'$ be the subset of states in the posterior-based representation that are reachable if no Pass/Fail decisions are made at any state until $c = m$ (i.e., the probabilities of Pass or Fail are 0 until $c = m$), while $\mathcal{S}$ is the set of all states in the answer-record representation. For each state $s \in \mathcal{S}$ there is a state $s' \in \mathcal{S}'$ which it maps to. For each state $s' \in \mathcal{S}'$, there can be multiple states $s_1, s_2, \ldots, \in \mathcal{S}$ that map to it.

We show that for every strategy in the answer-record representation, we can find a better strategy in the answer-record representation such that the same decision is made for all states that map to the same state in the posterior-based representation. This strategy is therefore a strategy in the posterior-based representation.

We use induction starting from states at $(*, m)$. Consider the set of all reachable states in $\mathcal{S}$ that map to a specific (p, m_0) in the posterior-based representation. It is easy to show that the same decision must be made at all those states (Pass or Fail). Else, we can improve the strategy by ensuring that the same (better) decision is made at all states. Thus, all the states in $\mathcal{S}$ mapping to (p, m) must have the same decision.

Now assume that all states until $c = i$ have been mapped to their corresponding states, with the identical decisions for all states in $\mathcal{S}$. Consider states at $c = i-1$. Let there be two states such that they both map to $(p, i - 1)$. These states are identical in that they transition to the same states after asking a question. Without loss of generality, assume that Pass is better at the states than Fail. In that case, the two states can only be improved by having the Fail probability set to 0. Now, we have two states such that the Continue probability is x for one, and y for the other with $x < y$. Let the current path flow into the first state be f_1 and the path flow into the second state be f_2. The lemma above indicates that the path flow from both these states will be indistinguishable to future states. If we set the continue probability for both states to be $(f_1x + f_2y)/(f_1 + f_2)$, then the same amount of path flow leaves both the states, and the strategy is unchanged in terms of cost and error. This generalizes to the case when we have multiple states mapping to the same one.

Therefore, all states in $\mathcal{S}$ mapping to $(p, i - 1)$ for all p in the optimal strategy must have the same decision.

Hence proved. □

Furthermore, we have:

Theorem 4.6 (Asymptotic Optimality). *For infinite worker classes, as $\delta \to \infty$, the cost of the optimal strategy in the approximate posterior-based representation will tend to the cost of the optimal strategy in the full posterior-based representation.*

Approximation to Worker Abilities: While we have proved optimality for the posterior-based representation for infinite worker classes, the proof does not capture the worker abilities aspect discussed in Section 3.1 precisely, because as soon as a worker answers a question (with a YES/NO), the worker can no longer be asked any further questions. Therefore, even as δ increases without bound, by representing the state using just two numbers p and c, we are certainly losing information if we do not record *exactly which worker gave us which answer* (like we do in the answer record representation). Thus, our solution will be necessarily approximate.

We now further approximate via discretization (as discussed in the previous section). Therefore, in adapting to r distinct workers, there are two sources of approximation: one, from discretization (like we saw in the previous section), and second, from using the r infinite worker classes approach for the r distinct worker case.

However, as we will see in the experiments in Section 6, the two approximations we have made do not hurt performance. We have the following, a straightforward extension of the complexity argument of Theorem 3.1:

Theorem 4.7. *We can find a posterior-based strategy for the Problem 1 with worker abilities provided, in $O(m^{3.5}\delta^{3.5}\log(mr\delta) + m\delta r)$, where δ is the discretization factor.*

Notice that r appears as a logarithmic factor in first term of the complexity. This is because the linear equations in the linear program scale up by $O(r)$ — we need to consider transitions from each state (p, c) based on r possible answers: YES/NO from each worker. Since the complexity is no longer exponential in r, it is much faster to compute the optimal strategy in the approximate posterior based representation than it is in the answer-record representation.

4.3 Incorporating Prior Information

We now consider the aspect described in Section 3.2. Recall that our approach for the answer-record representation was to have a strategy computed for each individual distinct prior probability s'_j value as provided by an automated algorithm or human. This number could be as large as $O(n)$, where n is the number of items. As a result, our solution, even when the number of workers or worker classes is small, ended up being difficult to compute.

We now discuss how we may leverage the posterior-based representation $S = (p, c)$ for this aspect. We discuss our solution for the basic setting, that is, when all workers are identical, though our technique is generalizable to when we have distinct worker abilities.

The key idea that we use is to set the path flow into $(s'_j, 0)$ to be equal to $frac(s'_j)$, i.e., the fraction of items with prior probability s'_j. Thus, the total path flow into all states with cost $c = 0$ is still 1, as before. For instance, if we had 50% of the items with prior probability 0.4, and the remaining with prior probability 0.8, then we would start half a path at $(0.4, 0)$, and half a path at $(0.8, 0)$.

With the full posterior-based representation, the optimal strategy has just as low cost as the answer-record representation, formalized in the theorem below:

Theorem 4.8 (Optimality). *With $s_1, s_2, \ldots, s_n$ provided, the optimal strategy in the posterior-based representation for Problem 1 (with identical workers) has the same cost as the optimal strategy in the answer-record representation.*

We will now discretize the probability p, as before. As we increase the discretization factor δ, the cost of the optimal strategy in the discretized posterior-based representation will tend to the cost of the optimal strategy in the full posterior-based representation.

Theorem 4.9 (Asymptotic Optimality with Priors). *As $\delta \rightarrow \infty$, with priors, the cost of the optimal strategy for Problem 1 (with identical workers) in the approximate posterior-based representation will tend to the cost of the optimal strategy in the answer-record representation.*

We then have:

Theorem 4.10. *We can find a posterior-based strategy for Problem 1 with prior probabilities provided, in $O(\delta^{3.5}m^{3.5}\log(ml\delta)+lm\delta)$, where δ is the discretization factor.*

Thus, unlike the answer-record representation, this representation does not have a computationally expensive $O(n^{3.5})$ factor.

Note that for the same reasons as the previous subsection (i.e., that we do not keep track of which workers answered questions), the asymptotic optimality guarantee no longer holds for distinct worker abilities.

5. OTHER ASPECTS

We now discuss other aspects described in the introduction. We first discuss aspects that improve the cost (for fixed error), then discuss aspects that provide enhanced functionality. All these aspects can be captured by the two representations discussed previously.

For the sake of clarity, we describe the addition of each individual aspect one at a time to the basic setting discussed in the previous section. In practice, we may wish to use all the aspects at once. It is straightforward to construct the solution involving all aspects at the same time.

Furthermore, we describe our solution for the answer-record representation. It is straightforward to construct the posterior-based representation based on our solution for the answer-record representation.

We consider the following aspects:

- *Improvements:*
  - *Latent Difficulty:* The algorithms in CrowdScreen assume that all items are equally hard or equally easy to filter—that is, they assume that all humans have the same error rate on every item. However, this assumption may not hold in practice. As an example, checking if a blurry picture contains a cat is much harder to do (and is more error-prone) than a clear picture. Furthermore, difficulty information is not provided to us up-front; we need to infer if an item is difficult or not based on answers we get from humans.
- *Functionality Additions:*
  - *Requesting Specific Workers:* The algorithms in CrowdScreen do not request that specific workers answer, nor pay workers differently. In the marketplace ODesk [5], for instance, there are better qualified workers who are paid more while not-so well qualified workers who are paid less, and for any question, we may choose to use a more qualified or less qualified worker. Here, we consider the addition of the functionality of being able to request that specific workers answer and being able to pay them different amounts.
  - *Latency:* The problem statements described so far only have monetary cost and error as objectives, not latency. Latency is important in many crowdsourcing applications. We consider the addition of a simple latency constraint in our problem statement.
  - *Scoring:* The problem statements described so far only consider binary filtering: we would also like to perform scoring, i.e., identifying the appropriate score or rating of an item, say from 1 . . . 5. Furthermore, we allow weighted error objectives. For instance, it is much worse to score an item with true rating 1, as a 5, instead of a 2. Briefly, scoring is handled by increasing the dimensionality of states by recording if a worker gave an item a score of $1, 2, \ldots, r$.

5.1 Outline and Summary of Results

In Table 1, we show the complexity results for each of the aspects considered; the two columns correspond to the complexity of algorithm computing the strategy using the answer-record representation, and the complexity of the algorithm computing the strategy using the posterior-based representation.

We divide the rows into two parts: the complexity on adding each of the individual improvement-based aspects to the setting with worker abilities, followed by the complexity on adding each of the functionality addition-based aspects to the setting with worker abilities.

5.2 Overall Approach

Our approach is to construct a representation that records all possible information pertaining to the state of processing of an item (possibly by adding additional dimensions), and then leveraging path

Functionality	Answer-record	Approximate Posterior-based (δ)
Problem 1 (Worker Abilities)	$m^{7r} r \log m$	$(m\delta)^{3.5} \log(mr\delta) + m\delta r$
Problem 1 (Worker Abilities)+Priors	$m^{7r} r l^{3.5} \log(ml) + m^{2r} l$	$(m\delta)^{3.5} \log(mrl\delta) + m\delta rl$
Problem 1 (Worker Abilities)+Difficulty	$m^{7r} r \log(md) + m^{2r} d$	$(m\delta)^{3.5} \log(mrd\delta) + m\delta rd$
Problem 1 (Worker Abilities)+Picking Workers	$m^{7r} r \log m$	$(m\delta)^{3.5} \log(mr\delta) + m\delta r$
Problem 1 (Worker Abilities)+Latency	$m^{7r} r t_0^{3.5} \log(mt_0) + m^{2r} t_0$	$(m\delta t_0)^{3.5} \log(mr\delta t_0) + m\delta r t_0$
Problem 1 (Worker Abilities)+Scores	$m^{3.5ru} ru \log m + m^{ru}$	$(m\delta^{(u-1)})^{3.5} u \log(mr\delta) + m\delta^{u-1} r$

Table 1: Comparison of complexity: For clarity, we only show the complexity of adding one aspect at a time to the setting of Problem 1; it is straightforward to construct the solution for applying all the aspects at the same time. Notation: m is the upper-bound on cost, r is the number of workers, δ is the discretization factor, d is the number of distinct error rates, l is the number of distinct probabilities, t_0 is the upper-bound on latency, and u is the number of distinct scores.

flows so that the optimal strategy can be found using linear programing. That is, we need to show that p_0, p_1, E, C are linear functions of the $path$ variables under new assumptions. This process in fact requires the same steps we have used above for the two aspects of worker abilities and prior information:

- Describing the new assumptions and the new metrics.
- Constructing the state space and arguing that the state space captures all the information necessary for a strategy to make a decision at a state.
- Describing the set of possible decisions made at a state (sometimes more than Pass, Fail, Continue).
- Capturing the metrics of a strategy using non-linear equations like Equations 1, 2 above
- Arguing that by considering path flow, the transformation of non-linear equations into linear ones is correct.

Since describing all the items enumerated above is rather tedious, we will instead only describe the key ideas for each of the aspects.

We begin with the improvement aspects, then discuss functionality addition aspects.

5.3 Improvement: Difficulty

In the previous section, we assumed that the error rate e_0, e_1 is the same for every item — that is, e_0 and e_1 are fixed. However, this assumption may not hold in practice. For instance, when identifying if a picture contains a cat, humans are much more likely to answer the question correctly if the picture is clear than if it is blurry.

Instead of having a single false positive or false negative rate for all items, we assume that there is an inherent known distribution across error rates for items. We have the following (discrete) distributions — estimated in advance using sampling, or using prior history: $\Pr[e_0|V = 0]$ and $\Pr[e_1|V = 1]$

For instance, it may be the case that for items that satisfy the filter ($V = 1$), 10% of the items are really hard for a human to judge, with $e_1 = 0.4$ (i.e., most people make mistakes for those items), and 90% of the items are really easy for a human to judge, with $e_1 = 0.1$ (i.e., most people answer questions correctly for those items).

Then, p_0 — the probability of arriving at a state $S = (x_1, y_1, \ldots, x_r, y_r)$ and the item satisfying the filter — is now a sum of quantities instead of a single quantity: (where $const(S), const'(S)$ are constants that depend just on S and not on the strategy)

$$\begin{aligned}
p_0(S) &= \Pr[\text{reach}(S), V = 0] = \Pr[\text{reach}(S)|V = 0] \cdot \Pr[V = 0] \\
&= \sum_e \Pr[e_0 = e|V = 0] \cdot \Pr[\text{reach}(S)|V = 0 \wedge e_0 = e] \cdot \Pr[V = 0] \\
&= path(S) \sum_e \Pr[e_0 = e|V = 0] \cdot const(S) \\
&= path(S) \cdot const'(S)
\end{aligned}$$

Similar relationships hold for p_1. It is easy to see that E, C are also linear functions of the $path$ variables. Thus:

Theorem 5.1 (Difficulty). *We can find the optimal strategy for Problem 1 with difficulty information in $O(m^{7r} r \log mrd + m^{2r} d)$, where d is the number of discrete values for e_0, e_1.*

Incidentally, notice that difficulty also captures the case where errors between humans are correlated; the fact that many humans are making mistakes on the same item can be explained by the item having a higher error rate or difficulty than a different non-ambiguous item.

5.4 Functionality: Picking Workers with Costs

Unlike the situation in Section 3.1, where the strategy is assigned a worker randomly when it chooses to ask a question, we now consider the case where we can choose to ask any one of our r workers, who have differing costs $c_1, \ldots, c_r$. This scenario is relevant in many crowdsourcing marketplaces, such as ODesk, where the worker who is asked a question may be controlled by the application.

The state space is identical to the one described in Section 3.1, that is: $S = (x_1, y_1, x_2, y_2, \ldots, x_r, y_r)$ However, the decision that is made at any state is now not restricted to "Pass", "Fail" or "Continue" — when continuing to process the item, we can choose to ask any one of the r workers (assuming they haven't been asked before). We have:

$$\begin{aligned}
path(x_1, y_1, \ldots, x_r, y_r) &= path_{fail}(x_1, y_1, x_2, y_2, \ldots, x_r, y_r) \\
&+ path_{pass}(x_1, y_1, x_2, y_2, \ldots, x_r, y_r) \\
&+ \sum_{\substack{1 \le i \le r; \\ x_i = y_i = 0}} path^i_{ask}(\ldots, x_{i-1}, y_{i-1}, 0, 0, x_{i+1}, y_{i+1}, \ldots)
\end{aligned}$$

where $path^i_{ask}$ represents the path flow based on asking the i th worker an additional question (if the worker has not answered a question before). Moreover, we have:

$$path(x_1, y_1, x_2, y_2, \ldots, x_r, y_r) = \sum_{1 \le i \le r;\ x_i = 1 \text{ or } y_i = 1} path^i_{ask}(x_1, y_1, \ldots, x_{i-1}, y_{i-1}, 0, 0, x_{i+1}, y_{i+1}, \ldots, x_r, y_r)$$

where $path_{ask}$ is now replaced by $path^i_{ask}$, the path flow on asking the ith human worker. We then have:

Theorem 5.2 (Picking Workers with Costs). *We can find the optimal strategy for Problem 1 with the ability of picking specific workers in $O(m^{7r} r \log m)$.*

Discussion: This aspect suffers from the same problem as the related aspect in Section 3.1: when r is large, the complexity can be rather high, due to r being present in the exponent. Therefore, the approach will not work if r is large.

Furthermore, if the workers have different (non-integral) costs, then our cost coordinate may be rational rather than integral, as a result, we may need to discretize the rational numbers. Adding an additional discretization would only make our solution more approximate.

5.5 Functionality: Latency

So far, we have not considered latency as part of our optimization, focusing instead on cost and accuracy. The reason is the following: our latency is, in the wost case, as large as the time taken for m questions to be answered in sequence. We ask one question on each item, and stop processing for each item based on the strategy, and we do so for all items in parallel. Thus, even in the worst case, the latency is not very large. If latency is critical, we can add a constraint to our objective, in the following manner.

We define latency as the number of round-trips taken to the crowdsourcing marketplace; we assume that multiple questions may be asked in parallel, and if asked in parallel, then would be all answered in one unit of time.

Then, our state space can be represented as: $S = (x_1, y_1, \ldots, x_r, y_r, t)$, where t is the total number of round-trips to the crowdsourcing marketplace taken so far in processing a given item.

We modify Problem 1 to the following by adding a constraint on the absolute number of round-trips:

Problem 2 (Problem 1+Latency). *Given an error threshold τ, a latency threshold per item t_0, and a budget threshold per item m, find a strategy that minimizes C under the constraint $E < \tau$ and $\forall$ reachable $(x_1, y_1, \ldots, x_r, y_r, t)$ $\sum_i [x_i + y_i] \leq m, t \leq t_0$.*

At each state, the strategy may ask one, two, …, or t_0 questions in parallel. Thus, instead of having one variable $path_{ask}$, we have $path^1_{ask} \ldots path^{t_0}_{ask}$, corresponding to path flow on asking $1, 2, \ldots, t_0$ questions in parallel. Also, we have:

$$\begin{array}{rcl} path(x_1, y_1, \ldots, x_r, y_r, t) & = & \sum_i path^i_{ask}(x_1, y_1, \ldots, x_r, y_r, t) \\ & + & path_{pass}(x_1, y_1, \ldots, x_r, y_r, t) \\ & + & path_{fail}(x_1, y_1, \ldots, x_r, y_r, t) \end{array}$$

We then have:

Theorem 5.3 (Latency). *We can find the optimal strategy for Problem 2 in $O(m^{7r} r t_0^{3.5} \log(m t_0) + m^{2r} t_0)$.*

Note that in addition to an upperbound on latency, we can also capture constraints on expected latency across items. For instance, we can enforce the constraint that the expected latency across all items should be less than some number t', or we can minimize expected latency, while enforcing constraints on expected cost and expected error.

5.6 Functionality: Scoring

So far, we have considered boolean filtering, where an item either satisfies or does not satisfy a filter. We can also handle the case where each item can have one of k scores or ratings $V = 1, \ldots, u$, with accuracies or error rates:

$$\Pr[w_k\text{'s answer is } j | V = i] = p_{(i,j)}(w_k)$$

where $p_{(i,j)}(w_k)$ is the probability that worker answers that the rating of an item is j when its actual rating is i. Thus, there is a set of $u \times (u-1)$ numbers that define the accuracy of a worker. (For each $i \in 1 \ldots u$, we need to know $(u-1)$ probability values corresponding to $p_{(i,j)}$ — the last probability value can be inferred from the remaining $u-1$.)

Then, at any point, we can represent the current set of answers using: $S = (x_1^1, x_1^2, \ldots, x_1^u, \ldots, x_r^1, x_r^2, \ldots, x_r^u)$. That is, we record whether each worker has scored the item as $1, 2, \ldots, u$. Once again, the path flow property and linear programming may be leveraged to find the optimal strategy in the answer-record representation.

Moreover, we can also handle error metrics that penalize differently for assigning i or i', $i \neq i'$ to an item whose actual rating is j. For instance, we may penalize i higher than i' if i' is closer to j than i. The function $0 \leq pen(i,j) \leq 1$ refers to the penalty for judging an item with true value j, as i. Now, E is set to be the following expression:

$$E = \sum_S \sum_{j \in 1 \ldots u} p_j(S) \times (\sum_{i, i \neq j} pen(i,j) \cdot a_i(S))$$

where a_i is the probability that a score i is assigned to an item, and $p_i(S)$ is the probability that an item arrives at S with true score i.

6. PEER EVALUATION EXPERIMENTS

We describe the dataset and the setting first, followed by our experimental methodology.

Dataset Description: We validate our algorithms on a real MOOC course dataset—the Human Computer Interaction (HCI) course offered during Fall 2012 at Stanford. The HCI course involved around 1000 students, who were evaluated on five assignments, each containing five problems, for a total of 25 problems. Thus, the total number of student submissions (across all problems) was 25,000.

The course relied entirely on evaluation by peer graders to judge the quality of the student submissions for each problem. Each submission was graded independently by 10 (randomly selected) student graders on average, each grader providing a score between 0–5, both inclusive, i.e., one of six scores. Thus, the total number of evaluations (i.e., questions asked to humans) across all submissions was 250,000, with each grader grading 250,000/1000 = 250 items on average. This dataset is ordered, that is, for every submission, the scores provided by the ten graders are listed in the order in which they were received. For each score assigned to a submission, the identity of the grader who provided the score is also recorded as part of the dataset.

Mapping to Filtering: We treat each student submission on a problem as an item to be scored on a scale from 0—5 (both inclusive). Thus, we are operating under the scoring scenario described in Section 5, instead of filtering items as being YES/NO. Since there are 25 problems, each evaluated by 1000 students, we have a total of 25,000 items to be scored.

Grader Evaluation: The dataset also contains a set of 250 "test" submissions that were graded by all 1000 graders, as well as the course staff (instructors or TAs). These test submissions allows the peer evaluation system to calibrate the error rates of each grader prior to peer evaluation.

Since we are scoring items rather than performing binary filtering, the error rates or accuracies for each grader (or worker) w_k are of the form $p_{(i,j)}(w_k)$, representing the probability that a worker w_k examines an item with staff score j, and assigns it a score of i. Since we have 6 possible scores, each grader's error rate is therefore defined by a set of 36 $p_{(i,j)}$ values. We set the grader's error rate based on his or her performance on the 250 test submissions. Our estimate of $p_{(i,j)}(w_k)$ is simply: the fraction of items whose staff scores are j that the worker w_k judged to be i instead, over the total number of items with a staff score of j.

Complete Maximum Likelihood Score: In our evaluations we will need to compare the score provided by a strategy to the submission's "correct" score. Since we do not have TA scores (except for the test submissions), we will interpret the "correct score" as the best possible estimate if we had available *all* information. Thus we define the *complete maximum likelihood score* for an item a as the score $j \in 0 \ldots 5$ that the item is most likely to be, based on all existing information about a (that is, all grader evaluations of a). We assume that graders evaluate items with known accuracies corresponding to $p_{(i,j)}$. We define this concept more formally below.

We define $L(j,a), j \in 0 \ldots 5$ to be the probability that the score of item a is j, given the evidence we have. That is,

$$L(j,a) = \prod_{w_k\text{'s score for item } a = i} p_{(i,j)}(w_k)$$

Therefore, $L(j,a)$ encodes the product of the probabilities $p_{(i,j)}$ for all workers who looked at the item, and gave it a score of i, for some i. For instance, if worker w_1 gave an item a a score of 3, and worker w_3 gave a a score of 5, then: $L(j,a) = p_{(3,j)}(w_1) \cdot p_{(5,j)}(w_3)$. Now, the *complete maximum likelihood score* of an item a, $V(a)$ is defined as the score j that maximizes $L(j, a)$:

$$V(a) = \arg\max_{j \in 0 \ldots 5} L(j, a)$$

Thus, the complete maximum likelihood score of an item is the score that maximizes the product of the probabilities of the individual grader scores, across all graders who have provided scores for the item. As a result, we use the entire dataset to assign a score to each item. Note that we are overloading V to mean both the "true value" of items (as defined in Section 2), and the complete maximum likelihood score: this is because for all practical purposes, the complete maximum likelihood score is our best estimate of the true value of items given the entire dataset.

Overall Goal and Methodology: The goal of our experiments is to study the trade-off between expected cost and expected error for the filtering strategies output by our algorithms. Our methodology for comparison is to repeat the following for each algorithm:

- For each error threshold $\tau \in [0, 1]$, we execute the algorithm to generate a filtering strategy that obeys the expected error threshold, and is optimized for minimum expected cost.
- We then simulate a run of the generated filtering strategy on each item (i.e., each student submission) in the dataset. When the filtering strategy requests an additional grader score while processing an item, then this score as well as the identity of the grader who provided the score is retrieved from the dataset. That is, when the filtering strategy requests an additional grader score, it is allowed to "see" another score for the item from the dataset (in the order in which the scores were assigned by graders in the first place).
- When the simulation of the strategy on each item terminates, we record both the empirical cost (the number of scores requested for that item), and the empirical error (the difference between the complete maximum likelihood score—as assigned above—and the score output by the strategy for the item).
- We then measure the average empirical cost (i.e., total number of scores requested by the strategy, as a fraction of the total number of scores available in the dataset across all items), and the average empirical error (i.e., the average difference between the score assigned to an item and its complete maximum likelihood score, across all items).
- We repeat the procedure above for a range of τ, recording the average empirical cost and error, giving us a cost-error curve. These cost-error curves allow us to pictorially compare between algorithms over a range of cost and error values.

Algorithms: We evaluate four filtering algorithms. For all algorithms that we study, we use the posterior-based representation (as described in Section 4), wherein the state of processing is recorded using two quantities: the probability p that the item passes the filter, given answers seen so far, and cost spent so far, c. Since we wish to score items from $0 \ldots 5$, instead of performing binary filtering, the probability p is replaced with 5 probabilities $p_0, p_1, \ldots, p_4$, i.e., the probability that the item has score i, $i \in 0 \ldots 4$, given the grader evaluations seen so far. (The probability p_5 that the item has score 5 given the scores seen so far can be inferred from the remaining 5 values, and therefore need not be recorded.) Since we use the posterior-based representation, all the algorithms we study have a discretization factor δ, representing the number of intervals into which we divide the probability coordinate. As we saw in Section 4, the larger the δ, the more fine-grained our probability estimates are, but the more time it takes to compute the strategy.

The algorithms we consider are the following:

- *Single(δ):* This algorithm, for each threshold τ, generates the cost-optimal strategy assuming all workers have the same error rate, using techniques from CrowdScreen.
- *Complete(δ):* This more powerful algorithm, for each threshold τ, generates the cost-optimal strategy assuming worker abilities are all distinct, using techniques from Section 4.1.
- *Var(k, δ):* We define variance as the average difference between the scores provided by the grader and staff scores, as observed during testing (described above). This algorithm, for each value k, first places graders into k equal-sized intervals based on their variance: that is, graders are sorted based on their variance, and then we partition variance into k intervals such that the same number of graders are in each interval. This algorithm, for each error threshold τ, generates the cost-optimal strategy, assuming that workers within each partition have the same error rate. That is, workers in an partition are assumed to be equally capable of evaluating items. When $k = 1000$, i.e., equal to the total number of graders, then this algorithm is identical to the Complete algorithm, since in that case, each grader will be in a partition all by himself or herself. Additionally, when $k = 1$, then this algorithm is identical to the Single algorithm, since in that case all graders will be in the same partition. Thus, this algorithm can be viewed as a generalization of both Single and Complete, as is the Bias algorithm described next.
- *Bias(k, δ):* This algorithm is the same as the previous one, except we partition graders based on bias; we define bias as the average *signed* difference between the scores provided by the grader and staff scores, as observed during the testing period.

We compare the algorithms above to the following two baselines:

- *Median:* This heuristic is currently in use in the Coursera system for peer evaluation. For each submission, the scores given by the randomly selected student graders are combined using the *median* heuristic: that is, the median of the scores for each submission is the final grade assigned to the submission.

  We can generate a cost-error curve for the median algorithm by constraining the cost to be some fraction γ of the maximum possible cost—that is, with probability γ, we include each score assigned to an item while computing the median—and then we measure the error of the median scores assigned (i.e., the difference between the median score and the complete maximum likelihood score, on average across all items). We repeat this for multiple γ to give us a cost-error curve.
- *RMLE (Randomized Maximum Likelihood Estimation):* For each submission, this algorithm combines the scores given by randomly selected graders, using maximum likelihood, that is, the final grade assigned to the submission is the maximum likelihood estimate computed using the set of scores seen so far. As in the Median algorithm, we generate a cost-error curve by constraining the cost to be some fraction γ of the maximum possible cost—that is, with probability γ, we include each score assigned to an item—and then we measure the error of the score assigned by the RMLE algorithm. We repeat this for multiple γ to give us a cost-error curve.

We implemented all of our algorithms in Python and conducted our experiments on a large memory (100 GB) 25 processor server.

Dataset Statistics: We now describe some basic statistics on the dataset. First, we study the number of items evaluated across graders. We place graders in buckets based on the number of items they evaluated, and plot the fraction of graders that have graded a certain number of items in Figure 3(a). For instance, the fraction of graders who have graded between 80—120 items is around 18%. Thus, as can be seen in the figure, even though the average number of items scored across all graders is around 250 items, there are graders who have scored much more than 250 items, and much less as well. It is interesting to note that there is a student grader that has scored close to 1300 items.

Next, we study the bias of graders. We place graders in buckets based on their bias (computed as described above), and plot the frac-

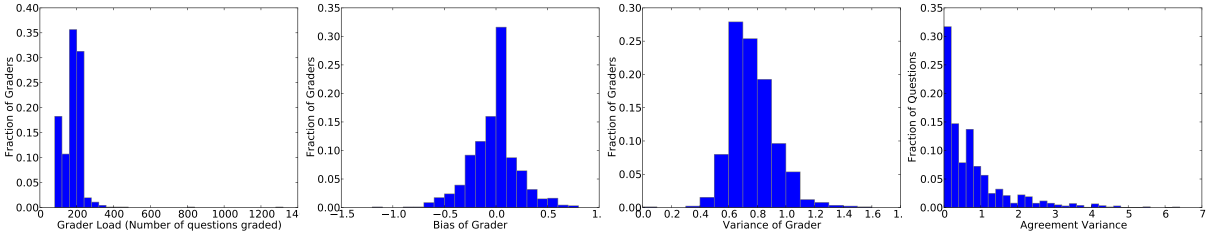

Figure 3: (a) Grader Load (b) Grader Bias (c) Grader Variance (d) Agreement Distribution

tion of graders that have a certain bias in Figure 3(b). As can be seen in the figure, the plot is centered around 0 (i.e., most graders have zero bias), but there are graders who have non-zero positive or negative bias. For instance, aggregating the three bars (approximately, 0.15, 0.10, and 0.10) to the left hand side of 0, we find that more than 1/3 of the graders have negative bias.

Similarly, we plot the variance of graders (as described above) in Figure 3(c). From the figure, we find that very few graders are *perfect*, i.e., have 0 variance. In fact, most of the graders have variance around 0.7.

Next, we wish to study whether some problems are harder to grade correctly than others, i.e., whether the problem itself has an impact on grading accuracy. To do this, we did the following: For each item, we computed the average variance in the evaluations for that item with respect to the items's mean grade. We then computed the average variance for each problem by simply taking the average across all items for that problem. We placed problems into buckets based on the average variance, and then plotted the fraction of problems versus the average variance in the evaluations for that problem (denoted agreement variance in the figure) in Figure 3(d). If we treat agreement variance as the difficulty of grading a problem, we find that many (close to 30%) problems are relatively easy to grade (i.e., with variance close to 0), however, there are problems with variance as high as 4.

Experiment 1: How much benefit do we get from optimized crowd-powered algorithms as compared to simple heuristics, and how much benefit do we get from considering worker abilities?

On comparing Single, Complete, RMLE, and Median on cost and error, we find that for the same error, Complete has significantly lower cost than Single and RMLE, which has significantly lower cost than Median. Additionally, on fixing cost, we find that Complete has significantly lower error than Single and RMLE, both of which have significantly lower cost than Median. For most costs, Complete has 70% of the error of Single and RMLE, and 50% the error of Median.

We use the methodology described above to trace the cost-error curve for Single, Complete (both for $\delta = 10$) — denoted Single-Factor10 and Complete-Factor10 respectively, RMLE, and Median. The results are displayed in Figure 4(a). The figure shows the fraction of the dataset that is "seen" or "consumed" by each of the algorithms (i.e., the total empirical cost) on the y-axis, versus the average difference between the complete maximum likelihood score and the estimated score (i.e., the average empirical error) on the x-axis. As can be seen in the figure, Complete has much lower cost and error than Single, RMLE, and Median. For instance, on fixing cost, say at 50%, which means that each of the algorithms requests 5 scores on average for each submission, Median has an error of 0.57, i.e., on average, the actual score assigned to a student is 0.57 away from the complete maximum likelihood score; Single and RMLE have an error of 0.4, 70% of the error of Median; Complete, on the other hand, has an error of 0.27, just 47% of the error of Median, and just 68% the error of Single or RMLE. (Of course, the better performing algorithms have a higher computational cost, as will be discussed in Experiment 5.)

In Figure 4(b), we trace the cost-error curve for Single (for $\delta = 20$), RMLE, and Median. Here, we find that unlike when $\delta = 10$ (where Single and RMLE performed similarly), Single itself has much lower cost and error than RMLE and Median. For instance, on fixing cost at 50%, Median has an error of 0.57 and RMLE has an error of 0.4, while Single has an error of around 0.3 (i.e., just 75% of the error of RMLE, and 50% of the error of Median.)

Thus, the optimized crowd-powered algorithms—Complete and Single—provide significant benefits in both cost and error over the algorithm currently used in the peer evaluation system (Median), as well as RMLE. This is because these algorithms find strategies that are optimized to make the right decision at every possible intermediate state of processing. Further, we find that Complete does much better than Single; thus there are significant benefits to tracking individual grader abilities rather than assuming that all graders have the same error rate. Since Complete takes into account individual grader abilities, it can appropriately "weigh" differently the same answer coming from two different graders with different abilities. The algorithm Single, on the other hand, is not able to take this information into account. Both Complete and Single do better than RMLE, because they are able to request more evaluations for the more problematic or controversial items. over others. RMLE, on the other hand, requests the same amount of evaluations on average for all items. While Single does not do much better than RMLE for $\delta = 10$, it does much better for $\delta = 20$, indicating that the discretization factor can have a significant impact on cost and error. We explore this aspect subsequently.

Experiment 2: How much does taking worker abilities into account impact performance? That is, how fine-grained should our worker ability partitions be?

On keeping δ fixed, increasing the number of worker partitions has the effect of reducing error for fixed cost, or vice versa. However, the impact of the number of partitions is more pronounced early on (for a small number of partitions), than later on, when the number of partitions is already large. Thus, increasing the number of partitions yields significant savings in cost even though it leads to higher computational cost while computing the strategy.

We next study how our hybrid algorithm for Variance performs in comparison with Single and Complete, on varying k, the number of worker partitions. We fix the discretization factor to be δ, and vary the number of partitions from 1 (i.e., Single), to 10, 100, and then finally to 1000 (i.e., Complete). Figure 4(c) depicts the cost-error curves for each of these four algorithms (the Variance curves are denoted Var-Class-k-Factor10 in the figure.) As can be seen in the figure, there are significant gains to be had in terms of cost and error in increasing the number of grader partitions from 1 to 10, from 10 to 100, and from 100 to 1000. For instance, if we fix the error to be around 0.35, Complete gives us a cost of 40%, Var-Class100-Factor10 (i.e., Variance with $k = 100$) has a cost of around 50%, Var-Class10-

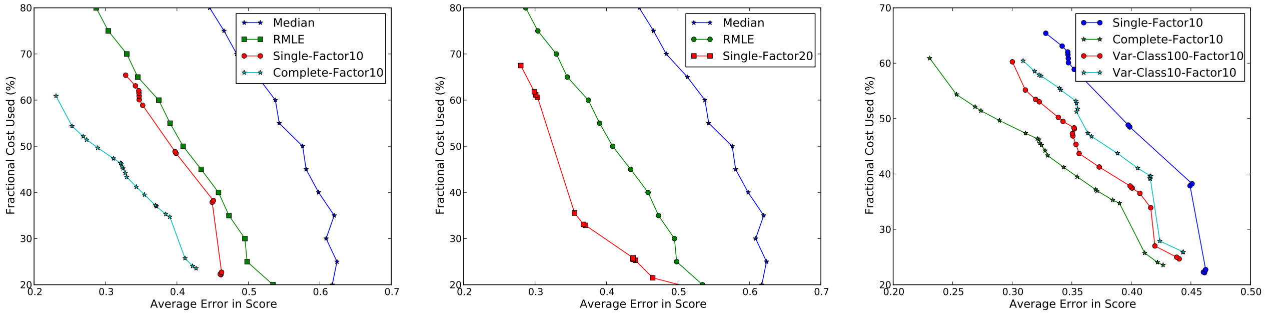

Figure 4: (a) Basic Comparison (b) Basic Comparison: Varying Factor (c) Varying Class Size

Factor10 (i.e., Variance with $k = 10$) has a cost of around 55%, and Single has a cost of around 60%.

As can be also seen in the figure, small changes in k are more likely to impact the cost-error curve when k is small, rather than when k is already large: for instance, the impact of changing k from 1 to 10 is as pronounced as the impact of changing k from 100 to 1000.

Thus, if the computing the strategy is feasible for large k, this figure shows that it is preferable to do so in order to take advantage of the additional cost savings to be had on increasing k. We consider the computational cost on varying k later on.

Experiment 3: How finely should we discretize probabilities?

> On keeping k fixed at 50, increasing the discretization factor has a significant impact on performance: that is, it has the effect of reducing error for fixed cost, or vice versa. Thus, increasing the discretization factor yields significant savings in monetary cost even though it leads to higher computational cost while computing the strategy.

For this experiment, we fix $k = 50$, and let δ be $4, 5, 10$, or 20. (These values of δ were chosen because each of these values are divisors of the number 100.) We then plot the cost-error curves for Variance for these values of δ in Figure 5(a), and for Bias for these values of δ in Figure 5(b). As can be seen in the figure, the cost-error curves for $\delta = 4$ or 5 are not as smooth as the ones for $\delta = 10$ or 20: this is because when the probability discretization is so coarse-grained, then there is a lot more noise, and the trade-off between cost and error is less predictable.

Further, as we can see here, as we increase δ, there are significant gains in both cost and error. For instance, in Figure 5(a), for error being equal to 0.35 the cost for $\delta = 20$ is 35%, while the cost for $\delta = 10$ is 45%, an almost 30% increase. The cost-error curves for 4 or 5 never manage to achieve error 0.35.

Thus, these set of results dictate that we should use as high a δ as possible, to profit from the gains in both monetary cost and error. However, increasing δ leads to much higher computational and storage cost. In fact, in our experiments, we were not able to compute the strategy for $\delta = 25$: this is because even storing the strategy (in a memoized form) would require an array of $10 \times 25^5 \times 50 \times 6 \approx 30$ Billion entries, which is more than we could manage on our Ubuntu server. We will study this aspect in more detail later.

Experiment 4: How should we partition graders?

> Partitioning graders on bias or variance gives similar results.

In Figure 5(c), we study the difference between using Bias or Variance to partition graders. We let $k = 50$, and plot the cost-error curves for both Bias and Variance for $\delta = 10$ and 20. As can be seen in the figure, Bias and Variance perform similarly: while it seems like Variance is better for higher δ and Bias is better for lower δ, these changes may be attributed to experimental noise, rather than to some systematic variation. Overall, using Bias to partition graders is just as good as using Variance.

Experiment 5: How does the computational cost of computing a strategy vary with k or δ?

> The cost of computing a strategy grows linearly with k and polynomially with δ.

We focus on the Variance worker partition scheme, and plot the cost of computing the strategy in minutes versus δ for different values of k: 1 (same as Single), 10, and 100, shown in Figure 6(a) and 6(b) (Figure 6(b) is the same as Figure 6(a), but with the y-axis in log scale). As you can see in Figure 6(a), the time to compute the strategy increases very rapidly with δ: for instance, for $k = 100$, the time varies from less than 10 minutes for $\delta = 4$, to three hours for $\delta = 10$, to half a day for $\delta = 20$. The growth curve is convex (i.e., the rate of change increases as we increase δ) for each of the three plots corresponding to different k. In our analysis of the posterior-based representation for the multiple scores case (Section 5.6), we mentioned that the complexity of representing the strategy itself (and computing it) is proportional to a large polynomial of δ, thus the experimental results confirm the theoretical analysis.

In Figure 6(b), the trend on increasing k is clear: for each value of δ, the difference between the log of the computation time for $k = 100$ and 10 is the same as the difference between that for $k = 10$ and 1 (for all δ). Thus, (a) the ratio between the time to compute strategies is proportional to the ratio of k values (b) this ratio is the same independent of δ. Thus, as predicted by theoretical analysis, the time to compute the strategy is linearly proportional to k.

Thus, the cost of computing strategies increases polynomially with δ and linearly with k. On the other hand, the cost of storing strategies increases polynomially with δ, but is not dependent on k.

Experiment 6: Should we increase k or δ?

> Both k and δ affect cost and error significantly; however, it may be preferable to increase k first, since it increases the complexity linearly rather than polynomially (as in the case of δ).

We focus on the Variance worker partition scheme, and consider two values each of k and δ: $k = 10, 100$, and $\delta = 10, 20$: we plot the cost-error curves for these four algorithms in Figure 6(c). We find that the two curves for $\delta = 20$, and the two curves for $\delta = 10$ perform similarly, with the curve for $k = 100$ performing better than the curve for $k = 10$ in both cases. However, the curves for $\delta = 10$ perform worse than $\delta = 20$. Thus, δ has a larger impact on cost and error than k. This impact comes at a price: the computational complexity is proportional to a large polynomial of δ, while being linearly proportional to k. And since the number of k values is not likely to be very large (in the hundreds or thousands, rather than the millions), it may be preferable to increase k first before δ.

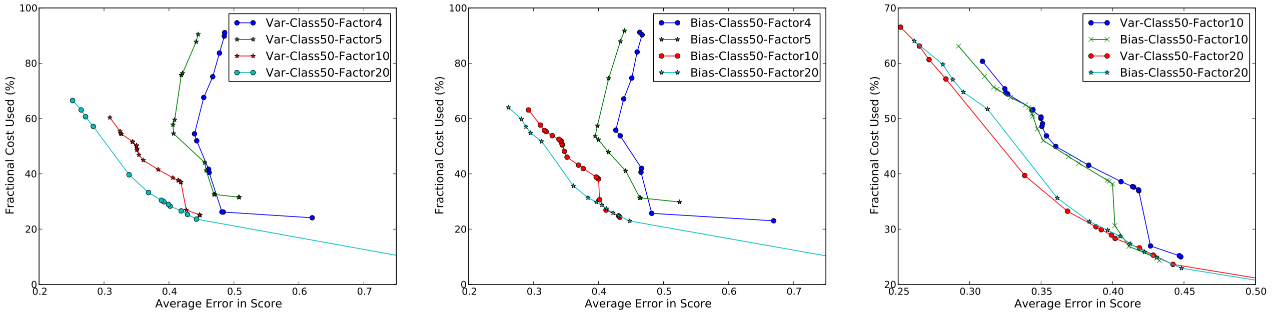

Figure 5: (a) Variance with Factor (b) Bias with Factor (c) Bias vs. Variance

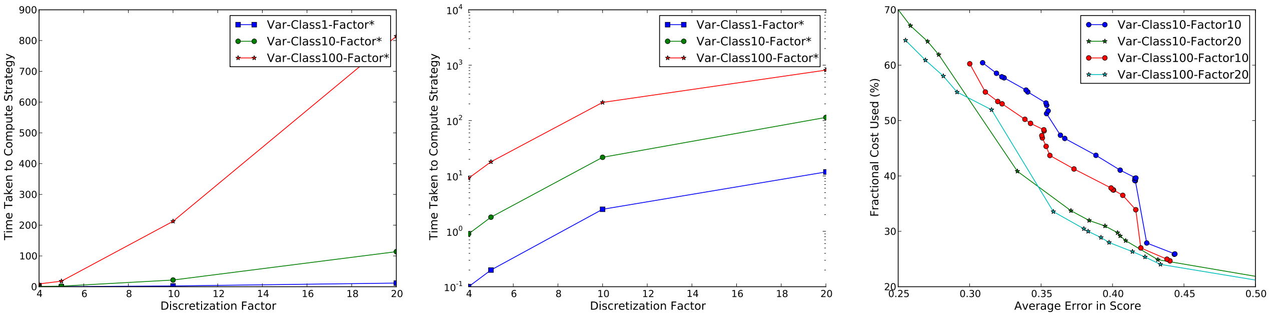

Figure 6: (a) Computational Cost of Varying Class Size (b) Computational Cost of Varying Class Size (c) Relative Impacts of k and δ

7. RELATED WORK

Crowd Algorithms: There is plenty of recent work on designing data processing algorithms that use humans as data processing units [8, 13, 17, 20, 21, 25, 29, 30, 36, 40, 43, 46]. Of these, the only paper focusing on filtering is CrowdScreen [36], which we compare against in this paper. The work by Karger et al. [25] also considers filtering, but instead of optimizing for an explicit objective, tries to minimize worst case error (a different, weaker objective than ours). As a result, their algorithm never obtain any (asymptotic) improvements from using an online scheme that asks questions based on the answers obtained thus far. Furthermore, Karger et al. assume that the same worker will never be asked again. In our case, the same set of students will be asked to answer many questions.

EM-Based Worker Quality Estimation: There are a number of papers that use the Expectation Maximization algorithm to estimate worker quality, and in the process, estimate the true answers for various tasks [18, 22, 23, 34, 35, 39, 44]. These algorithms collect annotations from humans, and does disagreement-based analysis *after the fact* to deduce the true answers. Our algorithms could certainly benefit from using some of these techniques to better assess the quality of the work provided by workers before designing strategies, perhaps combining them with multi-armed bandit schemes [45].

Systems: There are many crowd-powered systems that have been developed over the last few years [11, 12, 16, 19, 27, 28, 31, 33, 37, 38, 47]. Many of these systems require a quality control component, making sure that enough votes are gathered to ensure correctness.

MDPs: Dan Weld's group has used POMDPs (Partially Observable Markov Decision Processes) to design crowd-powered workflows [14, 16, 26, 27]. In particular, they model worker behavior, task difficulty, and output quality to dynamically choose the 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". Kamar et al. [24] use POMDPs to study how to best utilize participation in voluntary crowdsourcing systems, specifically, Galaxy Zoo, an astronomical data set verified by human workers.

Our filtering strategies also use decision theory, specifically, MDPs (Markov Decision Processes); however, unlike the papers listed above, our models are simpler, enabling us to get guarantees for optimality for our filtering strategies, while performing exceptionally well in practice. The papers mentioned above do not provide theoretical guarantees of any kind. Lastly, these papers do not focus on filtering.

8. CONCLUSIONS

In this paper, we described optimal filtering techniques that take into account a number of aspects found in real-world scenarios. We provided extensions of the strategy computation techniques in CrowdScreen [36] that enable us to address all of these aspects, but lead to intractability in the representation and computation of the strategy for some aspects. We then developed the posterior-based representation which does not suffer from the intractability issue in the answer-record representation, but leads to strategies that may not be optimal. We did, however, show that these strategies converge to optimal ones in the limit.

We then demonstrated the use of crowd-powered algorithms in a novel application: peer evaluation in MOOCs. Our algorithms provide significant reductions in both cost and error (as high as 30% savings in cost, and 30% improvement in accuracy) over schemes used in practice and intuitive baselines, as well as simpler CrowdScreen algorithms.

Even with a posterior-based representation, there may be significant computational costs in running the algorithm to derive the optimized strategy, as well as significant costs in representing and storing the optimized strategy. The computational costs are linearly dependent on k (the number of worker partitions based on ability) and polynomially dependent on δ (the strategy discretization factor). We demonstrated that there are significant benefits to increasing the strategy parameters k and δ; we should increase both k and δ as much as the computational capability allows. Since k is likely to be no more than a few hundred or a few thousand in our peer evaluation system, we prefer to increase k first, before δ. Of course, in other systems or other applications, we may wish to increase both k and δ simultaneously.

Acknowledgments

We thank Chinmay Kulkarni and Scott Klemmer for access to the MOOC data, and Akash Das Sarma for initial discussions.

9. REFERENCES

[1] Coursera Inc. (Retrieved 14 August 2013). *http://www.coursera.com.*

[2] CrowdFlower Content Moderation Platform (Retrieved 14 August 2013). *http://crowdflower.com/type-content-moderation.*

[3] EdX (Retrieved 14 August 2013). *http://edx.edu.*

[4] Enlightening Statistics about MOOCS (Retrieved 14 August 2013). *http://www.edtechmagazine.com/higher/article/2013/02/11-enlightening-statistics-about-massive-open-online-courses.*

[5] ODesk (Retrieved 22 July 2013). *http://odesk.com.*

[6] Udacity Inc. (Retrieved 14 August 2013) . *http://www.udacity.edu.*

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

[8] Y. Amsterdamer, Y. Grossman, T. Milo, and P. Senellart. Crowd mining. In *SIGMOD Conference*, pages 241–252, 2013.

[9] E. Bakshy, J. M. Hofman, W. A. Mason, and D. J. Watts. Everyone's an influencer: quantifying influence on twitter. In *WSDM*, pages 65–74, 2011.

[10] M. Barber, K. Donnelly, and S. Rizvi. *An Avalanche is Coming: Higher Education and the Revolution Ahead*. Institute for Public Policy Research, March 2013.

[11] M. S. Bernstein, J. Brandt, R. C. Miller, and D. R. Karger. Crowds in two seconds: enabling realtime crowd-powered interfaces. In *UIST*, pages 33–42, 2011.

[12] M. S. Bernstein, G. Little, R. C. Miller, B. Hartmann, M. S. Ackerman, D. R. Karger, D. Crowell, and K. Panovich. Soylent: a word processor with a crowd inside. In *UIST*, pages 313–322, 2010.

[13] A. Bozzon, M. Brambilla, and S. Ceri. Answering search queries with crowdsearcher. In *WWW*, pages 1009–1018, 2012.

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

[15] K.-T. Chen, C.-C. Wu, Y.-C. Chang, and C.-L. Lei. A crowdsourceable qoe evaluation framework for multimedia content. In *ACM Multimedia*, pages 491–500, 2009.

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

[17] S. B. Davidson, S. Khanna, T. Milo, and S. Roy. Using the crowd for top-k and group-by queries. In *ICDT*, pages 225–236, 2013.

[18] A. P. Dawid and A. M. Skene. Maximum likelihood estimation of observer error-rates using the em algorithm. *Applied Statistics*, 28(1):20–28, 1979.

[19] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. Crowddb: answering queries with crowdsourcing. In *SIGMOD*, pages 61–72, 2011.

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

[21] S. Guo, A. Parameswaran, and H. Garcia-Molina. So Who Won? Dynamic Max Discovery with the Crowd. In *SIGMOD*, pages 385–396, 2012.

[22] J. Whitehill et al. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *NIPS*. 2009.

[23] M. Joglekar, H. Garcia-Molina, and A. Parameswaran. Evaluating the crowd with confidence. In *KDD*, pages 686–694, 2013.

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

[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] G. Little, L. B. Chilton, M. Goldman, and R. C. Miller. Turkit: tools for iterative tasks on mechanical turk. In *HCOMP*, pages 29–30, 2009.

[29] I. Lotosh, T. Milo, and S. Novgorodov. Crowdplanr: Planning made easy with crowd. In *ICDE*, pages 1344–1347, 2013.

[30] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller. Human-powered sorts and joins. In *VLDB*, pages 13–24, 2012.

[31] A. Marcus, E. Wu, S. Madden, and R. Miller. Crowdsourced databases: Query processing with people. In *CIDR*, 2011.

[32] 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.

[33] J. Noronha, E. Hysen, H. Zhang, and K. Z. Gajos. Platemate: crowdsourcing nutritional analysis from food photographs. In *UIST*, pages 1–12, 2011.

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

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

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

[37] A. Parameswaran, M. H. Teh, H. Garcia-Molina, and J. Widom. Datasift: An expressive and accurate crowd-powered search toolkit. In *HCOMP*, 2013.

[38] H. Park, R. Pang, A. Parameswaran, H. Garcia-Molina, N. Polyzotis, and J. Widom. An overview of the deco system: Data model and query language; query processing and optimization. *ACM SIGMOD Record*, 41(4), December 2012.

[39] A. Ramesh, A. Parameswaran, H. Garcia-Molina, and N. Polyzotis. Identifying reliable workers swiftly. Infolab technical report, Stanford University, 2012.

[40] A. D. Sarma, A. Parameswaran, H. Garcia-Molina, and A. Halevy. Finding with the crowd. In *ICDE*, 2014 (To Appear).

[41] R. Snow, B. O'Connor, D. Jurafsky, and A. Y. Ng. Cheap and fast - but is it good? evaluating non-expert annotations for natural language tasks. In *EMNLP*, pages 254–263, 2008.

[42] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. A Bradford Book, 1998.

[43] B. Trushkowsky, T. Kraska, M. J. Franklin, and P. Sarkar. Getting it all from the crowd. *CoRR*, abs/1202.2335, 2012.

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

[45] J. Vermorel and M. Mohri. Multi-armed bandit algorithms and empirical evaluation. In *ECML*, pages 437–448, 2005.

[46] J. Wang, T. Kraska, M. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. In *VLDB*, 2012.

[47] H. Zhang, E. Law, R. Miller, K. Gajos, D. C. Parkes, and E. Horvitz. Human computation tasks with global constraints. In *CHI*, pages 217–226, 2012.