QASCA: A Quality-Aware Task Assignment System for Crowdsourcing Applications

Conference Paper · May 2015

DOI: 10.1145/2723372.2749430

Yudian Zheng
The University of Hong Kong
37 PUBLICATIONS **1,060** CITATIONS
SEE PROFILE

Jiannan Wang
Simon Fraser University
57 PUBLICATIONS **2,245** CITATIONS
SEE PROFILE

Guoliang Li
Tsinghua University
232 PUBLICATIONS **5,873** CITATIONS
SEE PROFILE

Reynold Cheng
The University of Hong Kong
177 PUBLICATIONS **5,958** CITATIONS
SEE PROFILE

QASCA: A Quality-Aware Task Assignment System for Crowdsourcing Applications

Yudian Zheng[†], Jiannan Wang[$], Guoliang Li[#], Reynold Cheng[†], Jianhua Feng[#]
[#]Tsinghua University, [†]The University of Hong Kong, [$]UC Berkeley
ydzheng2@cs.hku.hk, jnwang@berkeley.edu, liguoliang@tsinghua.edu.cn
ckcheng@cs.hku.hk, fengjh@tsinghua.edu.cn

ABSTRACT

A crowdsourcing system, such as the Amazon Mechanical Turk (AMT), provides a platform for a large number of questions to be answered by Internet workers. Such systems have been shown to be useful to solve problems that are difficult for computers, including entity resolution, sentiment analysis, and image recognition. In this paper, we investigate the *online task assignment problem*: Given a pool of n questions, which of the k questions should be assigned to a worker? A poor assignment may not only waste time and money, but may also hurt the quality of a crowdsourcing application that depends on the workers' answers. We propose to consider quality measures (also known as evaluation metrics) that are relevant to an application during the task assignment process. Particularly, we explore how Accuracy and F-score, two widely-used evaluation metrics for crowdsourcing applications, can facilitate task assignment. Since these two metrics assume that the ground truth of a question is known, we study their variants that make use of the probability distributions derived from workers' answers. We further investigate *online assignment strategies*, which enables optimal task assignments. Since these algorithms are expensive, we propose solutions that attain high quality in linear time. We develop a system called the Quality-Aware Task Assignment System for Crowdsourcing Applications (QASCA) on top of AMT. We evaluate our approaches on five real crowdsourcing applications. We find that QASCA is efficient, and attains better result quality (of more than 8% improvement) compared with existing methods.

Categories and Subject Descriptors

H.4.m [**Information Systems Applications**]: Miscellaneous

Keywords

Crowdsourcing; Quality Control; Online Task Assignment

1. INTRODUCTION

Crowdsourcing solutions have been proposed to solve problems that are often considered to be hard for computers (e.g., entity resolution [56,60] and sentiment analysis [30]). Consider the entity resolution problem, where objects in a database referring to the same real-world entity are to be identified. For instance, in a product review database, it is useful to collect all the users' comments about a particular product, which may be named differently by various users (e.g., *iPad Two* and *iPad 2*). To perform this task, crowdsourcing techniques have been developed to generate human understandable questions (e.g., Are *iPad Two* and *iPad 2* the same or not?) for the database owner (or *requester*) [56]. These questions, packed into *Human Intelligent Tasks* (HITs), are posted on a crowdsourcing platform (e.g., Amazon Mechanical Turk (AMT)). Internet users (or *workers*) are then invited to answer these questions, based on which the final result is returned to the requester.

In AMT, every HIT contains a certain number (k) of questions. Because a worker may give an incorrect answer, a HIT is assigned to a number (z) of workers. The result of each question is then derived based on voting strategies (e.g., Majority Vote). Upon completion of a HIT, the requester may pay a certain amount of money to the worker. A fundamental issue, which we call *task assignment*, is: Given a pool of n questions, which of the k questions should be selected and put to the HIT for the coming worker? In AMT, this issue is usually addressed in an *offline* manner: the questions assigned to all HITs were decided before they are shown to the workers. As pointed out in [3,30], the main drawback of this approach is that the difficulty level of a question is not considered: for an "easy" question, its final result can be determined even if the current number of answers received from workers is less than z, whereas a more difficult or controversial question may require answers from more than z workers. Notice that a requester may only have a limited amount of budget to pay the workers. It is thus important to decide the question(s) to be included in a HIT, in order to obtain the best answers under the limited budget. Recent solutions, such as CDAS [30] and AskIt! [3], address this problem through *online assignment strategies* – the HIT is generated *dynamically* when requested by a worker, i.e., the k questions are chosen for the HIT "on the fly". The statistical confidence of the answers obtained so far for each question is tracked, and questions whose answers are the least confident are put to the HIT. Thus, the number of times each question is asked can be different. These methods were shown to perform better than the AMT's approach.

However, existing online assignment strategies overlook an important factor – the applications that use the crowdsourced data. Depending on the application semantics, the metric used to gauge the quality of crowdsourced data can be different. In Twitter Sentiment Analysis, for instance, workers are invited to give their sentiments (e.g., "positive", "neutral" or "negative") for each crawled tweet [30]. The Accuracy metric, which is the fraction of returned tweets correctly classified, is often used to measure the quality of the sentiment labels [18,22,30,44]. As for entity resolution [56,60], which asks a worker to judge whether a pair of objects is "equal" or "non-equal", F-score [25,31,32,56,59] is often adopted to measure the quality of the entity-resolution results. As our experiments

SIGMOD'15, May 31–June 4, 2015, Melbourne, Victoria, Australia.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2758-9/15/05 ...$15.00.
http://dx.doi.org/10.1145/2723372.2749430.

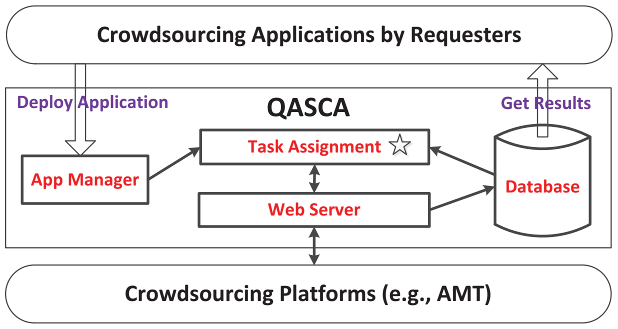


Figure 1: The QASCA Architecture.

show, considering evaluation metrics in the task assignment process can significantly improve the quality of the crowdsourced results.

Our solutions. In this paper, we propose a novel online task assignment framework, which takes application-driven evaluation metrics into account. We investigate two popular evaluation metrics, i.e., Accuracy and F-score, that are widely used by various crowdsourcing applications [18,22,30,44,56,59]. Conceptually, our algorithm enumerates all sets of k questions. For every set of k questions, our solution estimates the improvement in the quality of the answers, *if* these questions are really sent to the coming worker. The set of k questions that maximizes the quality improvement will constitute the new HIT. To realize this, we have to address two key issues:

• **Lack of ground truth.** Evaluation metrics, such as Accuracy and F-score, assume that each question's *ground truth* (or true answer) is known. However, during the task assignment process, it may not be possible to know the ground truth of a question. Thus, existing evaluation metrics are not readily used to solve the task assignment problem. We represent the possible true answer of a question by the *distribution matrix*, which captures the probability distributions of true answers. We study how to populate this matrix with two models, namely Worker Probability (WP) [16,26,30,62] and Confusion Matrix (CM) [1,22,61] that are commonly used to describe the performance of workers. We further incorporate the distribution matrix into Accuracy and F-score, resulting correspondingly in the proposed two functions: Accuracy* and F-score*.

• **Expensive evaluation.** Finding the best solution for the task assignment problem can be extremely expensive. Given a pool of n questions, there are $\binom{n}{k}$ sets of candidate questions for a HIT. Moreover, due to the incorporation of the distribution matrix, measuring the quality of the distribution matrix under Accuracy* or F-score* can be expensive. We explore efficient algorithms for quality measurement. We also propose two respective linear-time algorithms to find the best set of questions for assignment.

We have developed a system called QASCA. As shown in Figure 1, QASCA is run on top of a crowdsourcing platform (e.g., AMT). The *App Manager* stores the n questions and other information (e.g., budget) needed by the strategies. The *Task Assignment* runs the strategies and decides the k questions to be included in the HIT. The *Web Server* then sends the HIT to the workers. The workers' answers are then stored in the *Database* through the *Web Server*, and the derived results are sent back to the requester. The details of QASCA and how an entity resolution application can use QASCA are described in Appendix A.

To summarize, we make the following contributions:

(1) We propose a novel task assignment framework by incorporating evaluation metrics into assignment strategies, and formalize the *online task assignment problem* under the proposed framework;

(2) We generalize the definition of evaluation metrics to be able to quantify the result quality w.r.t a distribution matrix, and devise efficient algorithms to identify the optimal result of each question that can maximize the overall quality;

(3) We propose two respective linear online assignment algorithms that can efficiently select the best k questions for a coming worker;

(4) We develop a system called QASCA[1], which enables a popular crowdsourcing platform (i.e., AMT) to support our task assignment framework. We evaluate the performance of QASCA on five real applications. Experimental results indicate that QASCA can achieve much better (of more than 8% improvement) result quality compared with five state-of-the-art systems.

The remainder of this paper is organized as follows. Section 2 defines the task assignment problem. We define Accuracy* and F-score*, and explain how to evaluate them in Section 3. Efficient online assignment algorithms are devised in Section 4. Section 5 addresses how to compute distribution matrices. We present our experimental results in Section 6. Section 7 discusses related works. Finally, we present conclusions and future work in Section 8.

2. THE TASK ASSIGNMENT PROBLEM

We first discuss the question model in Section 2.1, and then formally define the task assignment problem in Section 2.2. Finally we explain the workflow of QASCA in Section 2.3.

2.1 Question Model

Let $S = \{q_1, q_2, \ldots, q_n\}$ denote the set of questions provided by a requester and each question has the same ℓ possible labels (or answers), denoted by $\{L_1, L_2, \ldots, L_\ell\}$. For example, the two labels for all generated questions in an entity resolution application [59] are $\{L_1$="equal", L_2="non-equal"$\}$. Let $D = \{D_1, D_2, \ldots, D_n\}$ denote the answer set for all questions. Each D_i contains a set of tuples where each tuple (w, j) denotes that question q_i has been answered by worker w with label L_j. For example, $D_2 = \{(w_1, 1), (w_3, 2)\}$ means that question q_2 is answered twice: worker w_1 has answered question q_2 with label L_1 and worker w_3 has answered question q_2 with label L_2.

When worker w completes a HIT, for each question q_i ($1 \leq i \leq n$), we can compute the probability distribution of question q_i's true label. The probability distributions of all questions form the question model, called *current distribution matrix*, denoted by Q^c, which is an $n \times \ell$ matrix. The i-th ($1 \leq i \leq n$) row $Q^c_i = [\, Q^c_{i,1}, Q^c_{i,2}, \ldots, Q^c_{i,\ell} \,]$ represents the probability distribution for question q_i's true label, and each cell $Q^c_{i,j}$ ($1 \leq i \leq n, 1 \leq j \leq \ell$) denotes the probability that question q_i's true label is L_j. We will discuss how to compute Q^c in Section 5.1.

Remarks: For ease of presentation, we assume that (1) the labels are pre-defined, and are the same for all questions; (2) each question's ground truth is a single label. These assumptions can be relaxed. First, if labels are not pre-defined, [50] addresses how to enumerate possible labels for questions. Second, if each question has multiple true labels, we can follow [40] to decompose each question into ℓ filter questions, where each filter question is to decide whether the original question satisfies a corresponding label or not. To handle a domain with continuous values, we can adopt the *bucketing* method [29], which discretizes the domain into different buckets, where each bucket indicates a label.

2.2 Task Assignment

Note that a worker may request multiple HITs, so we keep track of the history of previously assigned questions. Let S^w denote the candidate set of questions for worker w, i.e., the set of questions that have not been assigned to worker w (each worker has her unique S^w). QASCA will not assign duplicated questions to the same worker. When worker w requests a HIT, it selects k questions in S^w and assigns them to her. To select questions for worker

[1]http://i.cs.hku.hk/∼ydzheng2/QASCA

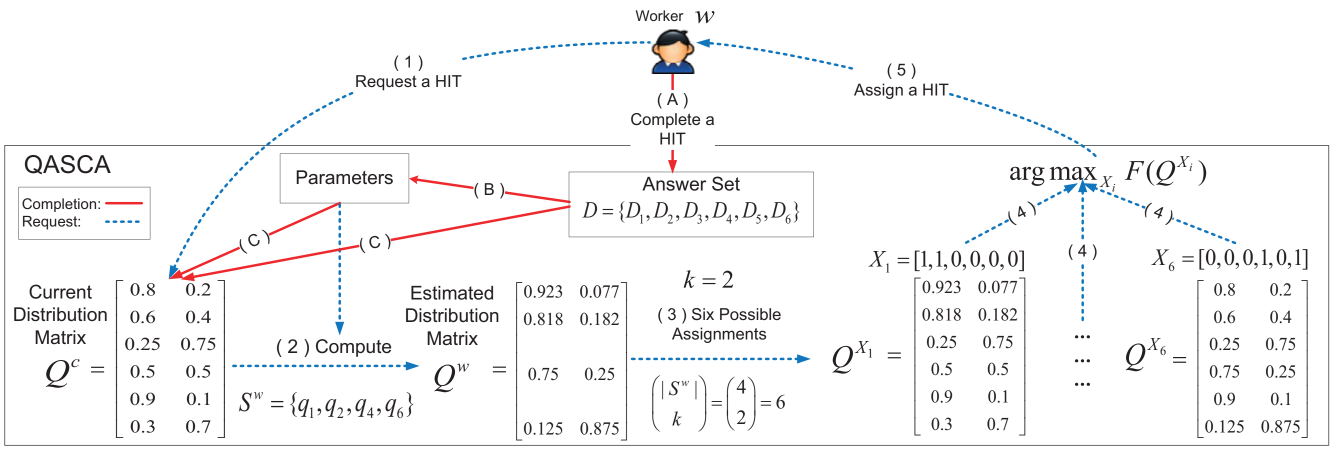


Figure 2: Workflow of QASCA.

w, QASCA first estimates the probability distribution of each question q_i's ($q_i \in S^w$) true label *if* worker w answers it. The estimated probability distributions of all questions in S^w form an $n \times \ell$ matrix Q^w, called *estimated distribution matrix for worker* w. The i-th ($q_i \in S^w$) row $Q^w_i = [\, Q^w_{i,1}, Q^w_{i,2}, \ldots, Q^w_{i,\ell}\,]$ represents the estimated probability distribution for question q_i's true label if it is answered by worker w. Each cell $Q^w_{i,j}$ ($q_i \in S^w, 1 \leq j \leq \ell$) denotes the estimated probability that question q_i's true label is L_j if it is answered by worker w. Note that row i (or Q^w_i) is empty when $q_i \notin S^w$. We will discuss how to compute Q^w in Section 5.3.

Let the vector $X = [x_1, x_2, \ldots, x_n]$ denote an assignment of HIT, where each element $x_i = 1$ (0) indicates that the question q_i will (not) be chosen to assign for the coming worker. When worker w comes, based on S^w, we define X as a *feasible assignment* if it satisfies (1) $\sum_{i=1}^{n} x_i = k$, and (2) if $x_i = 1$, then $q_i \in S^w$. There are k questions in a HIT and we can only assign questions in S^w to worker w. Thus the number of feasible X is $\binom{|S^w|}{k} \leq \binom{n}{k}$.

Given Q^c, Q^w, and a feasible X, we construct a matrix Q^X called *assignment distribution matrix for X*. The i-th row, Q^X_i, is the (estimated) probability distribution of question q_i if worker w answers all the assigned questions in X. We can construct Q^X using Q^c and Q^w: for an unassigned question q_i in X (or $x_i = 0$), its distribution (Q^X_i) remains to be Q^c_i; for an assigned question q_i in X (or $x_i = 1$), its distribution (Q^X_i) is estimated to be Q^w_i, thus

$$Q^X_i = \begin{cases} Q^c_i & \text{if } x_i = 0, \\ Q^w_i & \text{if } x_i = 1. \end{cases} \tag{1}$$

Let $F(\cdot)$ be an evaluation metric, which is used to evaluate the quality of a distribution matrix. When worker w requests a HIT, there are $\binom{|S^w|}{k}$ feasible X, and the problem is to choose the optimal feasible X^* that maximizes $F(Q^X)$. We formally define the **(online) task assignment problem** in Definition 1.

DEFINITION 1. *When a worker w requests a HIT, given the current distribution matrix (Q^c), the estimated distribution matrix for the worker w (Q^w), and the function $F(\cdot)$, the problem of task assignment for the worker w is to find the optimal feasible assignment vector X^* such that $X^* = argmax_X\ F(Q^X)$.*

2.3 Workflow of QASCA

To deploy an application, a requester needs to set n questions with ℓ labels and she should also indicate the number of questions in each HIT (k), the amount of money paid for each HIT (b), the total invested budget (B) and the evaluation metric. In [20,23], the issues of setting appropriate values of k and b are discussed. The evaluation metric (e.g., Accuracy and F-score), which depends on the application semantics, will be addressed in Section 3.

There are two events from workers that QASCA should process: the event when a worker completes a HIT (called "HIT completion") and the event when a worker requests a HIT (called "HIT request"). Based on the workflow in Figure 2, we give an example (Example 1) to show how QASCA processes these two events.

EXAMPLE 1. *The solid (red) lines and the dotted (blue) lines in Figure 2 represent how QASCA processes a HIT completion event and a HIT request event, respectively:*
(1) when a worker w completes a HIT (HIT completion process), QASCA does several updates in the database: it first updates the answer set D [2] *(step A), and then based on the new D, it updates some parameters such as workers' qualities (step B). Finally it uses these new parameters to update Q^c (step C);*
(2) when a worker w requests a HIT (HIT request process), suppose $S = \{q_1, q_2, q_3, q_4, q_5, q_6\}$, and each HIT contains $k = 2$ questions. QASCA first extracts Q^c (step 1) and parameters from database to compute Q^w for worker w (step 2). Assume worker w has answered q_3 and q_5 previously, now the candidate set of questions for her is $S^w = \{q_1, q_2, q_4, q_6\}$. Since $|S^w| = 4$ and $k = 2$, there are $\binom{4}{2} = 6$ feasible assignments for worker w (step 3). Consider the first feasible assignment $X_1 = [1, 1, 0, 0, 0, 0]$, which assigns q_1 and q_2 to worker w. We construct Q^{X_1} by Equation 1, so $Q^{X_1}_i = Q^w_i$ for $i = 1, 2$ and $Q^{X_1}_i = Q^c_i$ for $i = 3, 4, 5, 6$. Similarly, we can construct Q^{X_2}, Q^{X_3}, Q^{X_4}, Q^{X_5}, Q^{X_6} for other feasible assignments. Based on the chosen function $F(\cdot)$ (Accuracy or F-score), assume that $F(Q^{X_i})$ ($1 \leq i \leq 6$) is maximized on Q^{X_6} (step 4) and since $X_6 = [0, 0, 0, 1, 0, 1]$, QASCA batches questions $\{q_4, q_6\}$ in a HIT and assigns it to worker w (step 5).

To help readers better understand our paper, we summarize notations in Appendix B. There are three challenges in Figure 2. Firstly, how can we define $F(\cdot)$ for different evaluation metrics; secondly, given Q^c, Q^w and $F(\cdot)$, how can we efficiently compute the optimal assignment in Definition 1 (step 4); thirdly, how can we compute Q^c when a HIT is completed (step C) and estimate Q^w for worker w when a HIT is requested (step 2). We respectively address these challenges in the following three sections.

3. EVALUATING QUALITY METRICS

In this section, we study the two popular evaluation metrics (Accuracy and F-score) in Sections 3.1 and 3.2, respectively. For each evaluation metric, we first introduce its original definition assuming the known ground truth, then we study its variants by incorporating a distribution matrix Q. Finally we define $F(\cdot)$ by discussing how to evaluate the quality of Q w.r.t. an evaluation metric.

We first clarify some notations. We denote the result vector by $R = [r_1, r_2, \ldots, r_n]$, where $1 \leq r_i \leq \ell$ and L_{r_i} is the returned

[2] Note that the answer set D is continuously updated. That is, in the HIT request process, the current D is used to decide which questions should be assigned; while in the HIT completion process, QASCA updates D based on worker's answers.

label (or result) for q_i. We also denote the ground truth vector by $T = [t_1, t_2, \ldots, t_n]$, where $1 \le t_i \le \ell$ and L_{t_i} is the ground truth label for q_i. The formal definition of function Accuracy or F-score is $F(T, R)$, which evaluates the quality of R based on known T. In task assignment scenarios, the ground truth vector T is unknown, but we can obtain a distribution matrix Q based on the crowd's answers. Thus, we generalize the evaluation metrics to be able to quantify the quality of result vector R (called "result quality") w.r.t the distribution matrix Q, i.e., $F^*(Q, R)$. Since given a distribution matrix Q, the requesters want the best results $R^* = \arg\max_R F^*(Q, R)$ to be returned. So in order to evaluate the quality of Q, we consider the choice of R^* and use the best quality that Q can reach to evaluate the quality of Q, i.e., $F(Q) = \max_R F^*(Q, R) = F^*(Q, R^*)$.

3.1 Accuracy

Accuracy is an evaluation metric used by many crowdsourcing applications [8,21,22,30,44]. It aims to measure the overall classification quality among all labels. For example, in a sentiment analysis application, if a requester focuses on the overall quality among all three labels (i.e., "positive", "neutral" and "negative"), Accuracy can be used as the evaluation metric. It is defined as the fraction of returned labels that are correct. Let $\mathbb{1}_{\{\cdot\}}$ denote an indicator function which returns 1 if its argument is true; 0, otherwise. For example, $\mathbb{1}_{\{5=2\}} = 0$ and $\mathbb{1}_{\{5=5\}} = 1$. Then we derive

$$\text{Accuracy}(T, R) = \frac{\sum_{i=1}^{n} \mathbb{1}_{\{t_i = r_i\}}}{n}. \quad (2)$$

Consider an example where $n = 4$, $\ell = 3$, $T = [2, 1, 3, 2]$ and $R = [2, 1, 3, 1]$. Since it correctly identifies the labels of the 1st, 2nd, and 3rd questions, we have $\text{Accuracy}(T, R) = \frac{3}{4} = 0.75$.

*3.1.1 Accuracy**

As shown above, the definition of $\text{Accuracy}(T, R)$ requires to know the ground truth T. In practice, however, we only have the distribution matrix Q, which records the probability distribution of each question's true label. Therefore, we use the expected accuracy to measure the result quality. Since Q_i (the i-th row of Q) represents the probability distribution of question q_i's true label, then we have $P(t_i = j) = Q_{i,j}$ and $\mathbb{E}[\mathbb{1}_{\{t_i=j\}}] = P(t_i = j) \cdot 1 + P(t_i \ne j) \cdot 0 = Q_{i,j}$. Thus $\text{Accuracy}^*(Q, R)$ is defined as

$$\text{Accuracy}^*(Q, R) = \mathbb{E}[\, \text{Accuracy}(T, R)\,] = \frac{\sum_{i=1}^{n} Q_{i,r_i}}{n}. \quad (3)$$

Specifically, $\text{Accuracy}^*(Q, R)$ represents the *expected* number of correctly answered questions out of all questions. For example, consider the distribution matrix Q^c in Figure 2. Given a result vector of $R = [1, 2, 2, 1, 1, 1]$, its accuracy (w.r.t Q^c) is defined as $\text{Accuracy}^*(Q^c, R) = \frac{0.8+0.4+0.75+0.5+0.9+0.3}{6} = 60.83\%$.

*3.1.2 Identify the Optimal Result for Accuracy**

In order to measure the quality of a distribution matrix Q, we need to determine the optimal result vector R^* that maximizes $\text{Accuracy}^*(Q, R)$, i.e., $R^* = \text{argmax}_R\ \text{Accuracy}^*(Q, R)$. To compute R^*, an intuitive idea is to return the most likely label for each question, i.e., $R^* = [r_1^*, r_2^*, \ldots, r_n^*]$ where $r_i^* = \text{argmax}_j\ Q_{i,j}$. We next prove that the idea is correct in Theorem 1.

THEOREM 1. *For Accuracy*, the optimal result r_i^* ($1 \le i \le n$) of a question q_i is the label with the highest probability, i.e., $r_i^* = \arg\max_j Q_{i,j}$.*

Due to the space limits, we move all the proofs in our paper to the Appendix. Based on Theorem 1, we know that for Accuracy*, the optimal result of a question q_i only depends on its own distribution Q_i. Take Q^c in Figure 2 as an example. Consider the first question. Since $Q^c_{1,1}$ (0.8) $>$ $Q^c_{1,2}$ (0.2), the optimal result for q_1 is $r_1^* = 1$. Similarly we can derive the optimal result vector R^* for all questions, i.e., $R^* = [1, 1, 2, 1, 1, 2]$ (or $[1, 1, 2, 2, 1, 2]$ as $Q^c_{4,1} = Q^c_{4,2}$). Thus the quality of Q^c for the metric Accuracy, is measured as $F(Q^c) = \text{Accuracy}^*(Q^c, R^*) = 70.83\%$

For Accuracy, the definition (Equation 3) and the optimal result selection (Theorem 1) are not very difficult. But for F-score, these problems become more challenging.

3.2 F-score

In applications such as text classification [7], sentiment analysis [31,32], entity resolution [56,59,60] and fact finding [42,63], a requester may only be interested in a particular label (which we call *target label*). In this situation, a question can be simplified as a two-label question (i.e., $\ell = 2$), where the two labels are the "target label" and "non-target label", denoted by L_1 and L_2, respectively. For example, in an sentiment analysis application, if the requester wants to pick out the "positive" sentiment tweets with a high confidence, then each question is to identify whether a sentence's sentiment is L_1="positive" or L_2="non-positive".

The F-score, introduced in [51], can be used to capture the quality of answers to the two-label questions above. This measure (F-score) is formally defined as the weighted harmonic mean of two commonly known metrics, namely Precision and Recall:

$$\text{F-score} = \frac{1}{\alpha \cdot \frac{1}{\text{Precision}} + (1-\alpha) \cdot \frac{1}{\text{Recall}}}, \quad (4)$$

where
(1) **Precision** is the faction of questions with returned results L_1 that are actually correct, i.e.,

$$\text{Precision}(T, R) = \frac{\sum_{i=1}^{n} \mathbb{1}_{\{t_i=1\}} \cdot \mathbb{1}_{\{r_i=1\}}}{\sum_{i=1}^{n} \mathbb{1}_{\{r_i=1\}}}; \quad (5)$$

(2) **Recall** is the fraction of questions with ground truth L_1 that are actually returned with results L_1, i.e.,

$$\text{Recall}(T, R) = \frac{\sum_{i=1}^{n} \mathbb{1}_{\{t_i=1\}} \cdot \mathbb{1}_{\{r_i=1\}}}{\sum_{i=1}^{n} \mathbb{1}_{\{t_i=1\}}}; \quad (6)$$

(3) $\alpha \in (0, 1)$ is a parameter that controls the degree of emphasis: $\alpha \in (\frac{1}{2}, 1)$ emphasizes Precision; $\alpha \in (0, \frac{1}{2})$ emphasizes Recall; and we call the *balanced F-score* if $\alpha = \frac{1}{2}$.

By plugging Equations 5 and 6 into Equation 4, we obtain:

$$\text{F-score}(T, R, \alpha) = \frac{\sum_{i=1}^{n} \mathbb{1}_{\{t_i=1\}} \cdot \mathbb{1}_{\{r_i=1\}}}{\sum_{i=1}^{n}[\, \alpha \cdot \mathbb{1}_{\{r_i=1\}} + (1-\alpha) \cdot \mathbb{1}_{\{t_i=1\}}]}. \quad (7)$$

For the parameter α, if a requester wants to select "positive" sentiment sentences with a high confidence, she can set α to a large value (e.g., 0.75) which emphasizes Precision more. On the other hand, if a company manager wants to collect as many "positive" comments for their products as possible, she can give more attention to Recall by setting α to a lower value (e.g., 0.25).

3.2.1 Challenges

As the numerator and denominator of $\text{F-score}(T, R, \alpha)$ both have the random variable $\mathbb{1}_{\{t_i=1\}}$, its expectation cannot be computed as easily as for Accuracy (Equation 3), so we resort to another way of computing its expectation. We denote τ as the set of all possible ground truth vectors, i.e., $\tau = \{1, 2\}^n$ and $|\tau| = 2^n$. Given Q, the probability that $T' = [t'_1, t'_2, \ldots, t'_n] \in \tau$ is the ground truth vector is $\prod_{i=1}^{n} Q_{i,t'_i}$. Then, we have

$$\mathbb{E}[\, \text{F-score}(T, R, \alpha)\,] = \sum_{T' \in \tau} \text{F-score}(T', R, \alpha) \cdot \prod_{i=1}^{n} Q_{i,t'_i}. \quad (8)$$

There are two challenges related to this equation:

One is how to efficiently compute or estimate Equation 8. It is computationally expensive to enumerate all 2^n possible T'. But given α and R, for different $T' \in \tau$, the numerator and denominator of F-score(T', R, α) can only take at most $n+1$ possible values respectively, thus F-score(T', R, α) can only have at most $(n+1)^2$ possible values. Based on this observation, for a given Q and α, Equation 8 can be accurately calculated in $\mathcal{O}(n^3)$ time [24]. Nevertheless, this complexity is still very high to a task assignment system as it is a key step in task assignment. Thus it is better to find an accurate approximation that is efficient to compute.

Another challenge is that if we directly apply Equation 8 to derive the optimal result for each question, we observe two interesting facts that are different from Accuracy:

Observation 1: Returning the label with the highest probability in each question may not be optimal (even for $\alpha = 0.5$);

Observation 2: Deriving the optimal result of a question q_i does not only depend on the question's distribution (or Q_i) itself.

In order to verify these two observations, we next give two counter-examples in Example 2.

EXAMPLE 2. *Consider* $\alpha = 0.5$ *and* $Q = \begin{bmatrix} 0.35 & 0.65 \\ 0.55 & 0.45 \end{bmatrix}$. *If we return the label with the highest probability for each question, then we obtain a result vector* $\widetilde{R} = [2, 1]$ *and the corresponding* $\mathbb{E}[\text{ F-score}(T, \widetilde{R}, \alpha)\ ] = \frac{1}{0.5 \cdot 1 + 0.5 \cdot 2} \cdot 0.35 \cdot 0.55 + 0 \cdot 0.35 \cdot 0.45 + \frac{1}{0.5 \cdot 1 + 0.5 \cdot 1} \cdot 0.65 \cdot 0.55 + 0 \cdot 0.65 \cdot 0.45 = 48.58\%$ *(Equation 8). However, by enumerating all possible* $R \in \{1, 2\}^2$, *we can derive the optimal* $R^* = [1, 1]$ *and* $\mathbb{E}[\text{ F-score}(T, R^*, \alpha)\ ] = 53.58\%$. *This example verifies our first observation.*

Consider $\widehat{\alpha} = 0.5$ *and* $\widehat{Q} = \begin{bmatrix} 0.35 & 0.65 \\ 0.9 & 0.1 \end{bmatrix}$. *Using the same method as above, we can obtain the optimal* $\widehat{R}^* = [2, 1]$. *Compared with the above example, we can see that for the same* $\alpha = \widehat{\alpha} = 0.5$ *in F-score, even if* Q_1 *and* $\widehat{Q}_1$ *are the same (i.e.,* $[0.35, 0.65]$*), the computed* $r_1^* = 1$ *and* $\hat{r}_1^* = 2$ *are different. This example shows that the optimal result for each question is not only dependent on the question's distribution itself.*

To address the above two challenges, we first give an approximation for $\mathbb{E}[\text{ F-score}(T, R, \alpha)\ ]$ in Section 3.2.2 and then discuss how to compute R^* for the approximated function in Section 3.2.3.

3.2.2 F-score*

Following Equation 7, we approximate $\mathbb{E}[\text{ F-score}(T, R, \alpha)\ ]$ as $\frac{\mathbb{E}[\ \sum_{i=1}^{n} \mathbb{1}_{\{t_i=1\}} \cdot \mathbb{1}_{\{r_i=1\}}\ ]}{\mathbb{E}[\ \sum_{i=1}^{n} [\ \alpha \cdot \mathbb{1}_{\{r_i=1\}} + (1-\alpha) \cdot \mathbb{1}_{\{t_i=1\}}]\ ]}$ to boost efficiency, which is the ratio of the expectation of numerator and denominator of F-score(T, R, α). By plugging Q inside, we can formally define F-score$^*(Q, R, \alpha)$ as:

$$\text{F-score}^*(Q, R, \alpha) = \frac{\sum_{i=1}^{n} Q_{i,1} \cdot \mathbb{1}_{\{r_i=1\}}}{\sum_{i=1}^{n} [\ \alpha \cdot \mathbb{1}_{\{r_i=1\}} + (1-\alpha) \cdot Q_{i,1}\ ]}. \quad (9)$$

For example, consider the values of $\widehat{\alpha}$, $\widehat{Q}$ and $\widehat{R}^*$ in Example 2. Based on Equations 8 and 9, we can obtain $\mathbb{E}[\text{ F-score}(T, \widehat{R}^*, \widehat{\alpha})\ ] = 79.5\%$ and F-score$^*(\widehat{Q}, \widehat{R}^*, \widehat{\alpha}) = \frac{0.9}{0.5 \cdot 1 + 0.5 \cdot (0.35 + 0.9)} = 80\%$, respectively. The error in the example is 0.5%.

Approximation Error. Let A and B denote two random variables. The same approximation ($\mathbb{E}\left[\frac{A}{B}\right] \approx \frac{\mathbb{E}[A]}{\mathbb{E}[B]}$) has also been used by other works [4,28,39,45] for efficiency's sake. Furthermore, [45] gives a general formula by expanding Taylor series: $\mathbb{E}\left[\frac{A}{B}\right] = \frac{\mathbb{E}[A]}{\mathbb{E}[B]} + \sum_{v=1}^{\infty} \phi_v$, where $\phi_v = (-1)^v \cdot \frac{\mathbb{E}[A] \cdot <^v B> + <A, ^v B>}{(\mathbb{E}[B])^{v+1}}$, $<^v B> = \mathbb{E}[\ (B - \mathbb{E}[B])^v\ ]$, and $< A, ^v B > = \mathbb{E}[\ (A - \mathbb{E}[A]) \cdot (B - \mathbb{E}[B])^v\ ]$. The standard approximation formula (derived by delta method) used in [4,39] is $\mathbb{E}\left[\frac{A}{B}\right] = \frac{\mathbb{E}[A]}{\mathbb{E}[B]} + \mathcal{O}(n^{-1})$ for $n \to \infty$. By setting A and B as the numerator and denominator of F-score(T, R, α), we can generally derive $\mathbb{E}[\text{ F-score}(T, R, \alpha)\ ] =$ F-score$^*(Q, R, \alpha) + \mathcal{O}(n^{-1})$. We also verify this in experiments (Section 6.1.2) and show that the error is small ($\leq 0.01\%$) when $n \geq 10^3$, which means that our approximation is reasonable.

3.2.3 Identify the Optimal Result for F-score*

Unlike Accuracy*, to identify the optimal result for F-score*, the method that chooses the most possible label of each question is not correct anymore. However, the intuition that it is preferable to return a label with a large probability value may still hold. Based on this idea, we conjecture that there exists a threshold w.r.t each question that can determine whether a target label should be returned or not. That is, a target label should be returned only when its probability is not lower than the threshold; otherwise the non-target label is returned. More formally, let θ_i be the threshold w.r.t a question q_i. If $Q_{i,1} \geq \theta_i$, we return the target label (i.e., $r_i^* = 1$); otherwise, the non-target label is returned (i.e., $r_i^* = 2$). We prove the correctness of the conjecture in Theorem 2. An interesting observation from the theorem is that every question has the same threshold, which is equal to $\lambda^* \cdot \alpha$, where λ^* represents the optimal value for F-score* (or the quality evaluation of Q), i.e., $\lambda^* = \max_R$ F-score$^*(Q, R, \alpha)$.

THEOREM 2. *Given Q and* α, *for F-score**, *the optimal result* r_i^* $(1 \leq i \leq n)$ *of a question* q_i *can be derived by comparing* $Q_{i,1}$ *with the threshold* $\theta = \lambda^* \cdot \alpha$, *i.e.,* $r_i^* = 1$ *if* $Q_{i,1} \geq \theta$ *and* $r_i^* = 2$ *if* $Q_{i,1} < \theta$.

As $\lambda^* = \max_R$ F-score$^*(Q, R, \alpha)$, Theorem 2 shows that for F-score*, a question's optimal result is related to the optimal value for F-score* w.r.t. Q, which takes all the questions' distributions into consideration. This explains that why the claim for Accuracy* (Theorem 1), i.e., "a question's optimal result is only dependent on its own distribution" does not hold. Next we focus on the problem of how to efficiently derive λ^*.

We first show that this problem can be reduced to a 0-1 fractional programming (FP) problem, and then present an iterative algorithm to identify the optimal value (λ^*). Let $B = [b_1, b_2, \ldots, b_n]$, $D = [d_1, d_2, \ldots, d_n]$ denote two coefficient vectors, and β, γ denote two scalars. The 0-1 FP problem is defined as:

$$\max \quad f(\mathbf{z}) = \frac{\sum_{i=1}^{n} (z_i \cdot b_i) + \beta}{\sum_{i=1}^{n} (z_i \cdot d_i) + \gamma} \quad s.t. \quad \mathbf{z} \in \Omega \subseteq \{0, 1\}^n$$

Comparing F-score* (Equation 9) with $f(\mathbf{z})$, we find that F-score* can actually be rewritten in the form of $f(\mathbf{z})$ as follows:

$$\begin{cases} z_i = \mathbb{1}_{\{r_i=1\}},\ b_i = Q_{i,1},\ d_i = \alpha \ \text{ for } 1 \leq i \leq n; \\ \beta = 0,\ \gamma = \sum_{i=1}^{n} (1-\alpha) \cdot Q_{i,1},\ \ \Omega = \{0, 1\}^n. \end{cases} \quad (10)$$

Thus the problem of computing λ^* can be reduced to a 0-1 FP problem. The 0-1 FP problem can be efficiently solved based on the Dinkelbach framework [12]. We apply it to our problem, which consists of the following three steps:

<u>*Initialization:*</u> It first constructs a new function $g(\mathbf{z}, \lambda) = \sum_{i=1}^{n} (b_i - \lambda \cdot d_i) \cdot z_i$. Then it will iteratively update λ.

In our case, b_i, d_i and z_i $(1 \leq i \leq n)$ can be represented following Equation 10 and $g(\mathbf{z}, \lambda) = \sum_{i=1}^{n} (Q_{i,1} - \lambda \cdot \alpha) \cdot \mathbb{1}_{\{r_i=1\}}$.

<u>*Iteration:*</u> Let λ_t denote the λ for the t-th iteration. For the first iteration, the algorithm initializes λ as a constant value λ_{init} (for example, 0): $\lambda_1 = \lambda_{init}$, and then computes $\mathbf{z}' = \arg\max_{\mathbf{z}} g(\mathbf{z}, \lambda_1)$. By plugging the newly computed $\mathbf{z}'$ into $f(\mathbf{z})$, the algorithm updates λ_1 to $\lambda_2 = f(\mathbf{z}')$. Then it repeats the above iteration with the updated λ (i.e., λ_2).

In our case, in the first iteration, with initial $\lambda_1 = \lambda_{init}$, we have to compute $R' = \arg\max_R \sum_{i=1}^{n} (Q_{i,1} - \lambda_1 \cdot \alpha) \cdot \mathbb{1}_{\{r_i=1\}}$, and $R' = [r_1', r_2', \ldots, r_n']$ can be derived by setting

$$\mathbb{1}_{\{r_i'=1\}} = \begin{cases} 1 & \text{(i.e., } r_i' = 1) \ \text{ if } \ Q_{i,1} \geq \lambda_1 \cdot \alpha, \\ 0 & \text{(i.e., } r_i' = 2) \ \text{ if } \ Q_{i,1} < \lambda_1 \cdot \alpha. \end{cases}$$

Then λ_1 is updated as $\lambda_2 = \text{F-score}^*(Q, R', \alpha)$, and we repeat the above iteration with the updated λ_2.

Termination: The algorithm terminates at the c-th iteration when λ converges (or it is unchanged), i.e., $\lambda_{c+1} = \lambda_c$, and it returns λ_{c+1} which is computed for the c-th iteration.

The Dinkelbach framework guarantees that the iterative process will finally converge to the optimal objective value [12]. Thus our algorithm can return the optimal value for F-score*, i.e., $\lambda^* = \lambda_{c+1}$. The detailed algorithm is in Appendix E (Algorithm 1). We next run the two counter-examples demonstrated in Example 2, and analyze the time complexity in the end.

EXAMPLE 3. *Consider $\widehat{\alpha}$ and $\widehat{Q}$ in Example 2. In the 1st iteration, suppose $\lambda_1 = \lambda_{init} = 0$. As $\widehat{Q}_{1,1}$ and $\widehat{Q}_{2,1}$ are both $\geq \lambda_1 \cdot \widehat{\alpha} = 0$, then $R' = [1,1]$ and λ_1 is updated to $\lambda_2 = \text{F-score}(\widehat{Q}, R', \widehat{\alpha}) = \frac{0.35+0.9}{0.5\cdot 2+0.5\cdot(0.35+0.9)} = 0.77$. As $\lambda_2 \neq \lambda_1$, we continue. In the 2nd iteration, by comparing $\widehat{Q}_{1,1}$ and $\widehat{Q}_{2,1}$ with $\lambda_2 \cdot \widehat{\alpha} = 0.385$, we can construct $R' = [2,1]$, and $\lambda_3 = \text{F-score}(\widehat{Q}, R', \widehat{\alpha}) = \frac{0.9}{0.5\cdot 1+0.5\cdot(0.35+0.9)} = 0.8 \neq \lambda_2$. In the 3rd iteration, as the updated $\lambda_3 \cdot \widehat{\alpha} = 0.4$, we can construct $R' = [2,1]$. Since $\lambda_4 = \lambda_3 = 0.8$, it converges and $\widehat{\lambda}^* = 0.8$ is returned. Following Theorem 2, we can obtain the threshold $\widehat{\theta} = \widehat{\lambda}^* \cdot \widehat{\alpha} = 0.4$. Since $\widehat{Q}_{1,1} < \theta$ and $\widehat{Q}_{2,1} \geq \theta$, we have $\widehat{R}^* = [2,1]$.*

If we consider α and Q in Example 2, then similarly we derive $\lambda^* = 0.62$, $\theta = \lambda^* \cdot \alpha = 0.31$, and $R^* = [1,1]$. The above two examples show that the approximation function F-score$^*(\cdot)$ also conforms to the two observations verified in Example 2. Moreover, F-score$^*(\cdot)$ gives us an intuitive explanation of why two observations occur: in the two examples, the optimal values of F-score* (λ^* and $\widehat{\lambda}^*$) affect the individual threshold (θ and $\widehat{\theta}$), and thus affecting the optimal result vectors (especially r_1^* and $\hat{r}_1^*$).

Time Complexity. As each iteration requires $\mathcal{O}(n)$ time and there are c iterations in total, the time complexity is $\mathcal{O}(c \cdot n)$. To evaluate the time complexity in practice, we conduct extensive simulated experiments by randomly generating Q and $\alpha \in [0,1]$, which shows that the time complexity of the algorithm linearly increases w.r.t. n. It converges very fast, and $c \leq 15$ when $n = 2000$ (Section 6.1.2).

4. ONLINE ASSIGNMENT ALGORITHMS

Recall Definition 1, the quality of a given distribution matrix Q is measured as its maximal quality (or quality w.r.t R^*), i.e., $F(Q) = \max_R F^*(Q,R) = F^*(Q,R^*)$, where F can be Accuracy or F-score. To address the task assignment problem, one simple solution is to enumerate all feasible assignment vectors. For each one (X), Q^X can be constructed via Equation 1, and we compute the optimal result vector for Q^X, denoted as $R^X = \arg\max_R F^*(Q^X, R)$. Finally $X^* = \arg\max_X F(Q^X, R^X)$.

Obviously, this simple method is very expensive. To avoid enumerating $\binom{|S^w|}{k}$ feasible assignments, we propose two efficient algorithms: a Top-k Benefit Algorithm for Accuracy* (Section 4.1) and an Online Assignment Algorithm for F-score* (Section 4.2).

4.1 Accuracy*: Top-K Benefit Algorithm

Given Q^c and Q^w, let $R^c = [r_1^c, r_2^c, \ldots, r_n^c]$ and $R^w = [r_1^w, r_2^w, \ldots, r_n^w]$ denote their respective optimal result vectors, i.e., $r_i^c = \arg\max_j Q_{i,j}^c$ for $1 \leq i \leq n$ and $r_i^w = \arg\max_j Q_{i,j}^w$ for $q_i \in S^w$. As shown in Theorem 1, the choice of respective optimal result r_i^c (r_i^w) only depends on Q_i^c (Q_i^w). Therefore, based on the definition of Q^X (Equation 1), the optimal result vector for Q^X, denoted by $R^X = [r_1^X, r_2^X, \ldots, r_n^X]$, can be represented using R^c and R^w as follows:

$$r_i^X = \begin{cases} r_i^c & \text{if } x_i = 0, \\ r_i^w & \text{if } x_i = 1. \end{cases} \tag{11}$$

According to Definition 1, we aim to find the optimal feasible assignment X such that $\text{Accuracy}^*(Q^X, R^X) = \sum_{i=1}^n Q_{i,r_i^X}/n = \sum_{i=1}^n [\,(Q_{i,r_i^c}/n)\cdot \mathbb{1}_{\{x_i=0\}} + (Q_{i,r_i^w}/n)\cdot \mathbb{1}_{\{x_i=1\}}\,]$ is maximized. We can further derive $\text{Accuracy}^*(Q^X, R^X) =$

$$\sum_{i=1}^n \frac{Q_{i,r_i^c}^c}{n} + \sum_{i=1}^n \frac{Q_{i,r_i^w}^w - Q_{i,r_i^c}^c}{n} \cdot \mathbb{1}_{\{x_i=1\}}. \tag{12}$$

As for each X, the value $\sum_{i=1}^n Q_{i,r_i^c}^c/n$ is fixed. Then for each question q_i, we can define the benefit of assigning it to worker w as *Benefit*$(q_i) = Q_{i,r_i^w}^w - Q_{i,r_i^c}^c$, which indicates that the function Accuracy* will be increased by *Benefit*$(q_i)/n$ if q_i is assigned to worker w. Therefore, the optimal assignment consists of k questions with the largest benefits, and selecting them needs $\mathcal{O}(|S^w|) = \mathcal{O}(n)$ time[3]. Example 4 illustrates how to apply the *Top-k Benefit Algorithm* to assign questions in Figure 2 when the function is set as Accuracy*.

EXAMPLE 4. *Consider Q^c and Q^w in Figure 2. We can obtain $R^c = [1,1,2,1,1,2]$ (or $[1,1,2,2,1,2]$) and $R^w = [1,1,0,1,0,2]$.[4] For each $q_i \in S^w$, we compute its benefit as follows: Benefit$(q_1) = Q_{1,r_1^w}^w - Q_{1,r_1^c}^c = 0.123$, Benefit$(q_2) = 0.212$, Benefit$(q_4) = 0.25$ and Benefit$(q_6) = 0.175$. So q_2 and q_4 which have the highest benefits will be assigned to worker w.*

4.2 F-score*: Online Assignment Algorithm

Compared with Accuracy*, the online assignment for F-score* is more challenging. The main reason is that as shown in Section 3.2.3, based on F-score*, the optimal result for each question is not only dependent on its own distribution. Given a feasible X (Q^X can be constructed), deriving the optimal result vector R^X for Q^X is not as straightforward as Equation 11 for Accuracy*.

Next we show how to efficiently solve the task assignment problem for F-score*. Recall that X^* denotes the optimal assignment and let δ^* denote the optimal objective value, i.e.,

$$\delta^* = \max_R \text{ F-score}^*(Q^{X^*}, R, \alpha). \tag{13}$$

The basic idea of our solution is to first initialize $\delta_{init} \leq \delta^*$ (say $\delta_{init} = 0$), and then iteratively update the initial δ_{init} to δ^* until convergence. Since δ^* is unknown, the main problem is how to ensure that δ_{init} is always *increasing* until it reaches δ^*. More formally, let δ_t denote the δ at the t-th iteration. Given δ_t, the updated δ_{t+1} should satisfy the following two properties:

Property 1: if $\delta_t < \delta^*$, then the updated δ_{t+1} should satisfy $\delta_t < \delta_{t+1} \leq \delta^*$;

Property 2: if $\delta_t = \delta^*$, then the updated δ_{t+1} should satisfy $\delta_t = \delta_{t+1} = \delta^*$.

Intuitively, Property 1 guarantees that starting from $\delta_{init} < \delta^*$, δ_{init} will be iteratively increased until δ^*. Property 2 guarantees that at convergence ($\delta_t = \delta_{t+1}$), we can get $\delta_{t+1} = \delta^*$. There are two challenges in solving the problem:
(1) Given δ_t, how can we construct δ_{t+1} such that the above two properties hold?
(2) The update should be solved efficiently to satisfy the performance requirement for task assignment.

To address the first challenge, we present our designed update in Definition 2 as follows:

DEFINITION 2 (UPDATE). *Given δ_t, Q^c, Q^w, α, and S^w, the update from δ_t to δ_{t+1} is defined as*

$$\delta_{t+1} = \max_X \textit{F-score}^*(Q^X, \widehat{R}^X, \alpha), \tag{14}$$

[3]The problem that finds the top k elements in an array can be solved linearly using the PICK algorithm [2].

[4]Note that for $q_i \notin S^w$, Q_i^w does not need to be computed, so we set $r_i^w = 0$ for $q_i \notin S^w$ and it will never be used.

where for each feasible X (Q^X can be constructed via Equation 1), $\widehat{R}^X = [\widehat{r}_1^X, \widehat{r}_2^X, \ldots, \widehat{r}_n^X]$ is constructed based on δ_t, i.e.,

$$\widehat{r}_i^X = \begin{cases} 1 & if \;\; Q_{i,1}^X \geq \delta_t \cdot \alpha, \\ 2 & if \;\; Q_{i,1}^X < \delta_t \cdot \alpha. \end{cases} \tag{15}$$

To help understand the meaning of δ_{t+1} in Definition 2, we will present a brute-force method to obtain δ_{t+1}. This method enumerates all possible feasible assignment vectors. For each feasible X, as Q^X and $\widehat{R}^X$ can be constructed following Equation 1 and 15 respectively, then F-score$^*(Q^X, \widehat{R}^X, \alpha)$ can be computed. By comparing the computed values for all assignment vectors, we can obtain the maximum value, i.e., δ_{t+1}. Theorem 3 formally proves that the updated δ_{t+1} following Definition 2 satisfies the two properties.

THEOREM 3. *The defined δ_{t+1} (in Definition 2) satisfies Property 1 and Property 2.*

To address the second challenge above, as shown in Theorem 4, we find that this problem (computing δ_{t+1} in Definition 2) can actually be reduced to a 0-1 FP problem and efficiently solved by leveraging the Dinkelbach framework (similar to Section 3.2.3).

THEOREM 4. *The problem of computing δ_{t+1} (Definition 2) can be reduced to a 0-1 FP problem.*

Our designed algorithm, called *F-score Online Assignment Algorithm*, computes the optimal assignment. In each iteration, it calls the *Update Algorithm*, which leverages the Dinkelbach framework to compute the updated δ_{t+1}. Note that the Dinkelbach framework not only returns the updated δ_{t+1}, but also derives the corresponding assignment X' such that δ_{t+1} = F-score$^*(Q^{X'}, \widehat{R}^{X'}, \alpha)$. Thus, at the time that δ converges, the δ^* and its corresponding optimal assignment X^* are both returned by the *Update Algorithm*. Appendix H gives the pseudo-codes of *F-score Online Assignment Algorithm* (Algorithm 2) and *Update Algorithm* (Algorithm 3).

Time Complexity: We analyze the time complexity of *F-score Online Assignment Algorithm*. It is an iterative algorithm, where it runs *Update Algorithm* in each iteration. Following Dinkelbach framework, *Update Algorithm* adopts an iterative approach and takes $\mathcal{O}(n)$ in each iteration. Suppose *F-score Online Assignment Algorithm* requires u iterations to converge and *Update Algorithm* requires v iterations to converge, then the total time complexity is $\mathcal{O}(u \cdot v \cdot n)$. We conduct extensive simulated experiments by randomly generating Q^c and Q^w ($n = 2000$), and varying $\alpha \in [0, 1]$, which shows that the bound $u \cdot v <= 10$ in practice (Section 6.1.3).

EXAMPLE 5. *Given S^w, k, Q^c and Q^w in Figure 2, if the evaluation metric is set as F-score* with $\alpha = 0.75$, we next derive the optimal assignment for worker w. With an initial $\delta_1 = \delta_{init} = 0$, we need to get δ_2. The brute-force way[5] is to enumerate all 6 feasible assignments, where for the first $X = [1, 1, 0, 0, 0, 0]$, we construct Q^X. As $Q_{i,1}^X \geq \delta_1 \cdot \alpha = 0$ for all $1 \leq i \leq n$, thus $\widehat{R}^X = [1, 1, 1, 1, 1]$ and F-score$^*(Q^X, \widehat{R}^X, \alpha) = 0.68$. By considering all 6 feasible X, we derive the maximal F-score* value, i.e., 0.7, which corresponds to $X = [0, 1, 0, 1, 0, 0]$. Then $\delta_2 = 0.7$ and as $\delta_2 \neq \delta_1$, we continue with δ_2. Again consider $X = [1, 1, 0, 0, 0, 0]$, as $\delta_2 \cdot \alpha = 0.525$, by comparing each $Q_{i,1}^X$ ($1 \leq i \leq n$) with 0.525, we derive $\widehat{R}^X = [1, 1, 0, 0, 1, 0]$ and F-score$^*(Q^X, \widehat{R}^X, \alpha) = 0.832$. By considering all 6 feasible X, the assignment $X = [1, 1, 0, 0, 0, 0]$ corresponds to the maximal F-score*, and $\delta_3 = 0.832 \neq \delta_2$. In the third iteration, similarly the assignment $X = [1, 1, 0, 0, 0, 0]$ corresponds to the $\delta_4 = 0.832$. As $\delta_4 = \delta_3$, we have $\delta^* = 0.832$ and return $X^* = [1, 1, 0, 0, 0, 0]$.*

Compared with Accuracy* in Example 4, which assigns q_2 and q_4 with the highest benefits, here the optimal assignment is q_1 and q_2 if the evaluation metric is F-score* with $\alpha = 0.75$. The reason is that $\alpha = 0.75$ focuses on Precision, and it tries to assign questions such that the estimated probability for the target label L_1 is of high confidence (or $Q_{i,1}$ is high). Thus it is more beneficial to assign q_1 compared with q_4, as $Q_{1,1}^w$ (0.818) $\geq Q_{4,1}^w$ (0.75).

[5]Here we present the brute-force way for illustration purpose.

5. COMPUTING CURRENT AND ESTIMATED DISTRIBUTION MATRICES

In this section, we examine how to compute Q^c and Q^w in Section 5.1 and 5.3 respectively. Since computing these two matrices requires some parameters, Section 5.2 discusses the parameters and the existing heuristics to compute them.

5.1 Current Distribution Matrix

When a worker completes a HIT, based on $D = \{D_1, D_2, \ldots, D_n\}$, we compute the parameters (including prior probability and worker model) and Q^c. As $Q_{i,j}^c$ represents the probability that question q_i's true label is L_j, we compute $Q_{i,j}^c$ based on the answer set of q_i, i.e., D_i. From Bayes' theorem we get

$$Q_{i,j}^c = P(t_i = j \mid D_i) = \frac{P(D_i \mid t_i = j) \cdot P(t_i = j)}{P(D_i)}. \tag{16}$$

Thus $Q_{i,j}^c \propto P(D_i \mid t_i = j) \cdot P(t_i = j)$. It means that if we derive $P(D_i \mid t_i = j) \cdot P(t_i = j)$ for each label L_j ($1 \leq j \leq \ell$), then we can normalize and finally get $Q_{i,j}^c$ for $1 \leq j \leq \ell$. We next discuss how to compute $P(t_i = j)$ and $P(D_i \mid t_i = j)$:

(1) $P(t_i = j)$ is the prior probability which represents that a question's true label is L_j, and it is commonly regarded as the proportion of questions whose true label is L_j, which is the same among different questions, so w.l.o.g., we denote $p_j = P(t_i = j)$. Existing works [1,21,22,56] usually estimated the prior as the expected fraction of questions whose ground truth is label L_j, i.e., $p_j = \mathbb{E}[\, \frac{\sum_{i=1}^n \mathbb{1}_{\{t_i = j\}}}{n} \,] = \frac{\sum_{i=1}^n Q_{i,j}^c}{n}$;

(2) $P(D_i \mid t_i = j)$ is the probability that the answer set for question q_i is D_i given that the question q_i's true label is L_j. Assume that the answers in D_i are independently answered by different workers (which was also adopted in [10,16,22,30,62]). Let a_i^w denote the index of answered label for q_i by worker w, then

$$P(D_i \mid t_i = j) = \prod_{(w,j') \in D_i} P(a_i^w = j' \mid t_i = j),$$

where $P(a_i^w = j' \mid t_i = j)$ is the probability that worker w answers label $L_{j'}$ given the true label is L_j, which can be expressed by worker model (Section 5.2). Initially, Q^c is set as uniform distribution for each question q_i, i.e., $Q_{i,j}^c = \frac{1}{\ell}$ for $1 \leq j \leq \ell$.

5.2 Parameters

For the issue about how to model a worker's quality, several works [16,26,30,62] define a worker w's quality as a single value $m^w \in [0, 1]$ called Worker Probability (WP) and

$$P(a_i^w = j' \mid t_i = j) = \begin{cases} m^w, \text{ for } j = j' \\ \frac{1 - m^w}{\ell - 1}, \text{ for } j \neq j' \end{cases}.$$

Some other works [1,22,61] define a worker w's quality as a $\ell \times \ell$ matrix M^w called Confusion Matrix (CM) and

$$P(a_i^w = j' \mid t_i = j) = M_{j,j'}^w \;\text{ for } 1 \leq j, j' \leq \ell.$$

For example, if an application has two labels, an example of WP for worker w is $m^w = 0.6$ and an example of CM for worker w is $M^w = \begin{bmatrix} 0.6 & 0.4 \\ 0.3 & 0.7 \end{bmatrix}$. In experiments (Section 6.2.2) we study the properties of different worker models on real datasets. For the initialization of WP and CM, each worker can be assumed as a perfect worker in the beginning (the assumption made by some prior work, e.g., Ipeirotis et al. [22]). Then for WP, $m^w = 1$; while for CM, $M^w_{j,j} = 1$ for $1 \le j \le \ell$ and $M^w_{j,j'} = 0$ for $j \ne j'$. Next Example 6 shows how to compute Q^c based on WP and prior.

EXAMPLE 6. *Suppose a question q_2 with three labels ($\ell = 3$) has been answered by two workers: $D_2 = \{(w_1, L_3), (w_2, L_1)\}$, where the worker models are $m^{w_1} = 0.7$, $m^{w_2} = 0.6$ and the priors are $p_1 = p_2 = p_3 = \frac{1}{3}$. In order to compute Q^c_2, based on Equation 16, we get $Q^c_{2,1} = P(t_2 = 1 \mid D_2) \propto P(a^{w_1}_2 = 3 \mid t_2 = 1) \cdot P(a^{w_2}_2 = 1 \mid t_2 = 1) \cdot P(t_2 = 1) = \frac{1-m^{w_1}}{\ell-1} \cdot m^{w_2} \cdot p_1 = 0.03$ and similarly $Q^c_{2,2} \propto 0.01$, $Q^c_{2,3} \propto 0.0467$. By normalization, we get $Q^c_2 = [0.346, 0.115, 0.539]$.*

To compute the worker model and prior parameters, we leverage the EM [10] algorithm, which uses the answer set as input, and adopts an iterative approach (in each iteration, E-step and M-step are applied) to update all parameters until convergence. The EM algorithm has been widely used [1,21,22,56] to infer the parameters and has a fast convergence rate. There are some other works [21,22,48,52,53,61] studying how to derive worker model and prior parameters, and they can be easily adapted to our system.

5.3 Estimated Distribution Matrix

We next discuss how to compute Q^w based on Q^c and w's worker model. For each $q_i \in S^w$, the computation of Q^w_i consists of two steps: in the first step, we estimate l^w_i, which is denoted as the index of the label that worker w will answer for q_i; in the second step, we construct an estimated answer set (denoted as D^w_i) for q_i by adding a tuple containing l^w_i into D_i, i.e., $D^w_i = D_i \cup \{(w, l^w_i)\}$, and then Q^w_i can be computed based on D^w_i. We next talk about the details.

First Step: In order to estimate l^w_i, we first compute the label distribution that worker w will answer q_i. We can derive $P(a^w_i = j' \mid D_i)$ by considering all possible true labels of question q_i:

$$P(a^w_i = j' \,|D_i) = \sum\nolimits_{j=1}^{\ell} P(a^w_i = j' \mid t_i = j, D_i) \cdot P(t_i = j \,|D_i). \quad (17)$$

(1) Given $t_i = j$ is known, $a^w_i = j'$ is independent of D_i, so $P(a^w_i = j' \mid t_i = j, D_i) = P(a^w_i = j' \mid t_i = j)$, which can be derived by w's worker model. (2) We have $Q^c_{i,j} = P(t_i = j \mid D_i)$. So we can compute $P(a^w_i = j' \mid D_i)$ for $1 \le j' \le \ell$.

Next, we predict l^w_i by capturing the label distribution, and the weighted random sampling method [13] is leveraged to select a label index $l^w_i \in \{1, 2, \ldots, \ell\}$, where the label index j has probability $P(a^w_i = j \mid D_i)$ to be sampled.

Second Step: To derive Q^w_i, we first construct $D^w_i = D_i \cup \{(w, l^w_i)\}$, then $Q^w_{i,j} = P(t_i = j \mid D^w_i) \propto P(D^w_i \mid t_i = j) \cdot p_j$. Similar to Section 5.1, we get $P(D^w_i \mid t_i = j) = P(a^w_i = l^w_i \mid t_i = j) \cdot \prod_{(w,j') \in D_i} P(a^w_i = j' \mid t_i = j)$. Take a further step, by considering Equation 16, we get

$$Q^w_{i,j} \propto Q^c_{i,j} \cdot P(a^w_i = l^w_i \mid t_i = j), \quad (18)$$

thus Q^w_i can be computed by normalization if we get $Q^c_{i,j} \cdot P(a^w_i = l^w_i \mid t_i = j)$ for $1 \le j \le \ell$. We next give an example (Example 7) to show how to compute Q^w based on Q^c in Figure 2.

EXAMPLE 7. *Given Q^c in Figure 2, suppose worker w with $m^w = 0.75$ requests a HIT. To compute Q^w, we take Q^w_1 as an example. In the first step we derive the label distribution to predict ℓ^w_1. Following Equation 17, $P(a^w_1 = 1 \,|D_1) = \sum_{j=1}^{2} P(a^w_1 = 1 \mid t_1 = j) \cdot P(t_1 = j \,|D_1) = 0.75 \cdot 0.8 + 0.25 \cdot 0.2 = 0.65$ and $P(a^w_1 = 2 \,|D_1) = 0.35$, thus the label distribution is $[0.65, 0.35]$. After weighted random sampling, suppose we get $\ell^w_1 = 1$. In the second step, following Equation 18, the proportion can be derived by multiplying $P(a^w_1 = 1 \mid t_1 = j)$ to $Q^c_{1,j}$ ($1 \le j \le 2$), which is $(0.8 \cdot 0.75) : (0.2 \cdot 0.25)$. To normalize the proportion, we get $Q^w_2 = [0.923, 0.077]$. Similarly other Q^w_i for $q_i \in S^w$ can be computed and Q^w can be constructed in Figure 2.*

6. EXPERIMENTS

We have conducted experiments on both simulated datasets (Section 6.1) and real datasets (Section 6.2).

6.1 Experiments on Simulated Data

We first describe the experimental settings (Section 6.1.1), then evaluate the performance of F-score* (Section 6.1.2), and finally examine the online assignment algorithms (Section 6.1.3)

6.1.1 Settings for Simulated Data

We show how to generate distribution matrices in simulated experiments. For F-score, as it focuses on a target label (L_1), to generate an $n \times 2$ distribution matrix Q, each question q_i's ($1 \le i \le n$) distribution is generated as follows: first the probability for target label is randomly generated as $Q_{i,1} \in [0, 1]$, then $Q_{i,2} = 1 - Q_{i,1}$. For Accuracy, to generate an $n \times \ell$ distribution matrix Q, for each question q_i ($1 \le i \le n$), we randomly generate $Q_{i,j} \in [0, 1]$ for $1 \le j \le \ell$, and then normalize Q_i to get a distribution. To achieve statistical significance, we perform each experiment for 1000 times and record its average value. Experiments are implemented in Python on a 16GB memory Ubuntu server.

*6.1.2 Evaluating Our Techniques for F-Score**

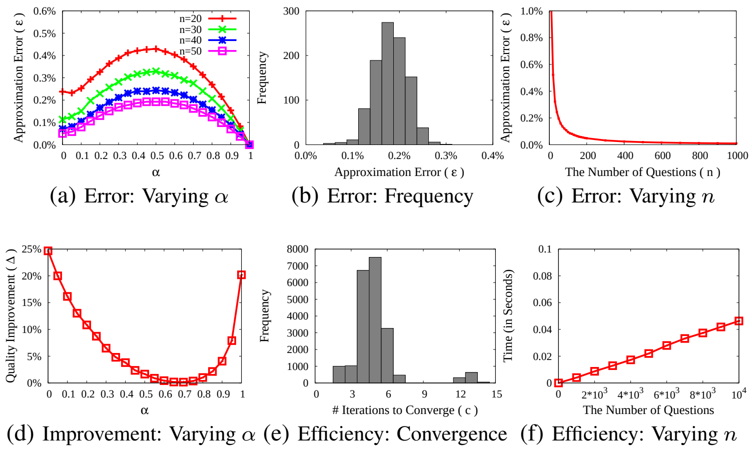
As directly computing $\mathbb{E}[\text{ F-score}(T, R, \alpha)]$ is inefficient, an approximation function F-score$^*(Q, R, \alpha)$ is proposed in Section 3.2.2. In this section, we evaluate this approximation on randomly generated distribution matrices. We first evaluate the approximation error, and then show the performance of its optimal result vector selection algorithm.

Approximation Error. We study how the proposed function F-score$^*(Q, R, \alpha)$ is approximate to $\mathbb{E}[\text{ F-score}(T, R, \alpha)]$ from Figure 3(a)-(c). In Figure 3(a), we vary $\alpha \in [0, 1]$ and $n \in [20, 50]$ to observe the approximation error. For a fixed α and n, we randomly generate Q and result vector R, and record the approximation error $\epsilon = |\text{ F-score}^*(Q, R, \alpha) - \mathbb{E}[\text{ F-score}(T, R, \alpha)]|$ in the figure (note that the error is averaged over 1000 repeated trials). It shows that as n varies in $[20, 50]$, the approximation error decreases, and when $n = 20$, the errors for different α are smaller than 0.5%. As α varies in $[0, 1]$, the average error reaches the highest at around $\alpha = 0.5$. Moreover, an interesting observation is that the curve is not symmetric, especially when $\alpha = 1$ (which is Precision) and $\alpha = 0$ (which is Recall). The reason is that (1) $\mathbb{E}[\text{ Precision}(T, R)] = \mathbb{E}[\frac{\sum_{i=1}^n \mathbb{1}_{\{t_i=1\}} \cdot \mathbb{1}_{\{r_i=1\}}}{\sum_{i=1}^n \mathbb{1}_{\{r_i=1\}}}] = \frac{\sum_{i=1}^n Q_{i,1} \cdot \mathbb{1}_{\{r_i=1\}}}{\sum_{i=1}^n \mathbb{1}_{\{r_i=1\}}} = \text{F-score}^*(Q, R, 1)$, so $\epsilon = 0$ as $\alpha = 1$; (2) $\mathbb{E}[\text{ Recall}(T, R)] = \mathbb{E}[\frac{\sum_{i=1}^n \mathbb{1}_{\{t_i=1\}} \cdot \mathbb{1}_{\{r_i=1\}}}{\sum_{i=1}^n \mathbb{1}_{\{t_i=1\}}}] \approx \frac{\sum_{i=1}^n Q_{i,1} \cdot \mathbb{1}_{\{r_i=1\}}}{\sum_{i=1}^n Q_{i,1}} = \text{F-score}^*(Q, R, 0)$, so $\epsilon \ne 0$ as $\alpha = 0$.

Furthermore, to observe the distribution of ϵ, in Figure 3(b), we set $n = 50$ and $\alpha = 0.5$, and record the frequency of ϵ over 1000 trials. We observe that the error is centered around the mean 0.19%, and ranges from 0% to 0.31%, which is small and stable. To observe how ϵ changes for a larger n, in Figure 3(c) we fix $\alpha = 0.5$ and record ϵ by varying $n \in [0, 10^3]$. The curve shows that ϵ consistently decreases and it conforms to the error bound (i.e., $\epsilon = \mathcal{O}(n^{-1})$) as derived in Section 3.2.2. Especially when $n = 1000$, the average $\epsilon \le 0.01\%$, which is fairly small.

To summarize, the proposed function F-score$^*(Q, R, \alpha)$ (Equation 9) is a good approximation of $\mathbb{E}[\text{ F-score}(T, R, \alpha)]$ in practice.

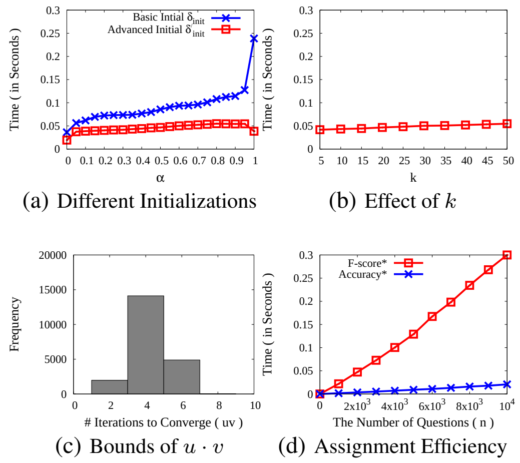
Optimal Result Vector Selection. In Figure 3(d)-(f), we study the result vector selection. Existing works [17,46,56] evaluate F-score by choosing the result of each question as the label with the highest probability. We first study the quality improvement by using our optimal returned result vector algorithm. Let R^* denote the optimal result vector, i.e., $R^* = \text{argmax}_R \text{ F-score}^*(Q, R, \alpha)$. Let



(a) Error: Varying α (b) Error: Frequency (c) Error: Varying n

(d) Improvement: Varying α (e) Efficiency: Convergence (f) Efficiency: Varying n

Figure 3: Evaluating F-Score* (Simulation)



(a) Different Initializations (b) Effect of k

(c) Bounds of $u \cdot v$ (d) Assignment Efficiency

Figure 4: Efficiency of Assignments (Simulation)

$\widetilde{R} = [\tilde{r}_1, \tilde{r}_2, \ldots, \tilde{r}_n]$ denote the result vector chosen by the existing works [17,46,56], i.e., $\tilde{r}_i = 1$ if $Q_{i,1} \geq Q_{i,2}$ (or $Q_{i,1} \geq 0.5$) and $\tilde{r}_i = 2$ if otherwise. We set $n = 2000$ and vary $\alpha \in [0,1]$, where for a fixed α, we randomly generate Q, and respectively compute R^* and $\widetilde{R}$ based on Q. Then we define the quality improvement as

$$\Delta = \mathbb{E}[\,\text{F-score}(T, R^*, \alpha)\,] - \mathbb{E}[\,\text{F-score}(T, \widetilde{R}, \alpha)\,], \qquad (19)$$

and we record Δ in Figure 3(d). It can be observed that R^* improves the quality of $\widetilde{R}$ a lot. As α varies in $[0,1]$, about 25% of the values of α result in an improvement of over 10%. We also observe that the curve is asymmetric bowl-shaped, especially when α is around 0.65, Δ becomes zero. Next we apply the approximated function F-score* to verify this interesting phenomenon in theory. For some unknown α', if $\widetilde{R}$ is equal to R^* (or $\widetilde{R} = R^*$),
(1) as $\widetilde{R}$ is constructed by comparing with the threshold 0.5, thus from Theorem 2 we know the threshold $\theta = \lambda^* \cdot \alpha' = 0.5$ and
(2) as $\lambda^* = \text{F-score}^*(Q, R^*, \alpha')$, and $R^* = \widetilde{R}$, we have $\lambda^* = \frac{\sum_{i=1}^{n} \mathbb{1}_{\{Q_{i,1} \geq 0.5\}} \cdot Q_{i,1}}{\alpha' \cdot \sum_{i=1}^{n} \mathbb{1}_{\{Q_{i,1} \geq 0.5\}} + (1-\alpha') \cdot \sum_{i=1}^{n} Q_{i,1}}$. Taking $\lambda^* \cdot \alpha' = 0.5$ inside, we can obtain $\sum_{i=1}^{n} Q_{i,1} \cdot \mathbb{1}_{\{Q_{i,1} \geq 0.5\}} = 0.5 \cdot \big[\sum_{i=1}^{n} \mathbb{1}_{\{Q_{i,1} \geq 0.5\}} + (\frac{1}{\alpha'} - 1) \cdot \sum_{i=1}^{n} Q_{i,1} \big]$. Note that as we randomly generate $Q_{i,1}$ $(1 \leq i \leq n)$ for all questions, it makes $Q_{i,1}$ $(1 \leq i \leq n)$ uniformly distributed in $[0,1]$. Thus if we take the expectation on both sides of the obtained formula, and apply the properties of uniform distribution, we can derive $0.75 \cdot \frac{n}{2} = 0.5 \cdot \big[\, \frac{n}{2} + (\frac{1}{\alpha'} - 1) \cdot 0.5 \cdot n \,\big]$, and then get $\alpha' = 0.667$, which verifies our observation (around 0.65).

As the time complexity of optimal vector selection algorithm (Section 3.2.3) is $\mathcal{O}(c \cdot n)$, we next evaluate the bound of c (#iterations to converge) in Figure 3(e). For a fixed $n = 2000$, we vary α from 0 to 1 with a step of 0.1, and for each α, we randomly generate Q and record c by running our algorithm. Figure 3(e) illustrates the frequency of c for all α. It can be seen that $c \leq 15$ in general (for different α). To evaluate the efficiency with a larger n, we vary $n \in [0, 10^4]$ and record the running time in Figure 3(f), which shows that the time linearly increases with n, and our algorithm finishes within 0.05s even when n is large ($n = 10^4$).

6.1.3 Evaluating Online Assignment's Efficiency

We evaluate the online assignment algorithm's efficiency in Figure 4(a)-(d). First we compare different ways of initialization in our algorithm. A basic way is to initialize $\delta_{init} = 0$. By considering that the optimal F-score* for Q^{X^*} may not be far away from the optimal F-score* for Q^c (as they are only different in k questions' distributions), we propose another way of initialization: $\delta'_{init} = \max_R \text{ F-score}^*(Q^c, R, \alpha)$, which can be computed using the optimal result vector selection algorithm (Section 3.2.3). To compare their efficiency, in Figure 4(a) we set $n = 2000$, $k = 20$, and vary $\alpha \in [0,1]$. For a fixed α, we randomly generate Q^c and a confusion matrix, then Q^w can be computed (Equation 18). We run our algorithm with respective δ_{init} and δ'_{init}, and record the time in Figure 4(a). It can be seen that both initializations are efficient (≤ 0.3s), but there is a sudden increase as $\alpha \geq 0.95$ for δ_{init} (mainly due to the fact that big α focuses on Precision, that is, only confident questions for L_1 will be returned, which leads to a big δ^*, and $\delta_{init} = 0$ is far away from δ^*). For δ'_{init}, it is steady with different α, as the initialization is computed by considering both Q^c and α. So we use δ'_{init} to initialize in later experiments.

We next evaluate the performance of different k in Figure 4(b). We set $n = 2000$ and vary $k \in [5, 50]$ to observe the assignment's efficiency, which shows that it is invariant with k. The main reason is that our algorithm iteratively updates δ, and the update solves a 0-1 FP problem via Dinkelbach framework, which is invariant of the size k. To evaluate the bound for $u \cdot v$, in Figure 4(c), we set $n = 2000$ and vary α from 0 to 1 with a step of 0.1. For each α, we generate Q^c and Q^w, and record the frequency of $u \cdot v$ in Figure 4(c). It showed that generally $u \cdot v \leq 10$ for different α.

We then evaluate the assignment's efficiency for Accuracy* and F-score* with a larger n. We set $k = 20$, $\alpha = 0.5$, and vary $n \in [0, 10^4]$ in Figure 4(d), which shows that both algorithms are efficient. For example, both of them finish within 0.3s when $n = 10^4$. As reported in [23], the number of HITs for a certain task is usually $\leq$ 5000, even for the top invested requesters. Therefore, our assignment algorithms can work well in a real crowdsourcing platform. Moreover, the assignment time both linearly increases for Accuracy* and F-score*, but with a larger factor in F-score*, as it has to deal with the converging complexity (i.e., $u \cdot v$).

To summarize, the assignment algorithms are both efficient for Accuracy* and F-score*, and they work well even for a large number of tasks (e.g., $n = 10^4$).

6.2 Experiments for Real Datasets

We first discuss the experimental setup (Section 6.2.1), then evaluate the properties of different worker models (Section 6.2.2), and finally compare with existing systems (Section 6.2.3).

6.2.1 Settings for Real Datasets

We generate five applications from real-world datasets (their ground truth is known for evaluation purposes). The result quality for the first two applications are evaluated in Accuracy, which is introduced as follows:

Films Poster (FS): *FS* uses the top 500 films (with posters) with the highest ratings from IMDB[6]. We generate 1,000 questions, where

[6]http://www.imdb.com/

Table 1: Application Properties (Per System)

Data	Labels	n	k	m	B	Evaluation Metric
FS	L_1 ="<", L_2 ="≥"	1000	4	750	\$15	Accuracy
SA	L_1 ="positive", L_2 ="neutral", L_3 ="negative"	1000	4	750	\$15	Accuracy
ER	L_1 ="equal", L_2 ="non-equal"	2000	4	1500	\$30	F-score for L_1 ($\alpha = 0.5$)
PSA	L_1 ="positive", L_2 ="non-positive"	1000	4	750	\$15	F-score for L_1 ($\alpha = 0.75$)
NSA	L_1 ="negative", L_2 ="non-negative"	1000	4	750	\$15	F-score for L_1 ($\alpha = 0.25$)

each question asks the crowd to compare two different films and the coming worker should decide whether one film is published earlier (<) or later (≥) than the other.

Sentiment Analysis (SA): *SA* uses the Twitter[7] dataset to label the sentiments of tweets w.r.t. different companies. It contains 5,152 hand-classified tweets. We select 1,000 tweets from them and each question asks the crowd to label the sentiment ("positive", "neutral" or "negative") of a tweet w.r.t. the related company.

The result quality for the other three applications is evaluated in F-score (with different α), which is introduced as follows:

Entity Resolution (ER): *ER* uses Product[8] dataset to evaluate whether the descriptions of two products refer to the same product or not. It contains 1,180,452 product pairs. For each pair (r_1,r_2), we compute the Jaccard similarity ($\frac{r_1 \cap r_2}{r_1 \cup r_2}$) and select 2,000 pairs with similarity ≥ 0.7 as our questions. Each question contains a product pair and the coming worker should decide whether they are "equal" or "non-equal". It is evaluated using balanced F-score for the "equal" label ($\alpha = 0.5$).

Positive Sentiment Analysis (PSA): Based on the demand that a company manager may want to select positive sentiment comments for their products with high confidence (emphasizing Precision), we generate *PSA* by selecting 1,000 tweets related to Apple company from the Twitter[7] dataset. Each question is to ask the crowd about whether the sentiment of a tweet related to Apple is "positive" or "non-positive". As it emphasizes Precision, we set the evaluation metric as F-score for "positive" where $\alpha = 0.75$.

Negative Sentiment Analysis (NSA): From another perspective, a company manager may want to collect as many negative sentiment comments as possible for their products (emphasizing Recall), then *NSA* is generated by selecting 1,000 tweets related to Apple company from the Twitter[7] dataset. Each question is to ask the crowd about whether the sentiment of a tweet related to Apple is "negative" or "non-negative". It is evaluated using F-score for "negative" where $\alpha = 0.25$ (emphasizing Recall).

We compare QASCA with other five systems, i.e., Baseline, CDAS [30], AskIt! [3], MaxMargin and ExpLoss:

(1) Baseline: It randomly assigns k questions to a coming worker.

(2) CDAS [30]: It adopts a quality-sensitive answering model to measure the confidence of questions' current results, and terminates assigning the questions which have already got confident results. It assigns k non-terminated questions.

(3) AskIt! [3]: It uses an entropy-like method to define the uncertainty of each question, and assigns k most uncertain questions.

(4) MaxMargin: It selects the next question with the highest expected marginal improvement, disregarding the characteristics of the worker, and it assigns k selected questions.

(5) ExpLoss: It selects the next question by considering the expected loss, defined as $\min_j \sum_{j'=1}^{\ell} p_{j'} \cdot \mathbb{1}_{\{j \neq j'\}}$ $(1 \leq j \leq \ell)$, and it assigns k selected questions.

[7] http://www.sananalytics.com/lab/twitter-sentiment/

[8] http://dbs.uni-leipzig.de/file/Abt-Buy.zip

Table 2: Comparison between Worker Models

	FS	*SA*	*ER*	*PSA*	*NSA*
CM	93.33%	87.09%	82.02%	91.73%	88.43%
WP	93.33%	78.64%	72.80%	92.61%	87.01%

Table 3: The average quality improvement ($\widehat{\Delta}$) of each system by using our optimal result selection algorithm (Section 3.2.3)

	Baseline	CDAS	AskIt!	MaxMargin	ExpLoss
ER ($\alpha = 0.5$)	2.59%	2.69%	4.56%	5.49%	4.32%
PSA ($\alpha = 0.75$)	4.14%	2.96%	1.26%	2.08%	1.66%
NSA ($\alpha = 0.25$)	14.12%	10.45%	12.44%	14.26%	9.98%

As workers may vary in quality for different rounds, we need to use the same set of workers in the comparison of all six systems. To achieve this, when worker w comes, we use each system to assign $k = 4$ questions, then $k \times 6 = 24$ questions are batched into a HIT in a random order and the HIT will be assigned to worker w. Following this way, we can evaluate six systems in a "parallel" way with the same set of workers. We pay each worker \$0.12 for doing a HIT, and assign each question to $z = 3$ workers on average, which are typical experimental settings in AMT [56,59]. The detailed settings are listed in Table 1. Take *FS* application as an example: there are $n = 1000$ generated questions, and the "parallel" way assigns $m = \frac{n \times z}{k} = \frac{1000 \times 3}{4} = 750$ HITs in total for all six systems, where each system corresponds to $k = 4$ questions in a HIT and each system takes $B = \frac{m \times \$0.12}{6} = \$15$ for *FS*. When HITs are finished, we compute the result quality w.r.t. the corresponding evaluation metric based on the questions' ground truth and report their comparison results. All the HITs are published during 5:00 pm $\sim$ 10:00 pm (PDT) on weekdays, and all the crowdsourcing applications are finished within 24 hours. The number of workers participated in five applications *FS*, *SA*, *ER*, *PSA* and *NSA* are respectively 97, 101, 193, 104, and 101.

We implement QASCA in Python using the Django web framework, and deploy it on a 16GB memory Ubuntu server. We conduct experiments on a widely-used crowdsourcing platform: AMT. We use the "external-HIT" way provided by AMT which embeds the generated HTML pages (by our server) into its frame and workers directly interact with our server through the frame. When a worker comes, we can identify the worker from the individual worker-id provided by AMT. Then we can dynamically batch the selected questions (from different systems) in a HIT for the coming worker.

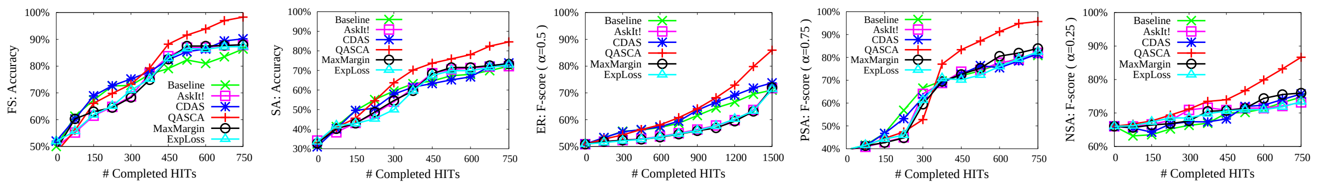
6.2.2 Evaluating Worker Models

In this section, we especially study the selection of worker models in computing the distribution matrix. Recall that Section 5.2 reviews two typical worker models used in existing works: Worker Probability (WP) and Confusion Matrix (CM). Intuitively CM is better than WP as it is more complex. Even though what we observe from real datasets validates the intuition, we try to explain some interesting observations. We first collect the crowd's answers from the five published applications on AMT.

In Table 2, we compare WP and CM on five applications by leveraging the ground truth (t_i) of each question. Based on the collected answer set $D = \{D_1, D_2, \ldots, D_n\}$, for each worker w we compute the real WP and CM as follows

$$\begin{cases} \widetilde{m}^w = \frac{\sum_{i=1}^{n} \mathbb{1}_{\{(w,t_i) \in D_i\}}}{\sum_{i=1}^{n} \sum_{j=1}^{\ell} \mathbb{1}_{\{(w,j) \in D_i\}}}, \\ \widetilde{M}^w_{j,j'} = \frac{\sum_{i=1}^{n} \mathbb{1}_{\{t_i = j\}} \cdot \mathbb{1}_{\{(w,j') \in D_i\}}}{\sum_{i=1}^{n} (\mathbb{1}_{\{t_i = j\}} \cdot \sum_{z=1}^{\ell} \mathbb{1}_{\{(w,z) \in D_i\}})}. \end{cases} \tag{20}$$

We leverage the real priors (computed as $\widetilde{p}_j = \frac{\sum_{i=1}^{n} \mathbb{1}_{\{t_i = j\}}}{n}$ for $1 \leq j \leq \ell$) and two respective real worker models to derive the distribution matrix based on Equation 16, and then the optimal result vector R^* w.r.t. corresponding evaluation metric can be computed. Finally we evaluate the quality of R^* using the ground truths. To avoid overfitting, we randomly select 80% of the ques-



(a) *FS*: Accuracy (b) *SA*: Accuracy (c) *ER*: F-score (d) *PSA*: F-score (e) *NSA*: F-score

Figure 5: End-to-End Experiments (Comparison on Real Result Quality by Varying # Completed HITs)

tions answered by each worker to compute WP and CM, respectively. The random process is repeated over 1000 trails and we record the average quality of the results in Table 2. We observe that CM performs better than WP for *SA* and *ER* applications, while they perform approximately the same for other three applications.

To explain this, we know that $\widetilde{m}^w$ performs equally with $\widetilde{M}^w$ only if the following requirements hold: (1) $\widetilde{M}^w_{j,j'} = \widetilde{M}^w_{j,j''}$ for $j' \neq j \neq j''$ (non-triangular elements in $\widetilde{M}^w$, which expresses the relations between pairwise labels); (2) $\widetilde{M}^w_{j,j} = \widetilde{M}^w_{j',j'}$ (triangular elements in $\widetilde{M}^w$, which expresses the difficulty of individual label).

CM is better than WP on *SA* and *ER*, as they do not satisfy the above requirements: for *SA*, if a question's true label is "positive", a worker is more likely to answer "neutral" compared with "negative", i.e., $\widetilde{M}^w_{1,2} > \widetilde{M}^w_{1,3}$, which violates the first requirement, as there exists relations between pairwise labels; for *ER*, two product descriptions are "equal" only if all features (brand, color, etc.) are the same, while they are "non-equal" once a different feature is spotted, which implies that identifying a "non-equal" pair is easier than correctly identifying an "equal" pair, i.e., $\widetilde{M}^w_{1,1} < \widetilde{M}^w_{2,2}$ (*easier* means a higher probability), which violates the second requirement, as different labels have different difficulties.

To summarize, the above observations among labels in the two applications (*SA* and *ER*) can only be captured by CM other than WP. However, the labels in the other three applications (*FS*, *PSA* and *NSA*) do not have explicit relations or difficulties, which perform similarly for WP and CM. As CM performs at least as good as WP, we use the CM worker model in later experiments.

6.2.3 End-to-End Experiments

We compare QASCA with five other systems on five crowdsourcing applications in AMT.

Improvement of Optimal Result Selection. As we have validated the improvement in Section 6.1.2 on simulated datasets, here we further explore its benefit on real datasets. Recall the definition of R^* and $\widetilde{R}$ in Section 6.1.2. As the ground truth vector T is known on real datasets, we define the real quality improvement w.r.t. different result vectors (R^* and $\widetilde{R}$) as

$$\widehat{\Delta} = \text{F-score}(T, R^*, \alpha) - \text{F-score}(T, \widetilde{R}, \alpha). \qquad (21)$$

As HITs are completed, we compute R^* and $\widetilde{R}$ based on Q by time, and record $\widehat{\Delta}$ at each time-stamp. Finally after all HITs are completed, we report the average $\widehat{\Delta}$ for three applications (i.e., *ER*, *PSA* and *NSA*)[9] with the evaluation metric F-score in Table 3. Recall that the three applications *ER*, *PSA* and *NSA* have different α (i.e., 0.5, 0.75 and 0.25), and their corresponding Δ in simulated experiments (Figure 3(d)) are around 2%, 1% and 9%, respectively. In Table 3, we observe that the average $\widehat{\Delta}$ are larger than zero on all applications, which means that all systems can benefit from choosing the optimal result vector R^*. We also observe that *NSA* ($\alpha = 0.25$) has a bigger improvement compared with *ER* and *PSA*, which conforms to the observation in simulated experiments. Note that the main reason that $\widehat{\Delta}$ may have some differences from Δ is that in our simulated experiments, the $Q_{i,1}$ for $1 \leq i \leq n$ is uniformly distributed. While in reality, as a question q_i gets more answers, the computed distribution Q_i may become more confident (either $Q_{i,1}$ is close to 1 or 0). To summarize, for F-score, all systems can benefit from choosing the optimal result vector R^* rather than returning a label with the highest probability ($\widetilde{R}$).

[9] In Theorem 1 we have proved that $R^* = \widetilde{R}$ for Accuracy*, so we did not include the applications with the evaluation metric Accuracy (i.e., *FS* and *SA*).

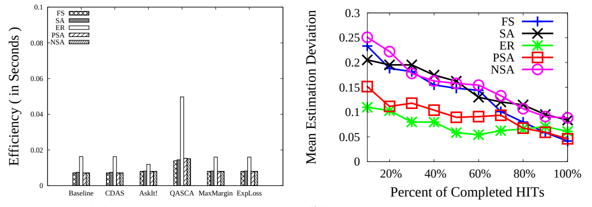
Table 4: Overall Result Quality (All HITs completed)

Dataset	*FS*	*SA*	*ER*	*PSA*	*NSA*
Baseline	86.40%	72.20%	71.07%	80.85%	74.73%
CDAS	90.20%	73.80%	73.75%	81.58%	75.68%
AskIt!	87.90%	72.20%	71.78%	81.95%	73.16%
QASCA	**98.30%**	**84.60%**	**85.96%**	**95.70%**	**86.65%**
MaxMargin	88.00%	73.30%	72.02%	83.92%	75.96%
ExpLoss	87.30%	72.90%	71.36%	82.43%	73.38%

System Comparison Results. We next show the main results, i.e., the real result quality compared with other systems on all applications in Figure 5(a)-(e). We collect the completed HITs by time and calculate the corresponding result quality based on the derived results of different systems. Since we have shown that the quality improves a lot by selecting the optimal result vector R^* for F-score* both in simulated datasets (Figure 3(d)) and real datasets (Table 3). To make a fair comparison, we apply this optimization (i.e., selecting R^*) to all systems when the evaluation metric is F-score.

From Figure 5(a)-(e), we can observe that in the beginning, when the number of completed HITs is small, all systems have similar performance, as they all do not know much information about questions or workers. However, QASCA dominates other systems as time goes by. This is because QASCA assigns the best k questions for each coming worker to maximize the evaluation metric value (Accuracy* or F-score*), while other systems do not explicitly consider the impact of evaluation metrics on result quality in their task assignment process. For a clear comparison, Table 4 shows the final result quality value when all HITs are completed, and QASCA leads other systems over 8% for all applications. To be specific, QASCA leads the second best system by 8.1%, 10.8%, 12.21%, 11.78% and 10.69% on five real-world datasets (*FS*, *SA*, *ER*, *PSA* and *NSA*), respectively. We can also observe that MaxMargin outperforms ExpLoss, as for the inherently ambiguous questions, they will have high expected loss (ExpLoss will continuously assign them); while the marginal benefit of assigning these questions will be much lower as more answers are collected (MaxMargin can save the assignments for more beneficial questions).

We compare the efficiency of different systems in Figure 6(a). To make a fair comparison, for an application in each system, we record the worst case assignment time during the assignment process for all HITs. It is shown that all systems are efficient, and the worst case assignment of all systems can be finished within 0.06s, which is fairly acceptable in real applications. Even though QASCA is less efficient than other systems due to its complexity in assignments, it can significantly improve the result quality (over 8%). The reason why *ER* runs slower is that it contains 2000 questions while other applications contain 1000 questions.



(a) Efficiency (b) Mean Estimation Deviation

Figure 6: Efficiency and Mean Estimation Deviation

Estimation of Worker Quality. We next examine how the estimated quality of workers is different from the real quality of workers[10] by ranging the percentage of completed HITs. Let us denote the real CM as $\widetilde{M}^w$ and the estimated CM as M^w. For a worker w, we further define the *estimation deviation* of worker quality as the absolute difference of estimated quality and real quality, i.e., $\frac{1}{\ell \times \ell} \cdot \sum_{j=1}^{\ell} \sum_{j'=1}^{\ell} |M^w_{j,j'} - \widetilde{M}^w_{j,j'}|$. Then we calculate the *mean estimation deviation* by averaging the calculated estimation deviation among all workers. The smaller the mean estimation deviation is, the closer the estimated worker quality (M^w) is to the real one ($\widetilde{M}^w$). We report the mean estimation deviation by ranging the percentage of completed HITs for all datasets in Figure 6(b). It shows that as more HITs are completed, the estimated worker quality gets closer to the real one, which may explain why QASCA performs much better compared with other systems as time goes by. That is, as more HITs are completed, the worker's quality is more accurately estimated, then QASCA takes the desired quality metric into consideration and can better leverage the estimated worker's quality to judge how the worker's answers might affect the quality metric if questions are assigned. Then it selects the assignment that could maximize the quality metric.

7. RELATED WORK

Crowdsourcing Systems: Nowadays, crowdsourcing has been widely used to address challenging problems that are hard for machines. Crowdsourced query processing systems, such as CrowdDB [14], Deco [41] and Qurk [35] are built to incorporate the crowd into query processing. Many studies investigate how to implement crowd-based operators, e.g., Join [34,56,59], Sort [5,34], Max [16,54], Group-by [9,33], and Enumeration [50]. Some recent work [57] uses sampling to reduce operators' costs. We explore how to devise online task assignment approaches to improve the result quality of generated questions. Our work can be extended to some of them. For example, in [59], the authors iteratively publish a batch of questions on the crowdsourcing platform and can benefit by publishing questions using our system in each iteration.

Evaluation Metrics: Accuracy [8,21,22,30,44] and F-score [25, 31,42,56,59,60,63] are two widely used evaluation metrics for crowdsourcing applications, where Accuracy evaluates the overall quality and F-score focuses on the quality of a specific label. Note that there are other metrics defined based on different purposes. In entity resolution, there are also several cluster-based metrics, such as K-measure [27], GMD measure [36] and Rand-index [49]. For strings, similarity-based metrics including Jaccard, Dice, Cosine and Edit Distances are defined and used [11,55,58]. We focus on studying the question-based metrics and propose solutions to estimate the result quality based on distribution matrices.

Parameter Computation: There are also some approaches addressing how to compute worker model and prior parameters. Two typical worker models are worker probability (WP) [16,26,30,62] and confusion matrix (CM) [1,22,61]. To compute them, Raykar et al. [43] leveraged empirical Bayesian analysis to estimate the parameters; Ipeirotis et al. [22] applied EM algorithm to iteratively update the parameters; Venanzi et al. [52] used Community Detection methods to compute parameters, which works well for workers with limited answers. There are also some works [8,26,61] addressing more general worker models and discussing how to infer the parameters. Other works [21,48] make a comparison among different inference methods. Different from them (*offline inference problem*), we study an *online task assignment problem*, and their parameter computation methods can be adapted to our framework.

Task Assignment: CDAS [30] and AskIt! [3] are the two systems that are most similar to QASCA. CDAS [30] adopts a quality-sensitive answering model along with an early termination scheme, and AskIt! [3] utilizes entropy-like methods to measure each question's uncertainty. Compared with these systems, QASCA takes into account different evaluation metrics in the assignment scheme, and achieved better result quality. There are some other works about online task assignment in crowdsourcing, but they focus on different perspectives. Parameswaran et al. [40] develop algorithms to filter items based on a set of properties; Gao et al. [15] introduce a cost sensitive method to determine whether questions can be better solved by crowdsourcing or machine-based methods; Xi et al. [6] address the budget allocation problem using the proposed extended Markov Decision Process framework; Mo et al. [38] study how to optimize the plurality of a HIT; Sheng et al. [47] investigate how much repeated labeling helps in achieving better results; Ho et al. [18,19] and Mo et al. [37] explore how to assign heterogeneous tasks (tasks of multiple types) to workers.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we have studied the online task assignment problem. We proposed a novel assignment framework by incorporating evaluation metrics into assignment strategies. We generalized the widely used existing evaluation metrics (Accuracy and F-score) to be able to quantify the result quality w.r.t a distribution matrix. We designed optimal result vector selection algorithms and two respective efficient online assignment algorithms for Accuracy and F-score. We built the QASCA system by integrating our novel assignment framework with AMT, and evaluated our system on five real applications. The experimental results showed that QASCA achieved much better result quality than existing approaches.

In the future, we plan to extend our work in five aspects: (1) As QASCA maintains the worker history and expects them to come back to answer another set of questions, it might be interesting to construct certain scope of active advertisers and actively recruit them to answer questions; (2) We will further investigate the method of dealing with continuous values and more complicated question types (e.g., cluster-based question [56] and transcription question [23]); (3) More evaluation metrics will be incorporated in the assignment framework; (4) We focus on question assignments over a specific (or homogeneous) task, then how to incorporate heterogeneous tasks into the assignment framework is another direction; (5) We also plan to consider if a requester has multiple metrics in her mind (say both Accuracy and F-score), then how can we wisely assign questions to a coming worker.

Acknowledgement

We thank the reviewers for their insightful comments. Guoliang Li and Jianhua Feng were supported by the 973 Program of China (2015CB358700), the NSFC project (61422205, 61472198), the Chinese Special Project of Science and Technology (2013zx01039-002-002), Tencent, Huawei, SAP, YETP0105, the "NExT Research Center" (WBS: R-252-300-001-490), and the FDCT/106/2012/A3. Reynold Cheng and Yudian Zheng were supported by HKU (Projects 201311159095 and 201411159171).

[10]The real quality of each worker is calculated by leveraging the ground truth T and the answer set D. We can follow Section 6.2.2 (Equation 20) to compute the real CM.

9. REFERENCES

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

[2] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973.

[3] R. Boim, O. Greenshpan, T. Milo, S. Novgorodov, N. Polyzotis, and W. C. Tan. Asking the right questions in crowd data sourcing. In *ICDE*, 2012.

[4] C.G.Small. *Expansions and asymptotics for statistics*. CRC Press, 2010.

[5] X. Chen, P. N. Bennett, K. Collins-Thompson, and E. Horvitz. Pairwise ranking aggregation in a crowdsourced setting. In *WSDM*, pages 193–202, 2013.

[6] X. Chen, Q. Lin, and D. Zhou. Optimistic knowledge gradient policy for optimal budget allocation in crowdsourcing. In *ICML*, pages 64–72, 2013.

[7] J. Costa, C. Silva, M. Antunes, and B. Ribeiro. On using crowdsourcing and active learning to improve classification performance. In *ISDA*, 2011.

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

[9] 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.

[10] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *J.R.Statist.Soc.B*, 30(1):1–38, 1977.

[11] D. Deng, G. Li, S. Hao, J. Wang, and J. Feng. Massjoin: A mapreduce-based method for scalable string similarity joins. In *ICDE*, pages 340–351, 2014.

[12] W. Dinkelbach. On nonlinear fractional programming. *Management Science*, 13(7):492–498, March,1967.

[13] P. Efraimidis and P. G. Spirakis. Weighted random sampling with a reservoir. *Inf. Process. Lett.*, 97(5):181–185, 2006.

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

[15] J. Gao, X. Liu, B. C. Ooi, H. Wang, and G. Chen. An online cost sensitive decision-making method in crowdsourcing systems. In *SIGMOD*, 2013.

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

[17] K. Hara, V. Le, and J. Froehlich. Combining crowdsourcing and google street view to identify street-level accessibility problems. In *CHI*, 2013.

[18] C.-J. Ho, S. Jabbari, and J. W. Vaughan. Adaptive task assignment for crowdsourced classification. In *ICML (1)*, pages 534–542, 2013.

[19] C.-J. Ho and J. W. Vaughan. Online task assignment in crowdsourcing markets. In *AAAI*, 2012.

[20] J. J. Horton and L. B. Chilton. The labor economics of paid crowdsourcing. In *ACM Conference on Electronic Commerce*, pages 209–218, 2010.

[21] N. Q. V. Hung, N. T. Tam, L. N. Tran, and K. Aberer. An evaluation of aggregation techniques in crowdsourcing. In *WISE*, pages 1–15. Springer, 2013.

[22] P. Ipeirotis, F. Provost, and J. Wang. Quality management on amazon mechanical turk. In *SIGKDD workshop*, pages 64–67, 2010.

[23] P. G. Ipeirotis. Analyzing the amazon mechanical turk marketplace. *ACM Crossroads*, 17(2):16–21, 2010.

[24] M. Jansche. A maximum expected utility framework for binary sequence labeling. In *ACL*, 2007.

[25] S. R. Jeffery, M. J. Franklin, and A. Y. Halevy. Pay-as-you-go user feedback for dataspace systems. In *SIGMOD*, pages 847–860, 2008.

[26] D. R. Karger, S. Oh, and D. Shah. Iterative learning for reliable crowdsourcing systems. In *NIPS*, pages 1953–1961, 2011.

[27] A. H. Laender, M. A. Gonçalves, R. G. Cota, A. A. Ferreira, R. L. T. Santos, and A. J. Silva. Keeping a digital library clean: New solutions to old problems. In *DocEng*, pages 257–262. ACM, 2008.

[28] D. D. Lewis. Evaluating and optimizing autonomous text classification systems. In *SIGIR*, pages 246–254, 1995.

[29] X. Li, X. L. Dong, K. Lyons, W. Meng, and D. Srivastava. Truth finding on the deep web: Is the problem solved? *PVLDB*, 6(2):97–108, 2012.

[30] X. Liu, M. Lu, B. C. Ooi, Y. Shen, S. Wu, and M. Zhang. Cdas: A crowdsourcing data analytics system. *PVLDB*, 5(10):1040–1051, 2012.

[31] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008.

[32] C. D. Manning and H. Schütze. *Foundations of statistical natural language processing*. MIT Press, 2001.

[33] A. Marcus, D. R. Karger, S. Madden, R. Miller, and S. Oh. Counting with the crowd. *PVLDB*, 6(2):109–120, 2012.

[34] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller. Human-powered sorts and joins. *PVLDB*, 5(1):13–24, 2011.

[35] A. Marcus, E. Wu, S. Madden, and R. C. Miller. Crowdsourced databases: Query processing with people. In *CIDR*, pages 211–214, 2011.

[36] D. Menestrina, S. E. Whang, and H. Garcia-Molina. Evaluating entity resolution results. *PVLDB*, 3(1-2):208–219, 2010.

[37] K. Mo, E. Zhong, and Q. Yang. Cross-task crowdsourcing. In *KDD*, 2013.

[38] L. Mo, R. Cheng, B. Kao, X. S. Yang, C. Ren, S. Lei, D. W. Cheung, and E. Lo. Optimizing plurality for human intelligence tasks. In *CIKM*, 2013.

[39] S. Nowozin. Optimal decisions from probabilistic models: the intersection-over-union case. In *CVPR*, 2014.

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

[41] H. Park, R. Pang, A. G. Parameswaran, H. Garcia-Molina, N. Polyzotis, and J. Widom. Deco: A system for declarative crowdsourcing. *PVLDB*, 5(12), 2012.

[42] R. Pochampally, A. D. Sarma, X. L. Dong, A. Meliou, and D. Srivastava. Fusing data with correlations. In *SIGMOD*, pages 433–444, 2014.

[43] V. C. Raykar and S. Yu. Ranking annotators for crowdsourced labeling tasks. In *NIPS*, pages 1809–1817, 2011.

[44] V. C. Raykar, S. Yu, L. H. Zhao, G. H. Valadez, C. Florin, L. Bogoni, and L. Moy. Learning from crowds. *Journal of Machine Learning Research*, 2010.

[45] S. H. Rice. A stochastic version of the price equation reveals the interplay of deterministic and stochastic processes in evolution. *BMC evolutionary biology*, 8:262, 2008.

[46] A. B. Sayeed, T. J. Meyer, H. C. Nguyen, O. Buzek, and A. Weinberg. Crowdsourcing the evaluation of a domain-adapted named entity recognition system. In *HLT-NAACL*, pages 345–348, 2010.

[47] V. S. Sheng, F. J. Provost, and P. G. Ipeirotis. Get another label? improving data quality and data mining using multiple, noisy labelers. In *KDD*, 2008.

[48] A. Sheshadri and M. Lease. SQUARE: A Benchmark for Research on Computing Crowd Consensus. In *Proceedings of the 1st AAAI Conference on Human Computation (HCOMP)*, pages 156–164, 2013.

[49] J. R. Talburt. *Entity Resolution and Information Quality*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010.

[50] B. Trushkowsky, T. Kraska, M. J. Franklin, and P. Sarkar. Crowdsourced enumeration queries. In *ICDE*, pages 673–684, 2013.

[51] C. Van Rijsbergen. *Information retrieval*. Butterworths, 1979.

[52] M. Venanzi, J. Guiver, G. Kazai, P. Kohli, and M. Shokouhi. Community-based bayesian aggregation models for crowdsourcing. In *WWW*, 2014.

[53] P. Venetis and H. Garcia-Molina. Quality control for comparison microtasks. In *Proceedings of the First International Workshop on Crowdsourcing and Data Mining*, pages 15–21. ACM, 2012.

[54] P. Venetis, H. Garcia-Molina, K. Huang, and N. Polyzotis. Max algorithms in crowdsourcing environments. In *WWW*, pages 989–998, 2012.

[55] R. Vernica, M. J. Carey, and C. Li. Efficient parallel set-similarity joins using mapreduce. In *SIGMOD*, pages 495–506. ACM, 2010.

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

[57] J. Wang, S. Krishnan, M. J. Franklin, K. Goldberg, T. Kraska, and T. Milo. A sample-and-clean framework for fast and accurate query processing on dirty data. In *SIGMOD*, pages 469–480. ACM, 2014.

[58] J. Wang, G. Li, and J. Feng. Can we beat the prefix filtering?: an adaptive framework for similarity join and search. In *SIGMOD*, pages 85–96, 2012.

[59] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng. Leveraging transitive relations for crowdsourced joins. In *SIGMOD*, 2013.

[60] S. E. Whang, P. Lofgren, and H. Garcia-Molina. Question selection for crowd entity resolution. *PVLDB*, 6(6):349–360, 2013.

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

[62] C. J. Zhang, L. Chen, H. V. Jagadish, and C. C. Cao. Reducing uncertainty of schema matching via crowdsourcing. *PVLDB*, 6(9):757–768, 2013.

[63] B. Zhao, B. I. Rubinstein, J. Gemmell, and J. Han. A bayesian approach to discovering truth from conflicting sources for data integration. *PVLDB*, 2012.

APPENDIX

A. QASCA ARCHITECTURE

QASCA contains four components, namely APP Manager, Web Server, Task Assignment and Database component in all. To deploy an application, a requester has to configure the three files in APP Manager. We first introduce the four components, and then show how to deploy an entity resolution application in QASCA.

APP Manager: To deploy n questions in QASCA, the requester first needs to create an application folder in APP Manager, which consists of three files: (1) Question File is a JSON-format file which stores the set of n questions and their possible labels; (2) UI Template File is used to render k questions as a user understandable HIT in HTML templates; (3) Configuration File contains all required information about the application, including the number of questions in each HIT (k), the amount of money paid for each HIT (b), the total budget (B), and the evaluation metric.

Web Server: Web Server processes the requests from workers in crowdsourcing platform (AMT). If a worker requests a HIT, Web Server calls Task Assignment, which dynamically generates a HIT and assigns it to the worker; if a worker completes a HIT, Web Server updates the answer set D and parameters (including prior and worker model) in Database.

Table 5: Table of Notations

Symbol	Description
Question Model	
q_i	The i-th question ($1 \le i \le n$)
L_j	The j-th label ($1 \le j \le \ell$)
t_i	The index of true label for q_i ($1 \le i \le n$), $1 \le t_i \le \ell$
D_i	Answer set for question q_i ($1 \le i \le n$)
S	Questions set: $S = \{q_1, q_2, \ldots, q_n\}$
D	Answer set for all questions: $D = \{D_1, D_2, \ldots, D_n\}$
Q^c	Current distribution matrix (size $n \times \ell$ matrix)
Task Assignment	
k	The number of questions per HIT
S^w	Candidate questions set for worker w
Q^w	Estimated distribution matrix for worker w (size $n \times \ell$ matrix)
X	Assignment vector ($1 \times n$), where each element $x_i = \{0, 1\}$
Q^X	Assignment distribution matrix for X (size $n \times \ell$ matrix)
R	Result vector ($1 \times n$), where each element $1 \le r_i \le \ell$
a_i^w	The index of answered label by worker w for q_i
Parameters	
m^w	Worker Probability (WP) for worker w
M^w	Confusion Matrix (CM) for worker w (size $\ell \times \ell$ matrix)
p_j	Prior probability for label L_j ($1 \le j \le \ell$)

Task Assignment: Task Assignment is the core component in QASCA. When a worker requests a HIT through Web Server, based on the question model and the worker model stored in Database, Task Assignment identifies k questions for the worker by considering the evaluation metric specified in APP Manager. Then it dynamically creates a HIT consisting of the identified k questions, and assigns the HIT to the worker via Web Server. The online task assignment problem is formally defined in Section 2.

Database: Database is the component that stores tables containing question and worker model. When a worker requests a HIT through Web Server, tables are queried by Task Assignment; when a worker completes a HIT, tables are updated by Web Server. After all HITs are completed, Database returns the result of each question based on the question model and the evaluation metric.

Suppose a requester wants to deploy an entity resolution application on QASCA and the application has generated $n = 1000$ questions where each question has the labels L_1 ="equal" and L_2 ="non-equal". The requester first creates an application folder in the APP Manager component. In the created folder, the requester (1) deploys the questions as JSON-format in the Questions File, (2) specifies the HTML template in the UI Template File, (3) indicates in the Configuration File that each HIT contains $k = 10$ questions and is paid $b = \$0.02$, and the total invested budget is $B = \$7$, and the evaluation metric is set as F-score for "equal" with α=0.5.

When a worker requests a HIT, Web Server acquires worker-id from AMT and passes it to Task Assignment, which identifies $k = 10$ questions based on the specified evaluation metric (F-score for "equal" with α=0.5) in APP Manager, and returns a HIT containing the identified questions to the worker. When a worker completes a HIT, Web Server updates the answer set and parameters.

The total number of HITs is denoted as $m = B/b = 350$. After obtaining the answers of all $m = 350$ HITs, QASCA terminates and returns the derived result for each question based on considering the question model (stored in Database) and the evaluation metric (F-score for "equal" with $\alpha = 0.5$).

B. SUMMARY OF NOTATIONS

We present the summary of notations in Table 5.

C. PROOF FOR THEOREM 1

Theorem 1: For Accuracy*, the optimal result r_i^* ($1 \le i \le n$) of a question q_i is the label with the highest probability, i.e., $r_i^* = \arg\max_j \ Q_{i,j}$.

PROOF. We prove the theorem by proof of contradiction. Suppose the theorem does not hold. Then in the optimal result vector $R^* = [r_1^*, r_2^*, \ldots, r_n^*]$, there exists an index t ($1 \le t \le n$), such that $r_t^* \ne \arg\max_j \ Q_{t,j}$. So we can construct a result vector $R' = [r_1', r_2', \ldots, r_n']$ where $r_t' = \arg\max_j \ Q_{t,j}$ and $r_i' = r_i^*$ for $i \ne t$. Then we have Accuracy$^*(Q, R')$ − Accuracy$^*(Q, R^*)$ = $(Q_{t,r_t'} - Q_{t,r_t^*})/n > 0$, which contradicts that R^* is the optimal result vector. Thus the theorem is correct. $\square$

D. PROOF FOR THEOREM 2

Theorem 2: Given Q and α, for F-score*, the optimal result r_i^* ($1 \le i \le n$) of a question q_i can be derived by comparing $Q_{i,1}$ with the threshold $\theta = \lambda^* \cdot \alpha$, i.e., $r_i^* = 1$ if $Q_{i,1} \ge \theta$ and $r_i^* = 2$ if $Q_{i,1} < \theta$.

PROOF. In the proof for Theorem 2, we assume that λ^* is known, and we try to exploit how the optimal result vector R^* can be constructed with the known λ^*. As $\lambda^* = \max_R$ F-score$^*(Q, R, \alpha)$, which means that for any $R \in \{1, 2\}^n$, the inequality $\lambda^* \ge$ F-score$^*(Q, R, \alpha)$ holds, i.e., we have $\lambda^* \ge \frac{\sum_{i=1}^n Q_{i,1} \cdot \mathbb{1}_{\{r_i=1\}}}{\alpha \cdot \sum_{i=1}^n \mathbb{1}_{\{r_i=1\}} + (1-\alpha) \cdot \sum_{i=1}^n Q_{i,1}}$, then we can further derive

$$\sum\nolimits_{i=1}^{n} (Q_{i,1} - \lambda^* \cdot \alpha) \cdot \mathbb{1}_{\{r_i=1\}} \le \lambda^* \cdot (1-\alpha) \cdot \sum\nolimits_{i=1}^{n} Q_{i,1}. \quad (22)$$

From another perspective, the optimal result vector R^* satisfies λ^* = F-score$^*(Q, R^*, \alpha)$, thus similarly we can derive

$$\sum\nolimits_{i=1}^{n} (Q_{i,1} - \lambda^* \cdot \alpha) \cdot \mathbb{1}_{\{r_i^*=1\}} = \lambda^* \cdot (1-\alpha) \cdot \sum\nolimits_{i=1}^{n} Q_{i,1}. \quad (23)$$

As λ^*, Q, and α are known, for ease of representation let us denote a fixed constant A and a function $h(R)$ as follows

$$\begin{cases} A = \lambda^* \cdot (1-\alpha) \cdot \sum_{i=1}^n Q_{i,1}, \\ h(R) = \sum_{i=1}^n (Q_{i,1} - \lambda^* \cdot \alpha) \cdot \mathbb{1}_{\{r_i=1\}}. \end{cases}$$

Then Equation 22 and 23 can be represented as: (1) for any $R \in \{1, 2\}^2$, $h(R) \le A$; and (2) the optimal R^* satisfies $h(R^*) = A$. Next we prove that if we can derive $R' = \arg\max_R \{ \ h(R) \ \}$, then $R' = [r_1', r_2', \ldots, r_n']$ is the optimal result vector (i.e., $R' = R^*$). From Equation 22, since $R' \in \{1, 2\}^2$, we can derive $h(R') \le A$.

From $R' = \arg\max_R \{ \ h(R) \ \}$, we know that $\max_R\{ \ h(R) \ \} = h(R')$ and from Equation 23, we know that $\max_R\{ \ h(R) \ \} \ge h(R^*) = A$. So we have $h(R') \ge A$. As $h(R') \le A$, we have $h(R') = A$, or $\sum_{i=1}^n (Q_{i,1} - \lambda^* \cdot \alpha) \cdot \mathbb{1}_{\{r_i'=1\}} = \lambda^* \cdot (1-\alpha) \cdot \sum_{i=1}^n Q_{i,1}$, and finally we can derive λ^* from the above Equation: $\lambda^* = \frac{\sum_{i=1}^n Q_{i,1} \cdot \mathbb{1}_{\{r_i'=1\}}}{\alpha \cdot \sum_{i=1}^n \mathbb{1}_{\{r_i'=1\}} + (1-\alpha) \cdot \sum_{i=1}^n Q_{i,1}}$.

As λ^* = F-score$^*(Q, R', \alpha)$, i.e., R' derives the optimal λ^*, we know $R' = R^*$. Then $R^* = \arg\max_R\{ \ h(R) \ \}$, i.e., $R^* = \arg\max_R \ \big\{ \sum_{i=1}^n (Q_{i,1} - \lambda^* \cdot \alpha) \cdot \mathbb{1}_{\{r_i=1\}} \big\}$. In order to maximize $h(R)$, we can set $\mathbb{1}_{\{r_i=1\}} = 1$ (or $r_i = 1$) if $Q_{i,1} \ge \lambda^* \cdot \alpha$ and $\mathbb{1}_{\{r_i=1\}} = 0$ (or $r_i = 2$) if $Q_{i,1} < \lambda^* \cdot \alpha$. Then we get

$$r_i^* = \begin{cases} 1 & \text{if} \quad Q_{i,1} \ge \lambda^* \cdot \alpha, \\ 2 & \text{if} \quad Q_{i,2} < \lambda^* \cdot \alpha. \end{cases}$$

Thus we have proved that there exists a threshold $\theta = \lambda^* \cdot \alpha$, such that $R^* = [\ r_1^*, r_2^*, \ldots, r_n^* \ ]$ can be constructed as $r_i^* = 1$ if $Q_{i,1} \ge \lambda^* \cdot \alpha$ and $r_i^* = 2$ if otherwise. $\square$

E. COMPUTING THE QUALITY OF Q FOR F-SCORE (ALGORITHM 1)

For a given Q, In order to derive the optimal λ^* such that $\lambda^* = \max_R$ F-score$^*(Q, R, \alpha)$, following the discussions in Section 3.2.3, we design Algorithm 1. It iteratively updates λ until convergence. Let λ_t denote the λ for the t-th iteration, so initially $\lambda_1 = \lambda_{init} = 0$. In the t-th iteration (λ_t is known), it first constructs a new result vector R' using the known λ_t (lines 5-7) and

Algorithm 1 *Measure the Quality of Q for F-score*

```
Input:  $Q$, $\alpha$
Output: $\lambda$
1:  $\lambda = 0$ ; // initialized as 0 ($\lambda_{init} = 0$)
2:  $R' = [\,]$ ;
3:  while True do
4:      $\lambda_{pre} = \lambda$; // record $\lambda$ for this iteration
5:      // construct new $R' = [r'_1, r'_2 \ldots r'_n]$
6:      for $i = 1$ to $n$ do
7:          if $Q_{i,1} \geq \lambda \cdot \alpha$ then $r'_i = 1$ else $r'_i = 2$
8:      $\lambda = \frac{\sum_{i=1}^{n} Q_{i,1} \cdot \mathbb{1}_{\{r'_i=1\}}}{\sum_{i=1}^{n}[\,\alpha \cdot \mathbb{1}_{\{r'_i=1\}} + (1-\alpha) \cdot Q_{i,1}\,]}$; // F-score$^*(Q, R', \alpha)$
9:      if $\lambda_{pre} == \lambda$ then
10:         break
11:     else
12:         $\lambda_{pre} = \lambda$
13: return $\lambda$
```

then update λ_t to $\lambda_{t+1} =$ F-score$^*(Q, R', \alpha)$ (line 8) for the next iteration. The way to construct each r'_i $(1 \leq i \leq n)$ in R' is based on comparing $Q_{i,1}$ with the threshold $\lambda_t \cdot \alpha$, i.e., $r'_i = 1$ if $Q_{i,1} \geq \lambda_t \cdot \alpha$ and $r_i = 2$ if otherwise. Finally it decides whether it converges (i.e., $\lambda_{t+1} = \lambda_t$) or not (lines 9-12).

F. PROOF FOR THEOREM 3

Theorem 3: The defined δ_{t+1} (in Definition 2) satisfies Property 1 and Property 2.

PROOF. As mentioned before, in the definition of computing δ_{t+1} (Definition 2), the construction of R^X (Equation 15) is similar to the construction of R' in choosing the optimal result vector for a given Q (Algorithm 1). Thus the basic idea of the proof is to make a comparison with Algorithm 1.

Before making the comparison, we present some theoretical results proved in [12] for the Dinkelbach framework (which applies to Algorithm 1). It has proved that starting from the initial $\lambda_{init} \leq \lambda^*$, the λ_{init} will be iteratively *increased* to λ^* until convergence. It means that the update from λ_t to λ_{t+1} in Algorithm 1 also conforms to our two properties, i.e., (1) if $\lambda_t < \lambda^*$, then $\lambda_t < \lambda_{t+1} \leq \lambda^*$ and (2) if $\lambda_t = \lambda^*$, then $\lambda_t = \lambda_{t+1} = \lambda^*$.

Recall that δ^* and X^* respectively denote the optimal value and the optimal assignment vector, and we can derive $\delta^* = \max_R$ F-score$^*(Q^{X^*}, R, \alpha)$. Thus, if Algorithm 1 takes Q^{X^*} and α as the input, then $\lambda^* = \max_R$ F-score$^*(Q^{X^*}, R, \alpha)$, which is exactly δ^*, i.e., $\lambda^* = \delta^*$.

The comparison is conducted based on comparing our online assignment algorithm with Algorithm 1, which takes Q^{X^*} and α as the input. As derived above, the optimal value for both algorithms are the same (i.e., δ^*).

To prove Property 1, suppose both algorithms start with $\delta_t < \delta^*$, and they update their respective values (denoted as δ_{t+1} and λ_{t+1} respectively) for the next iteration as follows

$$\begin{cases} \delta_{t+1} = \max_X \text{ F-score}^*(Q^X, \widehat{R}^X, \alpha), \\ \lambda_{t+1} = \text{F-score}^*(Q^{X^*}, R', \alpha). \end{cases} \tag{24}$$

Note that for a feasible X, $\widehat{R}^X$ is constructed by comparing each $Q^X_{i,1}$ $(1 \leq i \leq n)$ with the threshold $\delta_t \cdot \alpha$. And R' is similarly constructed by comparing each $Q^{X^*}_{i,1}$ $(1 \leq i \leq n)$ with the same threshold $\delta_t \cdot \alpha$. As X^* is also a feasible assignment, we have $(Q^{X^*}, R') \in \{(Q^X, \widehat{R}^X) \mid X \text{ is feasible }\}$. Then by considering Equation 24, we derive $\delta_{t+1} \geq \lambda_{t+1}$. As mentioned above, since $\delta_t < \delta^*$, the properties in [12] guarantee that $\lambda_{t+1} > \delta_t$, then we derive $\delta_{t+1} > \delta_t$. As δ^* is the optimal objective value, we can finally derive $\delta_t < \delta_{t+1} \leq \delta^*$, which proves Property 1.

To prove Property 2, we apply the same comparison. Suppose both algorithms start with $\delta_t = \delta^*$, then they are updated by Equation 24. Similarly we have $\delta_{t+1} \geq \lambda_{t+1}$. As $\delta_t = \delta^*$, following the properties in [12], we have $\lambda_{t+1} = \delta_t = \delta^*$. Since $\delta_{t+1} \leq \delta^*$, we derive $\delta_t = \delta_{t+1} = \delta^*$, which proves Property 2. □

Algorithm 2 *F-score Online Assignment*

```
Input:  $Q^c$, $Q^w$, $\alpha$, $k$, $S^w$
Output: HIT
1:  $\delta = 0$ ; // initialized as 0 ($\delta_{init} = 0$)
2:  while True do
3:      $\delta_{pre} = \delta$
4:      // get the updated $\delta_{t+1}$ and its corresponding $X$
5:      $X, \delta = Update(Q^c, Q^w, \alpha, k, S^w, \delta)$
6:      if $\delta_{pre} == \delta$ then
7:          break
8:      else
9:          $\delta_{pre} = \delta$
10: // construct HIT based on the returned $X$
11: for $i = 1$ to $n$ do
12:     if $x_i == 1$ then
13:         HIT = HIT $\cup \{q_i\}$
14: return HIT
```

G. PROOF FOR THEOREM 4

Theorem 4: The problem of computing δ_{t+1} (Definition 2) can be reduced to a 0-1 FP problem.

PROOF. Given Q^c, Q^w and a feasible X, from Equation 1 we know that Q^X can be expressed by X, Q^c and Q^w, then we have

$$Q^X_{i,1} = x_i \cdot Q^w_{i,1} + (1 - x_i) \cdot Q^c_{i,1}.$$

Given Q^X and δ_t, we know that $\widehat{R}^X$ is constructed by setting $\widehat{r}^X_i = 1$ if $Q^X_{i,1} \geq \delta_t \cdot \alpha$ and $\widehat{r}^X_i = 2$ if $Q^X_{i,1} < \delta_t \cdot \alpha$. It means that if we construct $\widehat{R}^c$ ($\widehat{R}^w$) by setting $\widehat{r}^c_i = 1$ ($\widehat{r}^w_i = 1$) if $Q^c_{i,1} \geq \delta_t \cdot \alpha$ ($Q^w_{i,1} \geq \delta_t \cdot \alpha$) and $\widehat{r}^c_i = 2$ ($\widehat{r}^w_i = 2$) if $Q^c_{i,1} < \delta_t \cdot \alpha$ ($Q^w_{i,1} < \delta_t \cdot \alpha$), then $\widehat{R}^X$ can be expressed with the given $\widehat{R}^c$ and $\widehat{R}^w$ as follows:

$$\mathbb{1}_{\{\widehat{r}^X_i = 1\}} = x_i \cdot \mathbb{1}_{\{\widehat{r}^w_i = 1\}} + (1 - x_i) \cdot \mathbb{1}_{\{\widehat{r}^c_i = 1\}}.$$

As $x_i = \{0, 1\}$, if we plug the above derived $Q^X_{i,1}$ and $\mathbb{1}_{\{\widehat{r}^X_i = 1\}}$ into F-score$^*(Q^X, \widehat{R}^X, \alpha) = \frac{\sum_{i=1}^{n} Q^X_{i,1} \cdot \mathbb{1}_{\{\widehat{r}^X_i=1\}}}{\sum_{i=1}^{n}[\,\alpha \cdot \mathbb{1}_{\{\widehat{r}^X_i=1\}} + (1-\alpha) \cdot Q^X_{i,1}\,]}$, and set the parameters b_i, d_i $(1 \leq i \leq n)$, β and γ following
(1) $b_i = d_i = 0$ for $q_i \notin S^w$, and
(2) b_i, d_i ($q_i \in S^w$), β and γ are set as:

$$\begin{cases} b_i = Q^w_{i,1} \cdot \mathbb{1}_{\{\widehat{r}^w_i=1\}} - Q^c_{i,1} \cdot \mathbb{1}_{\{\widehat{r}^c_i=1\}} \\ d_i = \alpha \cdot (\mathbb{1}_{\{\widehat{r}^w_i=1\}} - \mathbb{1}_{\{\widehat{r}^c_i=1\}}) + (1-\alpha) \cdot (Q^w_{i,1} - Q^c_{i,1}) \\ \beta = \sum_{i=1}^{n} Q^c_{i,1} \cdot \mathbb{1}_{\{\widehat{r}^c_i=1\}} \\ \gamma = \sum_{i=1}^{n}[\,\alpha \cdot \mathbb{1}_{\{\widehat{r}^c_i=1\}} + (1-\alpha) \cdot Q^c_{i,1}\,], \end{cases}$$

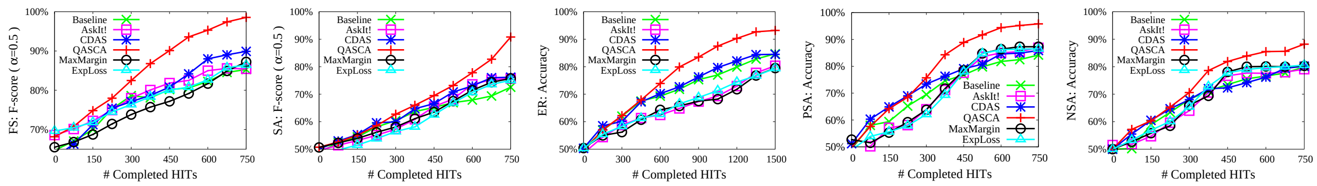
then the problem is to maximize

$$\text{F-score}^*(Q^X, \widehat{R}^X, \alpha) = \frac{\sum_{i=1}^{n}(x_i \cdot b_i) + \beta}{\sum_{i=1}^{n}(x_i \cdot d_i) + \gamma}. \tag{25}$$

s.t. X is a feasible assignment vector. As Q^c, Q^w, $\widehat{R}^c$, $\widehat{R}^w$, S^w, α, and δ_t are known, then x_i $(1 \leq i \leq n)$ are the only unknown variables. Let all the feasible assignment vectors form the subspace $\Omega \subseteq \{0,1\}^n$ where $|\Omega| = \binom{|S^w|}{k}$. If we set $z_i = x_i$ $(1 \leq i \leq n)$, then each $\mathbf{z} \in \Omega$ corresponds to a feasible assignment vector X. So the problem is to maximize $\frac{\sum_{i=1}^{n}(z_i \cdot b_i) + \beta}{\sum_{i=1}^{n}(z_i \cdot d_i) + \gamma}$ s.t. $\mathbf{z} \in \Omega \subseteq \{0,1\}^n$, which is a 0-1 FP problem. □

H. F-SCORE ONLINE ASSIGNMENT ALGORITHMS (ALGORITHM 2 AND 3)

Algorithm 2 is the *F-score Online Assignment Algorithm*, which iteratively updates δ_{init} until convergence. In each iteration (lines



(a) *FS*: F-score (b) *SA*: F-score (c) *ER*: Accuracy (d) *PSA*: Accuracy (e) *NSA*: Accuracy

Figure 7: Additional Results on Real Datasets by Varying # Completed HITs

Algorithm 3 *Update*

```
Input:  $Q^c$, $Q^w$, $\alpha$, $k$, $S^w$, $\delta$
Output:  $X, \lambda$
1: $\lambda = 0$ ; // initialized as 0 ($\lambda_{init} = 0$)
2: $X = [\,]$ ;
3: $\widehat{R}^c = [\,]$;  $\widehat{R}^w = [\,]$;
4: $b = d = [0, 0, \ldots, 0]$;  $\beta = 0$;  $\gamma = 0$;
5: // construct $\widehat{R}^c$ ($\widehat{R}^w$) by comparing $Q^c$ ($Q^w$) with $\delta \cdot \alpha$; (lines 6-9)
6: for $i = 1$ to $n$ do
7:     if $Q^c_{i,1} \geq \delta \cdot \alpha$ then $\widehat{r}^c_i = 1$ else $\widehat{r}^c_i = 2$
8: for $q_i \in S^w$ do
9:     if $Q^w_{i,1} \geq \delta \cdot \alpha$ then $\widehat{r}^w_i = 1$ else $\widehat{r}^w_i = 2$
10: Compute $b_i, d_i$ ($1 \leq i \leq n$) and $\beta, \gamma$ following the proof in Theorem 4;
11: // Update $\lambda$ from $\lambda_{init}$ until convergence; (line 12-21)
12: while True do
13:     $\lambda_{pre} = \lambda$
14:     compute $TOP$, a set which contains $k$ questions in $S^w$ that correspond to
        the highest value of $b_i - \lambda \cdot d_i$;
15:     for $i = 1$ to $n$ do
16:         if $q_i \in TOP$ then $x_i = 1$ else $x_i = 0$
17:     $\lambda = \frac{\sum_{i=1}^{n} (\ x_i \cdot b_i\ ) + \beta}{\sum_{i=1}^{n} (\ x_i \cdot d_i\ ) + \gamma}$ ;
18:     if $\lambda_{pre} == \lambda$ then
19:         break
20:     else
21:         $\lambda_{pre} = \lambda$
22: return $X, \lambda$
```

3-9), it first calls the *Update Algorithm* (the details are introduced in the following two paragraphs) to update δ (line 5), and then decide whether it converges or not (lines 6-9). Finally, it uses the assignment vector X corresponding to the converging δ to construct a HIT (lines 10-14). The converging δ and its corresponding assignment vector X are both optimal (i.e., respectively δ^* and X^*).

As we have proved that that the problem of computing λ_{t+1} can be reduced to a 0-1 FP problem (Theorem 4), following the Dinkelbach framework [12] as discussed in Section 3.2.3, the key is to solve the sub-problem $\mathbf{z}' = argmax_{\mathbf{z}} \sum_{i=1}^{n}(b_i - \lambda \cdot d_i) \cdot z_i$. In Theorem 4, in order to get $\mathbf{z}'$, due to the constraint of Ω (containing all feasible assignments), we should select k questions in S^w ($q_i \in S^w$) with the largest values of $(b_i - \lambda \cdot d_i)$.

We present the detailed *Update Algorithm* in Algorithm 3, which leverages Dinkelbach framework [12] to efficiently compute δ_{t+1} based on δ_t. For efficiency's sake it first constructs $\widehat{R}^c$, $\widehat{R}^w$ (lines 5-9), and b_i, d_i ($b_i \in S^w$), β, γ (line 10) following the details in Theorem 4 . Then it iteratively updates λ_{init} until convergence (lines 11-21). In each iteration (lines 13-21), with known λ, it selects k questions in S^w ($q_i \in S^w$) with the largest values of $(b_i - \lambda \cdot d_i)$ (line 14), which needs $\mathcal{O}(n)$ time by PICK algorithm [2]. Then it updates λ following Equation 25 (line 17). Finally it decides whether it converges or not (lines 18-21).

I. ADDITIONAL RESULTS

In order to show both Accuracy and F-score on all datasets, we add five additional experiments, and the evaluation metrics are respectively F-score for datasets *FS* and *SA*, and Accuracy for datasets *ER*, *PSA* and *NSA*. To be precise, for *FS*, we set the evaluation metric as F-score for label "$\geq$" with $\alpha = 0.5$; and for *SA*, we set the evaluation metric as F-score for label "positive" with $\alpha = 0.5$. Apart from the change of evaluation metrics, we apply the same experimental settings to the new experiments as our original ones (Table 1). We calculate the result quality for different systems with the number of completed HITs in Figure 7. The new experiments have similar results with our original ones and QASCA outperforms other systems on all datasets.

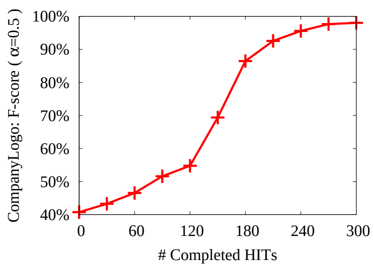


Figure 8: F-Score on CompanyLogo Dataset with 214 Labels

J. EXPERIMENT ON MULTIPLE LABELS

We evaluate the efficiency and effectiveness of our model for F-score on a real-world dataset with a large number of labels. The dataset was constructed using the logos of Fortune 500 companies[11], denoted by the "CompanyLogo" dataset. It had 500 questions in total, where each question contained a company logo, and we asked workers to decide in which country the company was founded. There were 214 labels (i.e., categories) in a question, where each label corresponded to a country. Each question was assigned to 3 workers on average and each HIT contained 5 questions. So there were $\frac{500 \times 3}{5} = 300$ HITs. We deployed the application on our system QASCA and set the evaluation metric as F-score for label "USA" with $\alpha = 0.5$. There were 128 out of 500 questions whose true labels corresponded to "USA". We recorded the result quality with different numbers of completed HITs and the maximal assignment time among all HITs.

The experimental results (see Figure 8) showed that QASCA achieved 90% F-score by completing only two thirds of the HITs. That is, each question got only 2 answers on average from the crowd. This result validated the effectiveness of our assignment model for a large number of categories. Moreover, the maximal assignment time was 0.005s, and thus our method was also fairly efficient. This is because F-score focuses on the quality of a specific label (or a certain group of labels), and we can optimize its calculation by reducing a large number of labels into two categories, where one category (called "*target label*") contains the focused label and the other category (called "*non-target label*") includes the remaining labels. In our example, the "target label" was "USA", and all the other labels formed the "non-target label". Therefore, the calculation of F-score was actually independent of the number of categories. If requesters focus on the quality of multiple labels, they can use Accuracy instead of F-score.

[11] http://en.wikipedia.org/wiki/Fortune_Global_500