Incentive-Based Entity Collection using Crowdsourcing

Chengliang Chai† Ju Fan‡ Guoliang Li†
† *Department of Computer Science, Tsinghua University, China*
‡ *DEKE Lab & School of Information, Renmin University of China, China*
chaicl15@mails.tsinghua.edu.cn, liguoliang@tsinghua.edu.cn,fanj@ruc.edu.cn

***Abstract*—Crowdsourced entity collection leverages human's ability to collect entities that are missing in a database, which has many real-world applications, such as knowledge base enrichment and enterprise data collection. There are several challenges. First, it is hard to evaluate the workers' quality because a worker's quality depends on not only the correctness of her provided entities but also the distinctness of these entities compared with the collected ones by other workers. Second, crowd workers are likely to provide popular entities and different workers will provide many duplicated entities, leading to a waste of money and low coverage.**

To address these challenges, we propose an incentive-based crowdsourced entity collection framework `CrowdEC` that encourages workers to provide more distinct items using an incentive strategy. `CrowdEC` has fundamental differences from existing crowdsourcing collection methods. One the one hand, `CrowdEC` proposes a worker model and evaluates a worker's quality based on cross validation and entity checking. `CrowdEC` devises a worker utility model that considers both worker's quality and entities' distinctness provided by workers. `CrowdEC` proposes a worker elimination method to block workers with a low utility, which solves the first challenge. On the other hand, `CrowdEC` proposes an incentive pricing technique that encourages each qualified (i.e., non-eliminated) worker to provide distinct entities rather than duplicates. `CrowdEC` provides two types of tasks and judiciously assigns workers with appropriate tasks to address the second challenge. We have conducted both real and simulated experiments, and the results show that `CrowdEC` outperforms existing state-of-the-art works on both cost and quality.

I. Introduction

Entity collection is an important operation in many applications, such as knowledge base enrichment and enterprise data collection. For example, a knowledge base aims to collect all the entities in a category, e.g., collecting all active NBA players. Google map wants to collect all points of interest. It is easy to get the popular entities, e.g., Lebron James, but it is rather hard to get the unpopular entities which do not frequently appeared due to the long tail phenomenon. Thus crowdsourced entity collection [25] is proposed that harnesses the crowd (aka workers) to collect entities. For example, we can ask crowd workers to answer a question like "Please give us an active NBA player". There are several major challenges in crowdsourced entity collection.

(1) Quality control for crowdsourced entity collection. It is important to select high-quality workers, however it is hard to evaluate the workers' quality on crowdsourced entity collection. The reasons are two-fold. First, it is hard to detect whether an entity provided by a worker is correct, leading to low quality. Traditional methods use golden test (e.g., tasks with ground truth) to evaluate a worker's quality. However, entity collection is an open-world task and we cannot provide golden test (even if we can provide some known entities, the workers may not return them as answers.). Second, a worker's quality depends on not only the correctness of her provided entities but also the distinctness of these entities compared with the collected ones by other workers.

(2) Cost control for crowdsourced entity collection. Workers usually provide popular entities and thus different workers will provide many duplicated entities, leading to a waste of money and low coverage. For example, many workers will return Lebron James but few workers return Patrick McCaw.

(3) Termination for crowdsourced entity collection. It is important to decide when to stop the crowdsourcing task. If the task terminates too early, the coverage of the collected entities is low. If it terminates too late, it will incur high cost.

Trushkowsky et al. [25] addressed the third challenge by evaluating the completeness but did not address the first two challenges. In this work we propose an incentive-based crowdsourced entity collection framework `CrowdEC` that selects high-quality workers and encourages them to provide more distinct items using an incentive strategy. `CrowdEC` has fundamental differences from existing studies. On the one hand, `CrowdEC` proposes a worker model and evaluates a worker's quality based on cross validation and entity checking. Cross validation evaluates an entity's correctness by checking whether it is also provided by other workers while entity validation asks others workers to check whether an entity is correct. `CrowdEC` devises a worker utility model that considers both worker's quality and distinctness of entities provided by workers. `CrowdEC` proposes a worker elimination method to block workers with a low utility, which solves the quality control problem. On the other hand, `CrowdEC` proposes an incentive pricing technique that encourages each qualified worker to provide distinct entities rather than duplicates. Since the crowdsourcing platforms do not support dynamic pricing, `CrowdEC` uses bonus to encourage a worker. `CrowdEC` provides two types of tasks for workers: normal task without bonus and task with bonus. For the former task, `CrowdEC` only provides a base reward. For the latter, if the worker provides a distinct entity that is not provided by other workers, `CrowdEC` gives a bonus to the worker. `CrowdEC` judiciously assigns workers with appropriate tasks, which solves the cost control problem.

To summarize, we make the following contributions.

(1) We propose an incentive-based crowdsourced entity collection framework `CrowdEC` that first eliminates unqualified workers and encourages qualified workers to provide more distinct items using an incentive strategy (see Section III).

(2) We propose a quality control method for crowdsourced

entity collection that evaluates workers' quality based on cross validation and entity checking (see Section IV). We devise a worker utility model that considers both worker's quality and distinctness of entities provided by workers. We design a worker elimination method to block low-utility workers.
(3) `CrowdEC` proposes an incentive pricing technique, provides two types of tasks for workers, and judiciously assigns workers with appropriate tasks (see Section V).
(4) We have conducted both real and simulated experiments, and the results show that `CrowdEC` outperforms existing state-of-the-art works on both cost and quality (see Section VI).

II. Preliminaries

We formulate the problem in Section II-A, and discuss the completion estimation technique in Section II-B.

A. Problem Formulation

Our work studies the *entity collection* problem that collects all the distinct entities in a particular data domain, e.g., collecting all the active NBA players, where the ground-truth of entities in the domain is unknown before collection. For simplicity, we use Ω to represent the ground-truth entity set.

In this paper, we focus on utilizing *crowdsourcing*, i.e., the workers on crowdsourcing platforms, such as Amazon Mechanical Turk (AMT), for entity collection, so as to leverage the advantages of human's collection. Without loss of generality, each crowdsourcing task t asks workers to submit one or more entities satisfying user's requirement, e.g., "*Please give us one more active NBA player*"[1]. We use a set $\mathcal{T}$ to denote the tasks answered by the workers, and a multiset $\mathcal{R}$ to denote the entities collected through the tasks in $\mathcal{T}$. Note that $\mathcal{R}$ may contain the same entities answered in multiple times, since different workers may contribute the same entities. We further use $\mathcal{O}$ to denote the *distinct entities* in $\mathcal{R}$. On the other hand, crowdsourcing is not free and each task t incurs monetary cost. Based on these notations, we define *crowdsourced entity collection* as follows.

Definition 1 (Crowdsourced Entity Collection): Given a data domain, it solicits crowd workers to collect a multiset $\mathcal{R}$ of entities that satisfies the following objectives.
1) *Correct*: we want to collect more *correct* entities included in Ω, i.e., maximizing the *precision* $|\mathcal{O} \cap \Omega|/|\mathcal{O}|$.
2) *Complete*: we aim to collect as many entities in Ω as possible, i.e., maximizing the *recall* $|\mathcal{O} \cap \Omega|/|\Omega|$.
3) *Less-Duplicated*: we do not want $\mathcal{R}$ having many duplicates, i.e., maximizing the *distinctness* $|\mathcal{O}|/|\mathcal{R}|$, because it not only incurs higher collection cost, but also requires the users to spend more efforts to remove the duplicates.

Consider our example of collecting active NBA players. Suppose that there are two workers who answer our tasks for entity collection: one worker submits $\{\texttt{James}, \texttt{Paul}\}$ while the other worker submits $\{\texttt{Carter}, \texttt{James}\}$. Then, the entity multiset $\mathcal{R}$ =$\{$`James`, `Paul`,`Carter`, `James`$\}$ and the distinct entity set will be $\mathcal{O} = \{\texttt{James}, \texttt{Paul}, \texttt{Carter}\}$.

There are several challenges in crowdsourced entity collection. The first is to estimate the completeness of the collected entities and we can use existing estimation technique to address this problem (Section II-B). The second is to guarantee the correctness of the collected entities, which may affect both the completion estimation and the collection cost. Moreover, workers can provide any "open" entities and traditional majority-voting based techniques do not work since different workers may provide different entities. We propose worker-quality estimation and entity validation techniques to address this problem (Section IV). The third is to reduce duplicated entities and we propose incentive pricing technique to encourage workers to provide distinct entities (Section V).

[1]We will introduce other question types later.

B. Completion Estimation

A previous work [25] on crowdsourced entity collection formalizes the problem as *species estimation*, which is well studied in biology and statistics. The basic idea is to consider that workers won't provide incorrect entities and model each worker as a *sampling process* from the underlying data domain Ω, where the entities in Ω have unknown probabilities $\{p_1, p_2, ..., p_{|\Omega|}\}$ to be sampled. Based on this worker modeling, the work in [25] focuses on estimating whether the sampled set $\mathcal{R}$ of entities is complete, i.e., $\mathcal{O} = \Omega$. For ease of presentation, we call this problem *completion estimation*.

As the work assumes that workers won't make mistakes, the key problem becomes how to estimate cardinality of the Ω, i.e., $|\Omega|$, since Ω is unknown to us. For simplicity, we also denote the estimate of $|\Omega|$ as $\widehat{N}$. Fortunately, we have the Chao92 [3] estimator. Intuitively, Chao92 first estimates $|\Omega|$ based on the sample coverage, denoted by SC. The idea is that, if we know, say 50% distinct entities have been collected, then $\widehat{N}$ can be easily computed as $\widehat{N} = |\mathcal{O}|/SC$. However, the obstacle is that even the the sample coverage SC is very difficult to estimate. To this end, the work in [25] employs the Good-Turing estimator [13] as

$$\hat{SC} = 1 - \frac{f_1}{|\mathcal{R}|}, \tag{1}$$

where f_i is the number of distinct entities that occurs exactly i times in $\mathcal{R}$. For example, f_1 counts the number of "singleton" entities in $\mathcal{R}$. The intuition of Equation (1) is that the smaller the f_1 is, the more likely that fewer new entities will be sampled from the workers in the future.

On the other hand, Chao92 considers that the underlying sampling probabilities $\{p_1, p_2, ..., p_{|\Omega|}\}$ are not evenly distributed. To address this problem, Chao92 uses the coefficient of variance of the distribution γ to improve the estimation. Specifically, it uses the following equation to estimate γ,

$$\hat{\gamma^2} = \max\{\frac{\frac{|\mathcal{O}|}{\hat{SC}}\sum_i i(i-1)f_i}{|\mathcal{R}|(\|\mathcal{R}|-1)} - 1, 0\} \tag{2}$$

Based on both $\hat{\gamma^2}$ and $\hat{SC}$, Chao92 estimates $\widehat{N}$ by combining these two factors using

$$\widehat{N} = \frac{|\mathcal{O}|}{\hat{SC}} + \frac{|\mathcal{R}|(1-\hat{SC})}{\hat{SC}}\gamma^2. \tag{3}$$

Trushkowsky [25] proposed an approach for crowd enumeration to estimate $\widehat{N}$ based on the Chao92 method. So we use their method to estimate $\widehat{N}$. Based on the estimate, if $\widehat{N}$=$|\mathcal{O}|$,

Notation	Description
Ω	the ground-truth of entity set
$\mathcal{W}$	set of workers
$\mathcal{R}_j$	set of worker w_j answers
$\mathcal{R}$	set of collected answers
E_j	the number of errors of w_j
v_j	throughput of w_j
q_j	w_j's probability of changing another answer
p_j	w_j's probability of requesting a task
$D_{\mathcal{W}}$	the error-bounded distinctness of $\mathcal{W}$

TABLE I
NOTATIONS USED IN THIS PAPER.

we terminate the collection process, as all distinct entities in Ω have been collected.

Note that the above estimation methods do not consider that workers may provide wrong entities (that are not in Ω), which would lead to a larger f_1 and thus overestimation of $\hat{N}$. We address this issue by using our quality control techniques that validate the entities during the collection time in Section IV.

For ease of presentation, we summarize notations used in this paper in Table I where some will be introduced later.

C. Related Work

Recently some works on crowdsourcing entity collection [25], [21], [9], [24], [4] have close relationship with our work. Fan et al. [9] proposed a distribution-aware crowdsourced entity collection framework. It aims to leverage the crowd to provide a set of entities and minimize the difference of entity distribution between what the crowd collected and an expected distribution. They focused on how to select workers adaptively to achieve the expected distribution. Park and Widom presented a system, `CrowdFill` [21], which leverages the crowd's ability to collect structured data. `CrowdFill` asked workers to work collaboratively to contribute new entities, fill some empty cells and upvote or downvote other workers' answers. Although `CrowdFill` also considers quality control (upvote or downvote) and pricing, the techniques in `CrowdFill` focus on interface design while we focus on cost optimization. Trushkowsky et al. used statistical methods to reason about the completeness of collected entities in a domain. They focused on how to estimate the cardinality of the domain size accurately based on crowdsourcing collection [25]. Chung et al. [4] utilized the statistic techniques to estimate the impact of the unknown data on aggregate queries like `SUM`, `AVG`, `MAX`, etc. They focused on overlap between different data sources to estimate the missing data. Rekatsinas et al. [24] focused on collecting entities from structured domains, which consists of attributes with hierarchical structure. They aimed to maximize the number of collected entities within a monetary budget.

The key difference of our framework from these existing ones is two-fold. Firstly, the problem settings are different. Duplicated answers are very common in the crowdsourcing collection task, which results in huge costs. We address this problem in an incentive way but other works ignore this phenomenon. Secondly, the optimization goals are different. They mostly aimed to estimate the cardinality accurately. We aim to spend the minimum cost to collect all entities. Moreover, we also focus on maximizing the workers' utility and controlling the quality. Although `CrowdFill` also considers quality control (upvote or downvote) and pricing, the techniques in `CrowdFill` focus on interface design while we focus on cost optimization.

Recently crowdsourced data management has become an area of increasing interest in research and industry [17]. In order to encapsulate the complexities of interacting with the crowd, several crowdsourced database systems, e.g., Deco [20], Qurk [19], CrowdDB [11], CrowdOp [10] and CDB [16] were proposed. Besides, many studies leverage crowd's ability to improve database operators like crowdsourced join [2], [8], [12], [27], crowdsourced selection [1], crowdsourced max [26], [14], crowdsourced sort [29]. Since the crowd may make mistakes, many techniques are proposed to improve the quality [7], [18], [30], [15], [22]. Compared with these studies, we focus on collecting distinct and high quality entities with a minimum cost, which is not well studied in existing works.

Qiu et al. [23] studied the incentive and pricing mechanisms for improving crowdsourcing utility. Different from that work, we focus more on collecting a complete entity set with less duplicates, which is not the crowdsourcing objectives of [23]. Moreover, we adopt different crowdsourcing settings. In [23], all workers finish one task before any move to the next (i.e., "coordination of tasks"). On the contrary, the coordination of tasks is not realistic in our setting due to the dynamic worker set in most real crowdsourcing platforms.

III. INCENTIVE-BASED CROWDSOURCING

Our incentive-based crowdsourced entity collection framework is illustrated in Figure 1: it interacts with the workers through a crowdsourcing platform, such as AMT, in the following way. At each time a worker requests for a new task, the framework first examines if the worker is qualified to provide "beneficial" entities using a WORKER CHECKING component. If not, it blocks the worker by not assigning any tasks to the worker. For a qualified worker who is not blocked, the framework applies a TASK MANAGEMENT component to assign the worker a task and collect the entities submitted by the worker. In particular, instead of using a fixed price for each task, it determines the pricing scheme, e.g., whether giving bonus to encourage the worker to submit more entities in the task (`Bonus` or `NoBonus`). Moreover, a COMPLETION ESTIMATION component continuously tracks progress of the collection (see Section II-B), and terminates the framework if it estimates that all distinct entities are collected, i.e., $\mathcal{O} = \Omega$.

To support the framework, we develop two incentive techniques, *worker elimination* and *incentive pricing*.

Worker Elimination. This technique estimates the *utility* of workers, so as to eliminate the ones with limited utility and therefore avoid monetary incentives. To facilitate our collection objective, we consider two factors that constitute worker utility. The intuition is that we prefer a worker set that is more likely to provide correct and distinct entities.

(1) *entity distinctness*: We focus on retrieving a *complete* set of entities with *less duplicates*. Thus, it is desirable to encourage workers to provide *distinct* entities, which are not yet collected. First, it would lead to faster completion

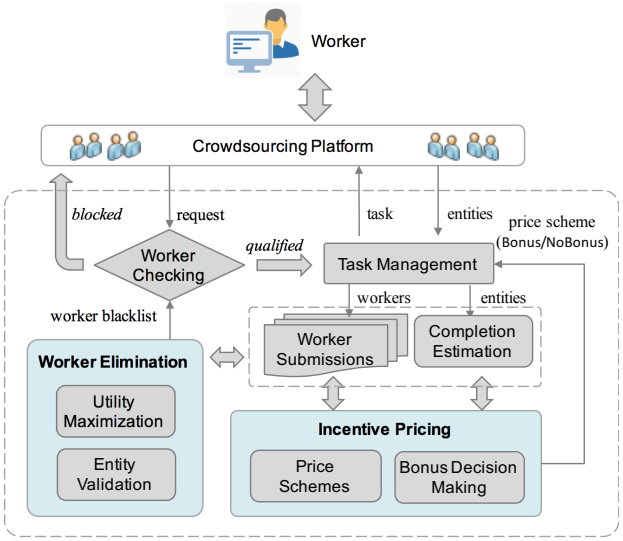

Fig. 1. Incentive-based crowdsourcing.

convergence. Second, it will also reduce the duplicates in the result, thus incurring less crowdsourcing cost.

(2) *worker quality*: we also consider *worker quality* in the proposed *utility* to improve the *correctness* of the result set $\mathcal{R}$, as some bad workers may give wrong entities.

For new workers, there is a "cold-start" problem that we may not have enough information to judge worker quality and entity distinctness. We address the problem by not considering a worker in WORKER ELIMINATION until she has contributed enough number of entities larger than a threshold.

To this end, as illustrated in Figure 1, worker elimination takes as input the workers' entity submissions and estimations of completion status. Based on the inputs, it selects a set of workers such that the utility of the worker set is maximized, and then inserts the workers not in the set into the blacklist. To support utility computation, we also devise the following two techniques: 1) the first technique is entity validation that facilitates the computation of worker's quality. The idea is to estimate the accuracy of each submitted entity from a worker by either checking if the entity can be cross-validated by other workers or publishing a new crowdsourcing task for entity validation. 2) The second technique is entity resolution that verifies whether two answers from different workers actually refer to the same entity. In this paper, we consider that entity resolution has been addressed manually or by the existing techniques [6], which is orthogonal to this paper. Even though the utility is intuitive, the problem of utility maximization is NP-hard. Thus, we introduce effective heuristic algorithms to solve the problem. More details on entity validation and utility maximization can be referred in Section IV.

Incentive Pricing. A straightforward solution to obtain more distinct entities is to publish more entity collection tasks. However, this will incur much more cost. For further reducing the cost, we propose a novel *incentive pricing* technique to encourage each qualified worker to provide distinct entities rather than duplicates.

Our technique is based on a *bonus-based incentive* mechanism that offers *bonus* to a worker if she submits distinct entities, which are essential to `CrowdEC` with respect to both

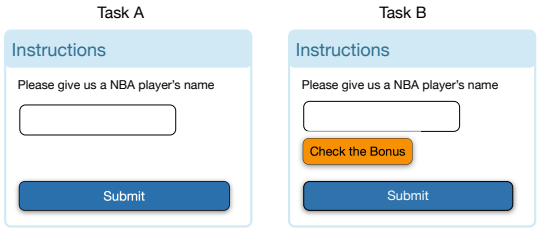

Fig. 2. Interfaces designed for pricing schemes.

complete and less-duplicated properties. Moreover, the reason that this paper does not use a more generalized and dynamic pricing scheme is due to the following two reasons. First, most crowdsourcing platforms, such as AMT and CrowdFlower, do not support any dynamic pricing schemes. Instead, they support a *bonus* mechanism to allow the requester to give extra rewards to some workers afterward[2]. Second, given that worker set on these platforms is very *dynamic*, we use the bonus to encourage a good worker to keep working for us, thus increasing the chance of providing *distinct entities* per unit cost. We will leave varying the bonus or more generalized pricing schemes as a future work. More specifically, our framework supports the following pricing schemes:

1) `NoBonus`: It gives a worker only one chance to submit an entity. No matter whether the entity is distinct or not, it will always give the worker the base reward.

2) `Bonus`: It gives a worker multiple chances to submit more than one entities. After each submission, it checks whether the entity is distinct. If so, it gives the worker the base reward as well as an extra bonus. Otherwise, it reminds the worker of the duplication and asks her to change her answer by submitting a new entity. If all the submissions fail to provide a distinct entity, the scheme only gives the worker the base reward. As the bonus is typically smaller than the base reward, the scheme has the potential to collect more entities using less cost.

For example, Figure 2 illustrates the interfaces designed for the two pricing schemes. The left interface supports the `NoBonus` scheme: a worker inputs an entity in the text box, submits the entity, and gets the base reward. On the other hand, the right interface supports the `Bonus` scheme: a worker can click the "Check the Bonus" button each time after she inputs an entity. The worker can click the button in multiple times to try getting the bonus (i.e., providing a distinct entity).

`NoBonus` suits for the workers who always submit distinct entities while `Bonus` suits for the workers who want to get more award by trying more entities. Our framework makes a decision that which pricing scheme (`Bonus` or `NoBonus`) should be provided each time a worker requests for a task. We formulate this as an optimization problem, and devise a lightweight method with good approximate property to solve the problem. Section V gives the details of incentive pricing.

Incentive-Based Framework. The pseudo-code of our framework is shown in Algorithm 1. The algorithm takes as input a small number ϵ for determining when to terminate and outputs a set $\mathcal{O}$ of distinct entities submitted by the crowd workers. After initializing worker set $\mathcal{W}$, entity multiset $\mathcal{R}$

[2]See http://docs.aws.amazon.com/ for details about awarding bonus on the AMT platform.

Algorithm 1: Incentive-Based Crowdsourcing

Input: ϵ: A small number
Output: $\mathcal{O}$: A set of distinct entities.

```
1  Initialize a worker set W ← ∅ ;
2  Initialize an entity multiset R ← ∅, entity set O ← ∅;
3  for each request from worker w do
4      N̂ ← EstimateCompletion (R) ;
5      if |N̂ − |O|| < ϵ × N̂ then break ;
6      if w ∉ W then W ← W ∪ {w} ;
7      W' ← MaxWorkerUtility(W, R) ;
8      if w ∉ W' then Block worker w ;
9      else
10         b ← DetermineBonus (w) ;
11         if b = 0 then
12             Assign an HIT t with NoBonus to w ;
13         else
14             Assign an HIT t with Bonus to w ;
15         R_t ← SubmitEntities (w, t);
16         R'_t ← ValidateEntities (R_t) ;
17         Insert R'_t into R ;
18         O ← ResolveEntities (R);
19     return O ;
```

~~and entity set $\mathcal{O}$, it processes workers' requests iteratively.~~ At each request from worker w, it first computes the current estimate $\widehat{N}$ for the size of entity domain Ω (line 4). If the difference between the number of distinct entities collected so far and $\widehat{N}$ is smaller than a small number $\epsilon \times \widehat{N}$, i.e., $\left|\widehat{N} - |\mathcal{O}|\right| < \epsilon \times \widehat{N}$, the algorithm terminates and returns $\mathcal{O}$. Otherwise, the algorithm first selects workers by utility maximization and obtains worker set $\mathcal{W}'$ (line 7). Then, it blocks worker w if the worker is not selected, i.e., $w \notin \mathcal{W}'$. Note $\mathcal{W}$ varies with the collection process going and the algorithm won't block those new-coming workers. It will decide these workers' eligibility after collecting enough number of entities for them. If worker w is not blocked, the algorithm then decides whether to give bonus (line 10), and assigns a corresponding task. After worker w answering the task, the algorithm checks the submitted entities (line 16). If the entities are of high quality, it will add them to multiset $\mathcal{R}$ (line 17) and also update the entity set $\mathcal{O}$ using entity resolution techniques.

Example 1: Suppose the workers in Table II answer our NBA player tasks. The first answer is "Curry" given by w_1 under a `NoBonus` task (with cost of \$1). We check the entity and add it in to $\mathcal{O}$. Similarly, w_2 gives an entity "Harden" and we add it into $\mathcal{O}$. Then we provide a `Bonus` task (with cost of \$1.5) for w_1, and w_1 answers a duplicated entity "Harden". After our reminder, she changes it to "James". Similarly, w_2 answers a duplicated entity "Curry" and she changes it to "James Jones" after the reminder. Therefore we spend \$5 to collect 4 different entities, which should take \$6 in the traditional way.

IV. Worker Elimination

This section presents worker elimination that eliminates the "unqualified" worker for entity collection. The challenge is to determine what constitutes the "unqualified" workers who make limited contributions to entity collection. We introduce the following two factors to address this challenge.

Worker	Answer set $\mathcal{R}$	v	E
w_1	{"Curry", "Harden", "James"}	3	0
w_2	{"Harden", "Curry", "Jones"}	1	0
w_3	{"Curry", "Durant", "Redick", "Young"}	6	0
w_4	{"Beckham", "Curry", "Charlie", "Lisa" }	8	3

TABLE II
An Example For Worker Elimination.

The first factor is *worker quality*. Workers with low quality who often provide incorrect entities should be naturally eliminated, so as to avoid wasteful incentive. For instance, consider our example of collecting active NBA players. Some low-quality workers may submit players in other colleagues or just irrelevant entities (e.g., nations, teams) for cheating the rewards. Some other workers may be so careless that they submit famous but retired NBA players. In either of the aforementioned cases, we would collect useless entities not belonging to our target domain Ω while still paying rewards.

The second factor is *entity distinctness*. We observe that, in some worker sets, entities from different workers have significant *overlaps* to each other. This is because the workers may have similar background about the entities. For example, workers familiar with the same NBA team are likely to provide overlapping players. For such worker sets, employing a few representative workers could be better than using them all. The other reason is that providing a distinct entity would become more and more difficult as the collection process goes. Especially at the late period of the collection, some workers will not accept the bonus mechanism (see Section V) and provide many entities duplicated with others. Thus, we naturally would like to eliminate such workers.

We formalize worker quality and entity distinctness and combine these two factors in an optimization problem, called *worker utility maximization*. We introduce the problem and prove its hardness in Section IV-A. Then, we provide a heuristic algorithm to solve the problem in Section IV-B, and present an *entity validation* technique that facilitates estimation of worker's quality in Section IV-C.

A. Worker Utility Maximization

Intuitively, worker utility is proposed to formalize the two factors, *worker quality* and *entity distinction*, which are introduced previously. Our basic idea is to derive these two factors from the existing entities which have already been submitted by the workers. To this end, we first introduce some notations. Let $\mathcal{R}_j$ denote the multiset of entities submitted by worker w_j. Note that a worker may provide duplicated entities, and so $\mathcal{R}_j$ may contain an instance in multiple times. Nevertheless, for ease of presentation, we first consider that $\mathcal{R}_j$ is a set without multiple instances of a same entity, which can be easily extended to the case that $\mathcal{R}_j$ is a multiset.

Worker quality. Based on the notations, we first define worker quality. A natural way is to examine the entities submitted by a worker and measure the number of the worker's *error entities*, denoted by E_j. The more error entities provided by the worker, the lower the worker's quality is. Formally,

consider a worker w_j and her entity multiset $\mathcal{R}_j$. According to the definition of entity collection, we consider an entity o_i as an error one *iif* it does not belong to our target domain Ω, i.e., $o_i \notin \Omega$. However, the non-trivial part is that we do not know whether a collected entity o_i belongs to Ω or not. We devise an *entity validation* technique to address this problem, and the output of entity validation is the probability that o_i is incorrect, denoted by $Pr(o_i \notin \Omega)$. We postpone the description of entity validation in Section IV-C. Given the probabilities returned by entity validation, we can compute E_j of worker w_j as $E_j = \sum_{o_i \in \mathcal{R}_j} Pr(o_i \notin \Omega)$.

Intuitively, considering a worker set $\mathcal{W}$, we would like to control the total number of error entities to make sure the number will not be too large. Formally, we introduce a *tolerance threshold* τ that represents the maximum ratio of error entities we can tolerate. Based on this, we introduce a worker quality constraint for worker set $\mathcal{W}$ as

$$\sum_{w_j \in \mathcal{W}} E_j \leqslant \tau \cdot |\mathcal{R}|, \tag{4}$$

where $|\mathcal{R}|$ is the total number of entities collected so far, and $\tau \cdot |\mathcal{R}|$ is the maximum number of error entities we can tolerate.

Entity distinctness. We prefer the workers who will provide more distinct entities in their following requests, as mentioned previously. To this end, we introduce the *entity distinctness*, denoted by $D_{\mathcal{W}}$, of a worker set $\mathcal{W}$, that represents the number of distinct entities in any time of the collection process. To compute it, we consider the following two aspects.

First, we examine all the entities submitted by the workers in $\mathcal{W}$, and estimate the *likelihood* of distinctness. Formally, let us consider entity set $\mathcal{R}_j$ of each worker w_j in $\mathcal{W}$. Then, $(|\cup \mathcal{R}_j|)/(\sum |\mathcal{R}_j|)$ is the proportion of distinct entities provided by the workers in $\mathcal{W}$, and we use this proportion to estimate the likelihood. However, solely considering the aforementioned likelihood may not be good. On the one hand, it would lead to a result consisting of only one worker (the likelihood for any individual worker is 1). On the other hand, it does not consider the "throughput" (number of entities per time) of the worker set. Obviously, a worker set with high throughput would benefit many other components of `CrowdEC`, such as entity validation, incentive pricing, etc. Thus, we introduce v_j to denote the throughput of worker w_j, i.e., the number of entities per time provided by w_j. This can be also estimated from worker's submission history in $\mathcal{R}_j$. Then, $\sum_{w_j \in \mathcal{W}} v_j$ can be used to represent the throughput of $\mathcal{W}$.

Based on the above-defined notations, we compute the entity distinctness $D_{\mathcal{W}}$ as follows.

$$D_{\mathcal{W}} = \frac{\left|\bigcup_{w_j \in \mathcal{W}} \mathcal{R}_j\right| \sum_{w_j \in \mathcal{W}} v_j}{\sum_{w_j \in \mathcal{W}} |\mathcal{R}_j|} \tag{5}$$

For instance, considering the example in table II, given $\mathcal{W} = \{w_1, w_2\}$, $\frac{|\mathcal{R}_1 \cup \mathcal{R}_2| \times (3+1)}{|\mathcal{R}_1| + |\mathcal{R}_2|} = \frac{16}{6}$.

Combining both entity distinctness and worker quality, we formalize worker elimination as an optimization problem that maximizes entity distinctness while bounding the errors.

Definition 2: (Error-Bounded Distinctness Maximization) Given a set $\mathcal{W}$ of workers, it selects a subset $\mathcal{W}^* \subseteq \mathcal{W}$ to maximize entity distinctness while bounding the errors, i.e., $\mathcal{W}^* = \arg_{\mathcal{W}' \subseteq \mathcal{W}} \max D_{W'}$ s.t. $\sum E_j < \tau \times |\mathcal{R}|, w_j \in \mathcal{W}^*$.

After determining $\mathcal{W}^*$, we eliminate workers in $\mathcal{W} \backslash \mathcal{W}^*$ from $\mathcal{W}$ and only use qualified worker in $\mathcal{W}^*$.

Example 2: For example, assume that we have 4 workers, table II shows their answer set, throughput and the number of errors respectively. We can see that w_1, w_2 and w_3 does not make mistake so their errors number is 0. However, since $E_4 = 3$ and $3 > 0.1 \times 14$, w_4 can not be involved in the optimal worker set. If we choose $\{w_1, w_2, w_3\}$, $D_{\{w_1,w_2,w_3\}} = \frac{7 \times (3+1+6)}{3+3+4} = 7$. If we choose $\{w_1, w_3\}$, $D_{\{w_1,w_3\}} = \frac{6 \times (3+6)}{3+4} = 7.7$ and $\mathcal{W}^* = \{w_1, w_3\}$.

Next, we show the hardness of error-bounded distinctness maximization problem, and we have the following theorem. We omit the proof due to the space constraint.

Theorem 1: The error-bounded distinctness maximization problem is NP-hard.

B. Heuristic Algorithm

This section presents a heuristic algorithm to address the error-bounded distinctness maximization problem. We find that workers with high value of $\frac{v_j}{|\mathcal{R}_j|}$ are likely to contribute much to maximize $D_{\mathcal{W}}$ according to Lemma 1. Therefore, given a worker set $\mathcal{W}$, the idea is to first sort the workers in the descending order of $\frac{v_j}{|\mathcal{R}_j|}$, i.e.,

$$\frac{v_1}{|\mathcal{R}_1|} \geqslant \frac{v_2}{|\mathcal{R}_2|} \geqslant \ldots \geqslant \frac{v_i}{|\mathcal{R}_i|} \geqslant \ldots \geqslant \frac{v_{|\mathcal{W}|}}{|\mathcal{R}_{|\mathcal{W}|}|}. \tag{6}$$

Then, we have the following interesting property. Consider any worker set $\mathcal{W}'$ and w_j has the largest $\frac{v_j}{|\mathcal{R}_j|}$ among workers in $\mathcal{W}'$. Then, we can always improve entity distinctness $D_{\mathcal{W}}$ if we insert the "prefix" of w_j in the above sequence (Equation 6) into $\mathcal{W}'$, as stated in the following lemma.

Lemma 1: Assume that w_j has the largest $\frac{v_j}{\mathcal{R}_j}$ among workers in $\mathcal{W}'$ and $\mathcal{W}_s$ is a set of workers who satisfy that $\frac{v_i}{\mathcal{R}_i} \geqslant \frac{|v_j|}{|\mathcal{R}_j|}, w_i \in \mathcal{W}_s$. Then we have $D_{\mathcal{W}_s \cup \mathcal{W}'} \geqslant D_{\mathcal{W}'}$.

Based on Lemma 1, we design a heuristic algorithm and the pseudo code is given in Algorithm 2. The algorithm first sorts

Algorithm 2: Heuristic Algorithm for Worker Selection

```
Input: W,R
Output: the selected workers set W*
Sort workers in W in descending order by v_j/|R_j|;
max = 0;
for i from 1 to |W| do
    W' ← {};
    for j from 1 to i do
        W'.add(w_j);
    while Σ_{w_k ∈ W'} E_k ⩽ τ · |R| do
        Find the worker w_{k'} largest E_{k'};
        W'.remove(w_{k'});
    if D_{W'} > max then
        max ← D_{W'};
        W* ← W';
return W*;
```

the workers in $\mathcal{W}$ in the descending order of $\frac{v_j}{|\mathcal{R}_j|}$ (line 1), and generates a worker sequence. Then, it examines every prefix of this sequence on distinctness and worker quality. If the workers in the prefix exceed the error bound $\tau \cdot |\mathcal{R}|$, the algorithm iteratively removes the workers with largest E_j until the error is bounded (line 8 and 9), and updates the value of $D_{\mathcal{W}}$. After considering all such prefixes, the algorithm chooses the one with the largest $D_{\mathcal{W}}$ (line 11 and 12).

Example 3: Given workers in Table II, we get $\frac{v_4}{|\mathcal{R}_4|} > \frac{v_3}{|\mathcal{R}_3|} > \frac{v_1}{|\mathcal{R}_1|} > \frac{v_2}{|\mathcal{R}_2|}$. We first consider w_4. Since w_4 provides so many wrong answers that $E_4 = 3$ is already larger that 1.4, she will be removed in any workers set. Therefore we consider worker set $\{w_3\}$, $\{w_3, w_1\}$ and $\{w_3, w_1, w_2\}$ in turn. Finally we return $\{w_3, w_1\}$ and block w_2 and w_4.

C. Entity Validation

Now, we discuss how to validate correctness of an entity o_t in $\mathcal{R}$, i.e., evaluating whether $o_t \in \Omega$. Obviously, traditional quality control techniques in crowdsourcing, such as majority voting, will not work for this open-end task. To address the challenge, we introduce an entity validation method, and the idea is two-fold. On the one hand, we solve this problem also using crowdsourcing, i.e., publishing entity validation tasks, such as "*Is Stephen Curry an active NBA player?*". However, to reduce crowdsourcing cost, we may not be able to publish too many such tasks. So on the other hand, if an entity is collected by multiple workers, we can identify that it would be more likely to be correct.

The non-trivial part here is to determine which entities should be crowdsourced for validation. Based on our observation, we find that few workers would give the same incorrect entity. Thus, we focus on crowdsourcing the distinct entities. However, especially in the beginning of collection, most of the collected entities are distinct, and we cannot afford to crowdsource them all. As a result, we need an effective method to distinguish which distinct entities are more likely to be error and thus should be crowdsourced.

Technically, we utilize a Bayesian-based method to address this problem. Formally, we slightly abuse the notation to also use o_t to denote the event that $o_t \in \Omega$, and $\overline{o_t}$ to denote $o_t \notin \Omega$. Moreover, we also introduce θ_t to denote the event that o_t is distinct and use $\overline{\theta_t}$ to represent negated event. Given a distinct entity, we compute the following probability

$$Pr(o_t|\theta_t) = \frac{Pr(\theta_t|o_t) \cdot Pr(o_t)}{Pr(\theta_t|o_t) \cdot Pr(o_t) + Pr(\theta_t|\overline{o_t}) \cdot Pr(\overline{o_t})}. \quad (7)$$

In Equation 7, $Pr(\theta_t|o_t)$ is the probability of distinctness for a correct entity, which can be easily estimated by the sample coverage SC (Section II-B), i.e., $Pr(\theta_t|o_t) = 1 - SC$. Moreover, consider probability $Pr(\theta_t|\overline{o_t})$ that means the probability of distinctness for an incorrect entity. As incorrect entity would have large likelihood to be distinct, we estimate $Pr(\theta_t|\overline{o_t})$ as 1. For the priors $Pr(o_t)$ and $Pr(\overline{o_t})$, we estimate them by previously estimates of error rate of the worker providing the entity, i.e., $E_j/|\mathcal{R}_j|$ (see Section IV-A for E_j).

After estimating $Pr(o_t|\theta_t)$ for each distinct entity in $\mathcal{R}$, we use a threshold β, crowdsource all the entities whose $Pr(o_t|\theta_t) \geqslant \beta$, and collect validation results from workers.

Remark. We can avoid asking the worker to validate the entity provided by herself via recording worker ID. To prevent workers from maliciously forming a group and submitting identical but wrong entities, we can use techniques in [28] to identify sybil workers who deliberately give the same entities and block them. Besides, utilizing external sources, such as Wikipedia, could be beneficial for the entity validation of some *coarse-grained* entity collection tasks, e.g., all countries, cities, etc. Nevertheless, it is challenging to employ Wikipedia for many collection tasks which require to find *fine-grained* entities satisfying some conditions, such as *active* NBA players.

V. Incentive Pricing

This section presents our *incentive pricing* technique that encourages each qualified worker to provide distinct entities rather than duplicates. We first introduce our pricing schemes, i.e., `NoBonus` and `Bonus` in Section V-A, and then formalize a decision making problem that aims to minimize the overall collection cost by judiciously assigning workers with appropriate pricing schemes in Section V-B. Finally, we present a lightweight algorithm that solves the decision-making problem with superior approximation guarantee in Section V-C.

A. Pricing Schemes

We introduce a *bonus*-based incentive mechanism, as most of the crowdsourcing platforms do not support dynamic pricing. The idea is to promise a worker an extra bonus if the worker wants to try multiple times and eventually submits a distinct entity. For example, suppose that we allow a worker to try at most 3 times. Then, we will not offer the bonus until a distinct entity, which is not duplicated with the others in $\mathcal{R}$, is submitted within the 3 times of attempts. Formally, we use $c_{\mathtt{r}}$ and $c_{\mathtt{b}}$ to respectively represent the base reward and the bonus of a collection task t, and $c(t)$ to denote the total reward paid for t. Let h denote an attempt constraint, i.e., the maximum number of entities we allow a worker to try. Moreover, recall that we have defined θ_o ($\overline{\theta_o}$) to represent that entity o is distinct (duplicated) as discussed in Section IV-C. Based on the notations, we define the bonus scheme as follows.

Definition 3 (`Bonus` *Scheme):* Given a constraint h, it uses task t to collect a set of entities, denoted by $\mathcal{R}_t = \{o_1, \ldots, o_{|\mathcal{R}_t|}\}$ s.t. $|\mathcal{R}_t| \leqslant h$. It decides the reward $c(t)$ by

$$c(t) = \begin{cases} c_{\mathtt{r}} + c_{\mathtt{b}} & \exists o \in \mathcal{R}_t, I[\theta_o] = 1 \\ c_{\mathtt{r}} & otherwise, \end{cases} \quad (8)$$

where $I[\cdot]$ is an indicator function that return either 1 or 0, and $I[\theta_o] = 1$ means that entity o is distinct.

On the other hand, we also provide the normal interface without bonus that allows worker to submit only one entity and always provides base reward, i.e.,

Definition 4 (`NoBonus` *Scheme):* It uses a task t to collect only one entity, i.e., $|\mathcal{R}_t| = 1$, and offers reward $c(t) = c_{\mathtt{r}}$.

Example 4: Suppose that a worker is assigned with the `Bonus` scheme and provides three entities $\{o_1, o_2, o_3\}$ with

distinctness status $\{\overline{\theta_{o_1}}, \overline{\theta_{o_1}}, \theta_{o_1}\}$. Moreover, consider base reward $c_{\mathtt{r}} = 1.0$ and bonus $c_{\mathtt{b}} = 0.5$. We need to pay 1.5 to the worker given the bonus scheme. On contrary, we have to pay 3.0 if a `NoBonus` scheme is assigned.

B. Optimal Incentive Pricing Problem

This section presents the optimal incentive pricing problem. Intuitively, for each request from a qualified worker, we need to make a decision that which pricing scheme (`Bonus` or `NoBonus`) should be assigned to the worker, so as to reduce the overall cost. Interestingly, we find there is no clear winner in these schemes. For example, in the beginning of the collection, `NoBonus` may be better because most of collected entities would be distinct. On the contrary, as more and more entities are collected, the difficulty of providing a distinct entity should be higher, and thus `Bonus` may be superior. Moreover, the decision may also depend on whether the requesting worker wants to try multiple attempts for submitting a distinct entity. If not, `Bonus` may not be better than `NoBonus` as bonus is useless to incent the worker.

To formalize the aforementioned intuitions, we introduce a cost-based optimization method. Formally, consider a request from worker w_j at any time of the collection process, and suppose that the number of entities collected in current $\mathcal{R}$ is N. For ease of presentation, we define $\langle w_j, N\rangle$ as a *state* of the collection process, the intuition of which is that we have collected N entities and encountered a request from worker w_j. Then, we introduce $C(w_j, N)$ to represent the total cost we paid from the state $\langle w_j, N\rangle$ to the end of collection. Intuitively, $C(w_j, N)$ depends on the following factors.

1) *Our pricing decisions*: for every requesting worker from w_j to the end of collection, we need to decide a bonus scheme. More formally, we introduce binary variable X where $X = 1$ if providing `Bonus` scheme, and $X = 0$ otherwise.

2) *Requesting workers*: we actually do not know which workers will request for our tasks in the future. To address this, we consider the expectation. We introduce p_j as the probability that w_j will request for our task, and the summation $\sum_{w_j \in \mathcal{W}} p_j = 1$ for all the qualified workers.

3) *Worker's behavior*: when assigned with a `Bonus` scheme, even being notified with duplications, a worker may not want to continuously submit more workers. This may be mainly because the bonus is not enough to incent the worker. To formulate this behavior, we introduce a probability q_j that models the likelihood the bonus can incent worker w_j to provide a new entity. Then, we introduce a random variable N_{w_j} that captures the number of entities provided by w_j if assigned with `Bonus` scheme. Given an attempt constraint h, the possible values of N_{w_j} are $\{1, 2, \ldots, h\}$. We introduce notation $Pr(N^k_{w_j})$ to indicate the probability that $N_{w_j} = k$ where $1 \leqslant k \leqslant h$. We postpone the estimation of $Pr(N^k_{w_j})$, and first assume that it is already known.

4) *Worker's reward*: according to Equation 8, the reward paid to worker w_j depends on whether w_j provides a distinct entity, which is a probabilistic event. For ease of presentation, we use $\mathbb{E}[c_{w_j}]$ to represent the expected cost. The estimation of $\mathbb{E}[c_{w_j}]$ will be described later.

Cost optimization. We first discuss the problem of optimizing cost $C(w_j, N)$. We define $C^*(w_j, N)$ as the minimum expectation of cost among all the possible values of $C(w_j, N)$, and we can see that $C^*(w_j, N)$ can be computed by its optimal subproblems. Let us introduce C_0 and C_1 to respectively denote the cost of assigning `NoBonus` (i.e., $X = 0$) and `Bonus` ($X = 1$). Then, we have

$$C_0 = c_{\mathtt{r}} + \sum_{w_i} p_i \cdot C^*(w_i, N+1), \tag{9}$$

which means paying $c_{\mathtt{r}}$ to w_j and $\sum_{w_i} p_i \cdot C^*(w_i, N+1)$ is the expectation of optimal subproblems.

Similarly, we have C_1 for assigning `Bonus`. The idea is to consider every possible workers in the following requests and every possible number of entities submitted by w_j submitted in this request, i.e.,

$$C_1 = \mathbb{E}[c_{w_j}] + \sum_{w_i} p_i \sum_k Pr(N^k_{w_j}) \cdot C^*(w_i, N+k) \tag{10}$$

Based on these equations, we can compute $C^*(w_j, N)$ as

$$C^*(w_j, N) = \min\{C_0, C_1\}. \tag{11}$$

Now we define the optimal incentive pricing problem.

Definition 5: (Optimal Incentive Pricing Problem) Given current state $\langle w_j, N\rangle$, it decides the best pricing scheme for w_j that leads to the minimum total cost $C^*(w_j, N)$ from the state to the end of collection, i.e., $X = \arg_i \min_{i \in \{0,1\}} C_i$.

Probability estimation. We next present how to estimate $Pr(N^k_{w_j})$ and $\mathbb{E}[c_{w_j}]$ in the above equations.

1) *Estimation of* $Pr(N^k_{w_j})$: recall that this is the probability of the event that the number of entities submitted by w_j equals k. Based on Definition 3, this is equivalent to an event that w_j submits at least one distinct entity or w_j stops after submitting k duplicated entities. Recall that we use θ ($\overline{\theta}$) to represent the event that an entity submitted by w_j is distinct (duplicated). Thus, $Pr(N^k_{w_j})$ can be computed as

$$Pr(N^k_{w_j}) = \big(q_j \cdot Pr(\overline{\theta})\big)^{k-1} \cdot \big(1 - q_j \cdot Pr(\overline{\theta})\big), \tag{12}$$

where $\big(q_j \cdot Pr(\overline{\theta})\big)^{k-1}$ captures the probability that w_j provides $k-1$ duplicated entities, and $\big(1 - q_j \cdot Pr(\overline{\theta})\big)$ is the probability that w_j either provides a distinct entities or stops after submitting a duplicated one.

Note that p_j can be computed by w_j's historical behaviors and $Pr(\theta)$ ($Pr(\overline{\theta})$) can be obtained from our completion estimator, which will be updated as $\mathcal{R}$ changes.

2) *Estimation of* $\mathbb{E}[c_{w_j}]$: according to Equation 8, the key to estimate $\mathbb{E}[c_{w_j}]$ is to determine when the event that w_j provides a distinct entity, denoted by $\theta_{\mathcal{R}_t}$, occurs. Similar to previous estimation, this probability can be estimated by

$$Pr(\theta_{\mathcal{R}_t}) = \sum_{k=1}^{h} Pr(\theta) \cdot (q_j \cdot Pr(\overline{\theta}))^{k-1}, \tag{13}$$

where the intuition of the equation is that w_j first provides $k-1$ duplicated and then gives a distinct one. Based on this, $\mathbb{E}[c_{w_j}]$ is estimated as

$$\mathbb{E}[c_{w_j}] = c_{\mathtt{r}} + c_{\mathtt{b}} \cdot Pr(\theta_{\mathcal{R}_t}). \tag{14}$$

For example, suppose $h = 2$, $Pr(\theta) = 0.9$, $q_j = 0.8$ and w_j requests task t. Then $Pr(\theta_{\mathcal{R}_t}) = 0.9 + 0.9 \times 0.8 \times 0.1 = 0.97$ and $Pr(N^2_{w_j}) = 0.8 \times (1-0.9) \times (1-0.8 \times (1-0.9)) = 0.074$.

C. Online Algorithm with Theoretical Guarantee

Since Equation 11 is a recursive function, we solve the optimal incentive pricing problem using a dynamic programming (DP) algorithm. However, there are two obstacles which prevent us from doing like that. Firstly, we require to know $Pr(\theta)$ in the future if we want to run the DP algorithm. But it is too hard to predict it because our problem is an online problem. Secondly, we cannot decide the boundary condition of the DP algorithm because we don't know how many entities needed to result in the completion of the collection. In order to address these problems, we propose a greedy online algorithm to solve this problem, which has an approximate ratio close to 1. We first illustrate the algorithm as follows. Each time when a worker w_j requests the task t, if w_j satisfies Equation 15,

$$c_{\mathtt{b}} \cdot Pr(\theta_{\mathcal{R}_t}) \geqslant (c_{\mathtt{r}} + c_{\mathtt{b}}) \times \sum_{k=2}^{h} Pr(N^k_{w_j})(k-1) \qquad (15)$$

we assign her `NoBonus` schema. Otherwise we assign her `Bonus` schema. For example, suppose $h = 2$, $Pr(\theta) = 0.9$, and w_j requests the task t with $q_j = 0.8$. Besides, $c_{\mathtt{r}} = 1, c_{\mathtt{b}} = 0.5$, then $c_{\mathtt{b}} \cdot Pr(\theta_{\mathcal{R}_t}) = 0.5 \times 0.97 = 0.49$, $(c_{\mathtt{r}} + c_{\mathtt{b}}) \times \sum_{k=2}^{h} Pr(N^k_{w_j})(k-1) = 1.5 \times 0.074 = 0.11$, so we choose the `NoBonus` schema.

Theorem 2 shows that even though we do not know these probabilities in advance, our greedy algorithm has an approximate ratio of $(1 + \frac{1}{\mathtt{OPT}})$, where `OPT` is the minimum cost, so it achieves a solution that is extremely close to the optimal one. We omit the proof due to the space constraint.

Theorem 2: The online algorithm has an approximate ratio of $(1 + \frac{1}{\mathtt{OPT}})$.

VI. Experiment Evaluation

This section evaluated our incentive-based framework. We have implemented our framework in Python on a Ubuntu server with Intel 2.4GHz Processor and 32GB memory, and interacted with AMT using their API.

A. Experimental Settings

Datasets. We evaluated `CrowdEC` using three real datasets. The statistics and default values of the parameters were shown in Table III. (1) `ActiveNBA` is for collecting active NBA players. A typical task for `ActiveNBA` is to ask the question, "Could you please give me a name of an active NBA player?". The ground-truth Ω of `ActiveNBA` contained 450 active players in NBA teams. Using crowdsourcing, we have collected 1059 entities in total from 27 workers. (2) `TopUniv` is for collecting top-100 universities in the world. Similarly, we ask the question, "Could you give me a name of one of top-100 universities?". The ground-truth contains 100 universities and we have collected 248 entities from 15 workers. The ground-truth is generated by merging two well-accepted university rankings, the `2017 U.S. News ranking`[3] and the `2017 Academic Ranking of World Universities`[4]. We first compute the intersection of these to rankings and obtain a new set U. Then, we determine the ranking order of the universities in U. There are multiple strategies to get top-100 from U [5]. We use a simple strategy that sorts the universities in U by the sum of rankings in the two lists. We can also use more sophisticated strategies [5]. (3) `AllNBA` is for collecting all (4260) NBA players. We get similar results and observations to the `ActiveNBA` dataset and we omit the results due to space constraints.

Name	$\lvert\Omega\rvert$	#workers	# entities	$c_{\mathtt{r}}$	$c_{\mathtt{b}}$
`ActiveNBA`	450	27	1059	\$0.1	\$0.15
`TopUniv`	100	15	248	\$0.1	\$0.15
`AllNBA`	4260	53	11500	\$0.1	\$0.15

TABLE III
Dataset

Implementation. We have deployed `CrowdEC` on AMT. There were two issues about the implementation. First, we needed to assign different pricing schemes as the collection process goes and eliminated unqualified workers, both of which were not natively supported by AMT. Thus, we used the "External Question" mechanism on AMT. We built a web server and interacted with AMT using the APIs for assigning optimal pricing scheme and eliminating workers. Second, as AMT did not support dynamic pricing, we utilized its bonus mechanism. We set the price of each task as the base reward, and utilized AMT's bonus API for giving the extra bonus.

Next, we presented parameter settings in our experiments. We set the error bound τ as 0.1. We set $\beta = 0.6$ and $\epsilon = 0.1$. For incentive pricing, we set the base reward $c_{\mathtt{r}} = \$0.1$ and bonus $c_{\mathtt{b}} = \$0.5$. For entity validation, we paid \$0.05 for task, and applied majority voting from 3 workers. Moreover, we set the attempt constraint in `Bonus`, i.e., the maximum number of answers a worker can provide as $h = 4$. We set $q_j = 1$ initially, and continuously updated it as we collected the entities.

Comparisons with state-of-the-art techniques. We compared `CrowdEC` with the existing approaches `Enumeration` [25] and `CrowdFill` [21]. For fair comparison, we made sure that different approaches were evaluated on the same set of workers. To this end, we collected as many entities as possible from the workers. For example, for the eliminated workers in `CrowdEC`, we will not block them and still assign `NoBonus` tasks to them for collecting entities. Based on the collected entities, we generated a sequence of worker requests, where each request in the sequence consisted of a worker and an entity from the worker. Then, based on the *same* sequence of worker requests, we respectively ran `Enumeration`, `CrowdFill` and `CrowdEC`, and evaluated their performance on crowdsourcing cost and entity quality.

Evaluation on alternative strategies. Based on the sequence of worker requests mentioned above, we also evaluated alternative strategies that could be used in `CrowdEC`.

For evaluating worker elimination, we compared our proposed method with three baselines `D_only`, `Q_only` and `None`. 1) `None` did not performs worker elimination. 2) `D_only` only considered entity distinctness while ignoring the error bound

[3]https://www.usnews.com/education/best-global-universities/rankings

[4]http://www.shanghairanking.com/ARWU2017.html

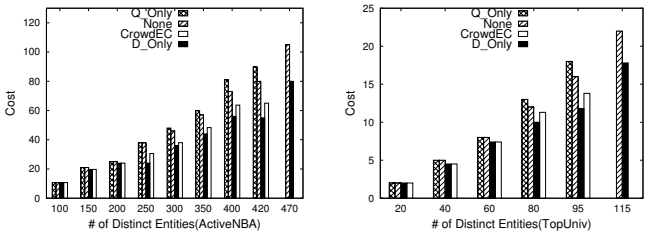

Fig. 3. Evaluate Worker Elimination: Cost

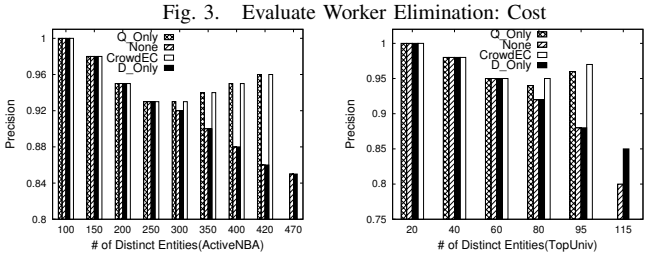

Fig. 4. Evaluate Worker Elimination: Precision

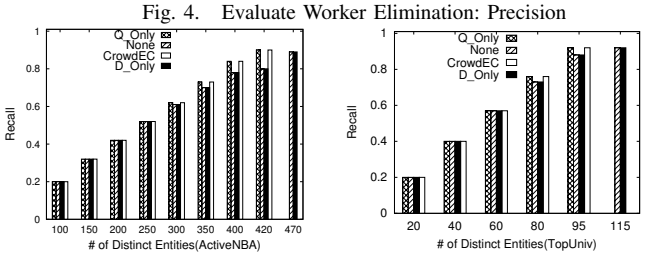

Fig. 5. Evaluate Worker Elimination: Recall

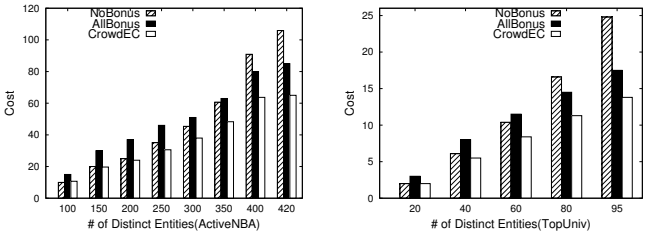

Fig. 6. Evaluate Incentive Pricing: Cost

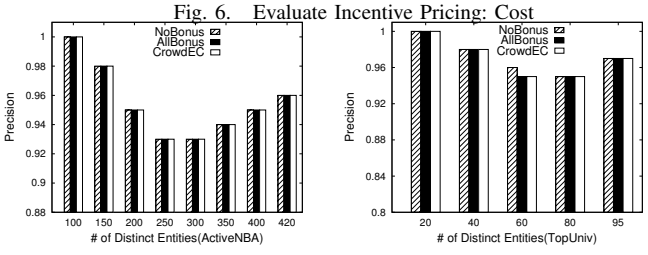

Fig. 7. Evaluate Incentive Pricing: Precision

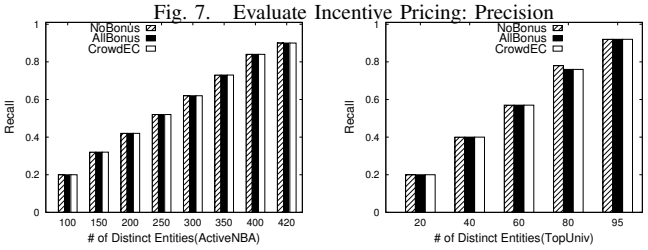

Fig. 8. Evaluate Incentive Pricing: Recall

for worker quality. 2) `Q_only` only selected workers such that their errors were bounded while ignoring entity distinctness.

For evaluating incentive pricing, we also considered the following two baselines. 1) `NoBonus` assigns `NoBonus` pricing scheme to all tasks. 2) `AllBonus` assigns `Bonus` pricing scheme to all tasks. Note that, when evaluating incentive pricing, the worker elimination was considered by default.

Evaluation metrics. We evaluated the performance of approaches on cost, precision and recall. Cost was the total cost an approach takes to complete the collection task. Precision and recall were used to measure the quality of the collected entities. Suppose the subset of correct entities in $\mathcal{O}$ was $\mathcal{O}_T$. Then the precision $p = \frac{|\mathcal{O}_T|}{|\mathcal{O}|}$ and the recall $r = \frac{|\mathcal{O}_T|}{|\Omega|}$.

B. Evaluation on Worker Elimination

We compared the cost, precision and recall among algorithms `None`, `D_only`, `Q_only` and `CrowdEC`.

For cost, as Figure 3 shows, the x-coordinate and the y-coordinate represented the number of distinct entities we have collected and the cost respectively. We can see that `Q_only` and `None` were merely different. This is because the factor worker quality has limited affect on the cost. On the other hand, `D_only` took much less than `Q_only` and `None`. For example, on the `ActiveNBA` dataset, `D_only` took \$80, which was 20% less than `None` when the distinct number was 470. The reason is that `D_only` blocked those workers who provided duplicated entities or were unwilling to submit a new distinct one. Our framework, `CrowdEC` outperformed `D_only` on both datasets. We can see that on `ActiveNBA` dataset, `CrowdEC` took \$60 at last but `D_only` took \$80 because `CrowdEC` considered both the utility and quality. Another observation was that `D_only` took less than `CrowdEC` when the distinct number was less than 420 on data `ActiveNBA`. This is because `D_only` could lead to wrong answers since it did not have entity validation. And wrong answers were always distinct ones because those low quality workers provided them to cheat money by giving some random strings, e.g., adding or removing some characters based on the true answers, which were not likely to be duplicated. Therefore, given the same number of distinct entities, `D_only` took less. However, since low quality workers gave more distinct answers, the completion estimator overestimated the size of Ω. Therefore, we can see that `CrowdEC` stops when it collected 420 distinct answers while `D_only` collected 470 in total on `ActiveNBA`. Moreover, on datasets `ActiveNBA` and `TopUniv`, we can see that when the number of distinct entities was smaller than 200 and 60 respectively, the difference of these four approaches was not distinct. This is because most workers provided distinct entities at the beginning and the entity validation threshold has not been below the parameter β.

For precision, we can see from Figure 4 that when the collection task completed, the precisions of `D_only` and `None` were much lower than `Q_only` and `CrowdEC`. This is because `Q_only` and `CrowdEC` not only checked some wrong entities and removed them but also eliminated those low quality workers to control quality. For example, on `ActiveNBA` dataset, `Q_only` and `CrowdEC` achieved a quality of 96% while the other two approaches were only 85% when the process stopped. Another observation is that the precision of all approaches went down during the front collection process because the entity validation had not started to work then. And during that time, the answers' quality was relative high because workers were patient and careful at the beginning. Since `CrowdEC` and `Q_only` had entity validation, the precision of them went up, and finally achieved 96% on both datasets, which was much higher than the other two approaches.

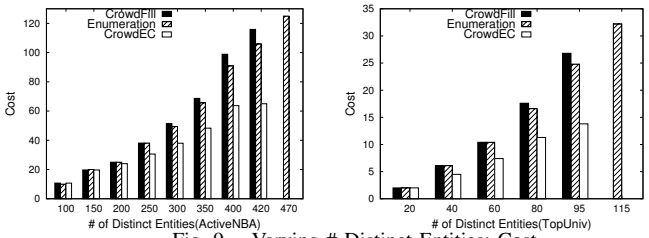

Fig. 9. Varying # Distinct Entities: Cost

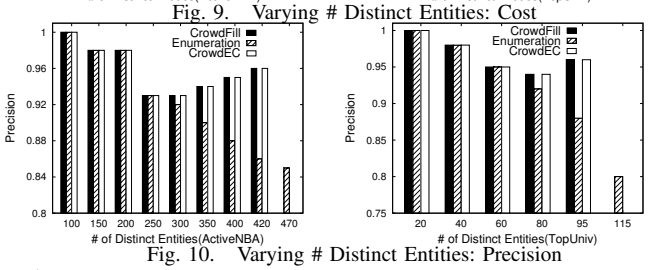

Fig. 10. Varying # Distinct Entities: Precision

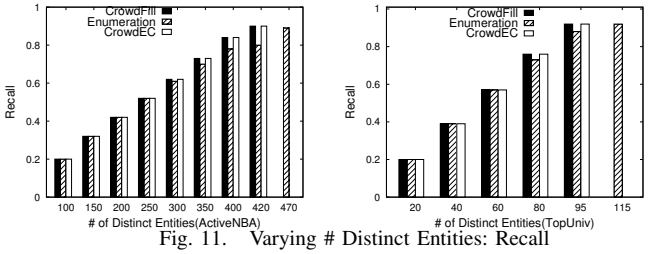

Fig. 11. Varying # Distinct Entities: Recall

For recall, we can see from Figure 5 that with the number of distinct answers increasing, the recall kept increasing on both datasets. During the front collection process, since workers had high quality, there was almost no difference among these four approaches. When the distinct number began to increase, `CrowdEC` and `Q_only` achieved a higher recall than the other two methods. Finally, since `D_only` and `None` collected more entities and stopped later, they achieved an almost equal high recall. They had a recall of 92% on both datasets.

C. Evaluation on Incentive Pricing

We evaluated the incentive pricing module. We can see from Figure 6 that at the beginning of the collection process, `NoBonus` took less than `AllBonus` because few workers provided duplicated answers at the beginning so that paying bonus was wasteful. For example, when the distinct number was 150 on dataset `ActiveNBA`, `AllBonus` costs \$30, more than `NoBonus` (\$20). As the collection process went, it was necessary to provide `Bonus` schema to incent workers to provide more answers because many duplicated answers appear. Therefore, `NoBonus` took more than `AllBonus` afterwards. For example, `NoBonus` took \$110 and \$25 while `AllBonus` took \$80 and \$17 on `ActiveNBA` and `TopUniv` respectively when the collection completed. Our framework `CrowdEC` outperformed both methods because we considered the sample coverage and the workers' individuality like q_j sufficiently. When a worker w_j had a low q_j, which indicated that she was not willing to change new answers, assigning `NoBonus` schema would be a better choice. For example, on `ActiveNBA`, when the distinct number was 420, `CrowdEC` cost \$60, which saved nearly 200% and 50% than `NoBonus` and `AllBonus` respectively. On dataset `TopUniv`, when the distinct number was 95, `CrowdEC` also took much less than `NoBonus` and `AllBonus`. Moreover, since there was worker elimination module in all three approaches, they all stopped when about 420 and 95 distinct entities were collected on `ActiveNBA` and `TopUniv` respectively.

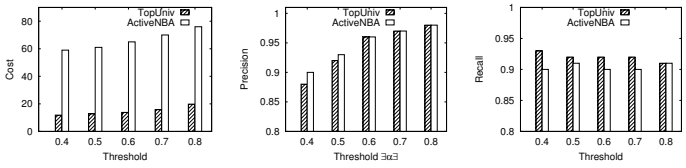

Fig. 12. Varying Threshold β

For precision, since we eliminated low quality workers during the collection time, our method achieved high precision on both `ActiveNBA` and `TopUniv` datasets (96% and 97% respectively), as shown in Figure 7. Besides, since the worker elimination algorithms of these 3 approaches were the same, the precision of them was almost the same. Furthermore, we can see from Figure 7 that the precision went down first and went up afterwards because the entity validation had started to work after some entities have been collected.

For recall, we can see from figure that the recall increased with the number of distinct entities increasing. Finally, we achieved recall of 92% on both datasets. Similar to the case of precision, there was little difference about the recall among the three methods as they used the same quality control method.

D. Comparation with Existing Work

In this section, we compared our framework `CrowdEC` with `Enumeration` [25] and `CrowdFill` [21] . `Enumeration` mainly focused on using collecting to estimate the cardinality of the Ω without considering the cost and quality. `CrowdFill` focused on the fill operation, but we can think about that filling a new record in a database was the same as collecting a new item in our task. `CrowdFill` used simple majority voting entity validation approach like us to control quality, but it did not consider duplicated answers.

For cost, we can see from Figure 9 that `CrowdEC`, `Enumeration` and `CrowdFill` took almost the same at the beginning because there were few duplicated answers and worker elimination module had not started at that time. Therefore, the advantage of `CrowdEC` has not been revealed yet. Then with the number of distinct entities increasing, considering many important factors like workers changing probability q_j, the sample coverage, `CrowdEC` can incent workers to provide more distinct answers to reduce the global cost. Also, `CrowdEC` can eliminate those workers who always submit duplicated answers. Therefore, it achieved much less cost than `Enumeration`. For example, on `ActiveNBA` dataset, `CrowdEC` took \$60, which saved 2 times more than `Enumeration` and `CrowdFill`. And on `TopUniv`, `Enumeration` spent \$34, which also took 2 times more than `CrowdEC` (\$14).

For precision, we can see from Figure 10 that the precision of `Enumeration` went down all the time because there was not any quality control approach method applied in it. But `CrowdEC` and `CrowdFill` can achieve a better precision because it can find out some wrong answers and remove them. Also, `CrowdEC` can eliminate those low quality workers through entity validation to prevent them from providing wrong answers continuously. For example, `CrowdEC` and `CrowdFill` achieved precision of about 95% on both datasets, which was much higher than that of `Enumeration` (85% and 80% on `ActiveNBA` and `TopUniv` respectively).

For recall, we can see from Figure 11 that the recall kept increasing on both datasets with the distinct number increasing.

At the beginning, all approaches had the nearly same recall. Afterwards, since CrowdEC and CrowdFill applied quality control method, it achieved higher recall than Enumeration. For example, on dataset ActiveNBA, when the distinct number was 420, CrowdEC had a recall of 85%, which was more than that of Enumeration (78%). Since Enumeration involved many wrong distinct answers, it will collect more answers than CrowdEC to arrive at the stop condition. Therefore, they achieved almost the same recall at the end(90% and 92% ActiveNBA and TopUniv respectively).

Evaluate β. In this paragraph, we evaluated an important parameter β, which helped to decide whether to check an entity. Intuitively, if we set β a high value, we can get a relative higher quality but cost more. If we set β a low value, we can save some money with sacrificing the quality. So we varied the value of β and evaluate the cost, precision and recall. For cost, we can see from the first picture in Figure 12 that when $\beta = 0.4$, the costs on two datasets were about \$58(ActiveNBA) and \$18(TopUniv). And with β increasing, the cost increased because we must check more in those cases. For example, when $\beta = 0.8$, the costs on two datasets were about \$78(ActiveNBA) and \$20(TopUniv). For precision, the second picture of in Figure 12 showed that the precision went up with β increases. For example, the precisions for ActiveNBA and TopUniv were 88% and 90% respectively when $\beta = 0.4$. When $\beta = 0.8$, the precisions increased to 98% for both datasets. It was noteworthy that when $\beta > 0.5$, the precision can be up to a high level(around or above 95%). This is not only because the entity validation is an effective quality control method, but also workers can check each other's answers during their collection process. So it is not necessary to set a so high value, which is kind of wasteful. For recall, in the third picture, we can see that the recalls for different β fluctuated slightly around 90% for both datasets, which indicated that β had little impact on recall and CrowdEC obtained most of entities in Ω robustly.

VII. Conclusion

We studied the crowdsourced entity collection problem. We proposed an incentive-based framework that collects entities with low cost and high quality. We devised a worker elimination approach to block those workers who always contribute low quality or duplicated answers. We designed two pricing schemes and formalized the optimal incentive pricing problem that assigns the best pricing schemes to different workers. Experimental results on real datasets showed that CrowdEC outperformed state-of-the-art methods on both cost and quality.

Acknowledgement. This work was supported by the 973 Program of China (2015CB358700), NSF of China (61632016,61472198,61521002,61661166012), and TAL education. Ju Fan was supported by the NSF of China (61602488, 61632016) and Tencent Social Ads Rhino-Bird Focused Research Grant.

References

[1] C. C. Cao, J. She, Y. Tong, and L. Chen. Whom to ask? jury selection for decision making tasks on micro-blog services. *PVLDB*, 5(11):1495–1506, 2012.

[2] C. Chai, G. Li, J. Li, D. Deng, and J. Feng. Cost-effective crowdsourced entity resolution: A partial-order approach. In *SIGMOD*, pages 969–984, 2016.

[3] A. Chao and S.-M. Lee. Estimating the number of classes via sample coverage. *Journal of the American statistical Association*, 87(417):210–217, 1992.

[4] Y. Chung, M. L. Mortensen, C. Binnig, and T. Kraska. Estimating the impact of unknown unknowns on aggregate query results. In *SIGMOD*, pages 861–876, 2016.

[5] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *WWW*, pages 613–622, 2001.

[6] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.

[7] J. Fan, G. Li, B. C. Ooi, K. Tan, and J. Feng. icrowd: An adaptive crowdsourcing framework. In *SIGMOD*, pages 1015–1030, 2015.

[8] J. Fan, M. Lu, B. C. Ooi, W. Tan, and M. Zhang. A hybrid machine-crowdsourcing system for matching web tables. In *ICDE 2014*, pages 976–987, 2014.

[9] J. Fan, Z. Wei, D. Zhang, J. Yang, and X. Du. Distribution-aware crowdsourced entity collection. *TKDE*, PP(99):1–1, 2017.

[10] J. Fan, M. Zhang, S. Kok, M. Lu, and B. C. Ooi. Crowdop: Query optimization for declarative crowdsourcing systems. *IEEE Trans. Knowl. Data Eng.*, 27(8):2078–2092, 2015.

[11] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. SIGMOD. pages 61–72, 2011.

[12] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. W. Shavlik, and X. Zhu. Corleone: hands-off crowdsourcing for entity matching. In *SIGMOD*, pages 601–612, 2014.

[13] I. J. Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3-4):237–264, 1953.

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

[15] P. G. Ipeirotis, F. Provost, and J. Wang. Quality management on amazon mechanical turk. In *SIGKDD Workshop*, HCOMP '10, pages 64–67, New York, NY, USA, 2010. ACM.

[16] G. Li, C. Chai, J. Fan, X. Weng, J. Li, Y. Zheng, Y. Li, X. Yu, X. Zhang, and H. Yuan. CDB: optimizing queries with crowd-based selections and joins. In *SIGMOD*, pages 1463–1478, 2017.

[17] G. Li, J. Wang, Y. Zheng, and M. J. Franklin. Crowdsourced data management: A survey. *IEEE Trans. Knowl. Data Eng.*, 28(9):2296–2319, 2016.

[18] 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.

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

[20] H. Park and J. Widom. Query optimization over crowdsourced data. *PVLDB*, 6(10):781–792, 2013.

[21] H. Park and J. Widom. Crowdfill: collecting structured data from the crowd. In *SIGMOD*, pages 577–588, 2014.

[22] C. Qiu, A. C. Squicciarini, B. Carminati, J. Caverlee, and D. R. Khare. Crowdselect: Increasing accuracy of crowdsourcing tasks through behavior prediction and user selection. In *CIKM 2016*, pages 539–548, 2016.

[23] C. Qiu, A. C. Squicciarini, S. M. Rajtmajer, and J. Caverlee. Dynamic contract design for heterogenous workers in crowdsourcing for quality control. In *ICDCS 2017*, pages 1168–1177, 2017.

[24] T. Rekatsinas, A. Deshpande, and A. G. Parameswaran. Crowdgather: Entity extraction over structured domains. *CoRR*, abs/1502.06823, 2015.

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

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

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

[28] D. Yuan, G. Li, Q. Li, and Y. Zheng. Sybil defense in crowdsourcing platforms. In *CIKM*, pages 1529–1538, 2017.

[29] X. Zhang, G. Li, and J. Feng. Crowdsourced top-k algorithms: An experimental evaluation. *VLDB*, 9(8):612–623, Apr. 2016.

[30] Y. Zheng, J. Wang, G. Li, R. Cheng, and J. Feng. QASCA: A quality-aware task assignment system for crowdsourcing applications. In *SIGMOD*, pages 1031–1046, 2015.