Distribution-Aware Crowdsourced Entity Collection

Ju Fan, Zhewei Wei, Dongxiang Zhang, Jingru Yang, Xiaoyong Du

Abstract—The problem of crowdsourced entity collection solicits people (a.k.a. workers) to complete missing data in a database and has witnessed many applications in knowledge base completion and enterprise data collection. Although previous studies have attempted to address the "open world" challenge of crowdsourced entity collection, they do not pay much attention to the "distribution" of the collected entities. Evidently, in many real applications, users may have *distribution requirements* on the collected entities, e.g., even spatial distribution when collecting points-of-interest. In this paper, we study a new research problem, *distribution-aware crowdsourced entity collection* (CROWDDEC): Given an expected distribution w.r.t. an attribute (e.g., region or year), it aims to collect a set of entities via crowdsourcing and minimize the difference of the entity distribution from the expected distribution. Due to the openness of crowdsourcing, the CROWDDEC problem calls for effective crowdsourcing quality control. We propose an adaptive worker selection approach to address this problem. The approach estimates underlying entity distribution of workers on-the-fly based on the collected entities. Then, it adaptively selects the best set of workers that minimizes the difference from the expected distribution. Once workers submit their answers, it adjusts the estimation of workers' underlying distributions for subsequent adaptive worker selections. We prove the hardness of the problem, and develop effective estimation techniques as well as efficient worker selection algorithms to support this approach. We deployed the proposed approach on Amazon Mechanical Turk and the experimental results on two real datasets show that the approach achieves superiority on both effectiveness and efficiency.

Index Terms—Crowdsourcing, Entity Collection, Sampling, Distribution-Aware

1 INTRODUCTION

Crowdsourcing exploits human intelligence to solve problems that are inherently difficult to machines, and has attracted growing interest recently. Many solutions have been proposed to support various operations on crowdsourced data, which can be broadly classified into two categories. The first one is crowdsourced data evaluation, which asks the crowd to evaluate data according to some criteria, including filter [32], [22], join [26], [39], sort / top-k [14], [7], etc. The second category is crowdsourced data collection [38], [6], [36], [35], which solicits the crowd to complete missing data in a database, where the missing data can be either specific attributes (e.g., homepage of a researcher) or entire entities (e.g., new researchers).

This paper focuses on the aforementioned second category of crowdsourced operations. In particular, we study the problem of *crowdsourced entity collection* that seeks to acquire missing entities (i.e., tuples) from the crowd to complete a database. As a running example, suppose that a market research analyst wants to study restaurants in New York City (NYC). The analyst can pose an entity collection query to publish crowdsourcing tasks of collecting restaurants in the city from the crowd. Crowdsourced entity collection has many applications, including knowledge base completion, structured data collection, etc.

A fundamental challenge of crowdsourced entity collection is the "open world" nature of crowdsourcing which may return unbounded amount of answers [38]. Recently, some approaches have been proposed to address the challenge. Trushkowsky et al. focused on estimating the coverage of the current entity set collected from the crowd [38]. Chung et al. have extended the estimation techniques to support aggregate queries, such as SUM, AVG, MAX / MIN, etc [6]. Rekatsinas et al. extended the data model to a structured domain with hierarchical structure, and aimed to maximize collection coverage under a budget [36]. Park and Widom developed a general framework CrowdFill that shows a partially filled table and asks the crowd to contribute new entities, fill empty cells, and up-vote / down-vote existing entities [35]. However, the studies do not pay much attention to the underlying distribution of the collected entities, which is often indispensable in the process of data collection.

Evidently, in many real scenarios, users may have *distribution requirements* on the entities collected from the crowd. One application is crowdsourced point-of-interest (POI) collection [17], which asks the crowd to submit locations with additional information, such as road-side parkings. Naturally, users often want the collected POIs (e.g., road-side parkings) to be evenly distributed in an area, instead of following a skew spatial distribution (e.g., only containing POIs near some popular regions). Distribution requirement is also common in market research. For example, suppose that a real estate expert wants to curate a list of representative houses via crowdsourcing. To make the research more

- *J. Fan, J. Yang and X. Du are with the Key Lab of Data Engineering and Knowledge Engineering and the School of Information, Renmin University of China. E-mails:{fanj, jingru, duyong}@ruc.edu.cn*
- *Z. Wei is with the School of Information, Renmin University of China and Beijing Key Laboratory of Big Data Management and Analysis Methods. Email: zhewei@ruc.edu.cn*
- *D. Zhang is with the School of Computer Science and Engineering, University of Electronic Science and Technology of China. Email: zhangdo@uestc.edu.cn*

comprehensive, the expert may want to include various house types, such as number of bedrooms, and would like to specify an expected distribution of the types based on her experience of the market. Moreover, distribution requirement is also desirable in other data collection tasks. For example, a university who wants to find faculty candidates from the job market is likely to have an expected distribution requirement on *specialization* of the applicants, such as `machine learning`, `database`, etc., based on its development directions.

Motivated by this requirement, we study a new research problem, *distribution-aware crowdsourced entity collection* (CROWDDEC), in this paper. Given a user-specific distribution requirement, a CROWDDEC query aims to find the best crowdsourcing strategy that makes the collected data to satisfy the distribution as much as possible, i.e., minimizing difference of the collected entity distribution from the expected distribution.

This CROWDDEC query is quite challenging to answer due to the openness of crowdsourcing, i.e., the entities are collected by an unknown group of people (i.e., workers). First, each individual worker may have its own *bias* of data collection, leading to diverse distributions across different workers. For instance, a worker only collects POIs in a particular spot, while another one randomly provides POIs in an area. Second, it is known that workers may have unevenly contributions: some workers, referred as *streakers* in [38], may provide significantly more entities than other workers. Therefore, the distribution of the collected entities will become *unpredictable* if no effective strategy is utilized to control the crowdsourcing process.

To tackle the difficulties in answering CROWDDEC, we introduce an adaptive crowdsourcing approach. The approach on-the-fly estimates the underlying entity distribution of each worker based on the observed entities the worker has submitted. Then, it selects an optimal set of workers such that entities to be provided by these workers would minimize the difference from the user-expected distribution. Moreover, once a worker submits her answers, it adjusts its estimation of the underlying distributions of workers to improve the subsequent worker selection. In such a way, the approach continuously estimates workers behaviors and adaptively selects workers, so as to approximate the collected entities to the expected distribution. To support this approach, we first formalize "difference" of entity distribution from the expected distribution using Kullback-Leibler (KL) divergence and formulate the CROWDDEC problem. Then, we introduce a probabilistic framework and effective statistical methods to estimate the underlying entity distribution of a worker, based on the worker's "history", i.e., the entities already collected by the worker. Next, by using the estimates, we select an optimal set of workers to minimize the KL divergence. We have proved the optimal worker selection problem is NP-complete, and developed a best-effort algorithm to find the exact solution and an approximate local search algorithm for instant worker selection. We conducted experiments on a real crowdsourcing platform, i.e., Amazon Mechanical Turk (AMT), and the experimental results show the performance superiority of our proposed techniques (Section 5).

To summarize, we make the following contributions.

(1) To the best of our knowledge, this is the first paper to study the problem of distribution-aware crowdsourced entity collection. We formalize the problem and introduce an adaptive crowdsourcing approach to solve it (see Section 2).

(2) We devise a probabilistic framework and effective statistical methods for estimating the underlying distribution of entities provided a worker based on the observed entities already provided by the worker (see Section 3).

(3) We have proved that the problem of optimal worker selection is NP-complete and developed a best-effort algorithm to find the exact solution and an approximate local search algorithm for instant worker selection (see Section 4).

2 OVERVIEW OF CROWDDEC

In this section, we present an overview of CROWDDEC. We define the distribution-aware entity collection query in Section 2.1, introduce a crowdsourcing model in Section 2.2, and propose an adaptive worker selection framework to fulfill CROWDDEC in Section 2.3.

2.1 Distribution-Aware Entity Collection

Consider entities in a specific data domain (e.g., movie), and an attribute A of the entities (e.g., year of the movie) with a value domain, $\Omega = \{a_1, a_2, \dots a_n\}$. We assume that Ω is already known by design, which is common in many practical applications. We will take the problem with unknown attribute domains in future work.

This paper studies the problem of collecting a set of entities, denoted by S, in the data domain. In particular, we would like to measure the *distribution* of the collected entities in S w.r.t. attribute A, which is defined as the ratios of entities having the attribute values:

Definition 1 (Entity Distribution). Entity distribution of an entity set S w.r.t. attribute A is a set of ratios defined on domain Ω, $\Phi_A^S = \{\phi_1^S, \phi_2^S, \dots \phi_n^S\}$. The i-th ratio ϕ_i^S is defined as

$$\phi_i^S = \frac{\sum_{e \in S} \mathbf{1}[e.A = a_i]}{|S|}, \tag{1}$$

where $\mathbf{1}[event]$ is an indicator function equal to 1 if the event occurs and 0 otherwise, and $\sum_{i=1}^{n} \phi_i^S = 1$.

As mentioned in Section 1, users often have distribution requirements on the collected entities w.r.t. attribute A. Therefore, we propose to study the *distribution-aware entity collection query*. In such a query, one is given an expected distribution w.r.t. attribute A, denoted by $\Psi_A = \{\psi_1, \psi_2, \dots \psi_n\}$ and the number k of entities to be collected. The answer of this query is a set S of entities with size k ($|S| = k$) such that the entity distribution Φ_A^S is as close to Ψ_A as possible. For ease of presentation, we use Φ^S and Ψ to represent Φ_A^S and Ψ_A respectively if the context is clear.

Definition 2 (Distribution-Aware Entity Collection). Given an expected distribution $\Psi = \{\psi_1, \psi_2, \dots \psi_n\}$ and the number k of entities to be collected, a distribution-aware entity collection query collects a set of entities S with size $|S| = k$ that minimizes the *difference* $D(\Psi, \Phi^S)$ of entity distribution Φ^S from the expected distribution Ψ, i.e., $S^* = \arg_{S,|S|=k} \min D(\Psi, \Phi^S)$.

TABLE 1
An Example of Restaurant Entities

ID	Region	ID	Region	ID	Region
e_1	South	e_6	North	e_{11}	Center
e_2	South	e_7	North	e_{12}	Center
e_3	South	e_8	North	e_{13}	Center
e_4	South	e_9	North	e_{14}	Center
e_5	South	e_{10}	North	e_{15}	Center

We use the well-known Kullback-Leibler (KL) divergence [20] to measure the above difference. Formally, the KL divergence of Φ^S from Ψ, denoted by $D_{\text{KL}}(\Psi||\Phi^S)$, is a measure of the information gained when one revises one's beliefs from Φ^S to Ψ, i.e.,

$$D_{\text{KL}}(\Psi||\Phi^S) = \sum_{i=1}^{n} \psi_i \cdot \log \frac{\psi_i}{\phi_i^S}. \quad (2)$$

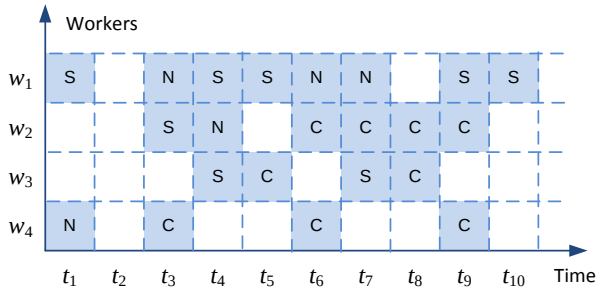
Example 1. Table 1 provides some entities in the `Restaurant` domain and an attribute `Region` with {`South`, `North`, `Center`} as its value domain Ω. Let us consider a set S_1 of entities $\{e_1, e_6, e_7, e_{11}, e_{12}, e_{13}\}$: its entity distribution w.r.t `Region` is $\{\frac{1}{6}, \frac{1}{3}, \frac{1}{2}\}$ as there are one restaurant in `South`, two in `North` and three in `Center`. Given an expected distribution $\Psi = \{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\}$, the KL divergence $D_{\text{KL}}(\Psi||\Phi^{S_1})$ of Φ^{S_1} from Ψ is 0.27. Based on this, we consider a distribution-aware entity collection query with Ψ and a number 6. The answer of this query is a set S^* of entities such that $|S^*| = 6$ and the difference $D_{\text{KL}}(\Psi||\Phi^{S^*})$ is minimized[1].

2.2 Crowd Model for Crowdsourced Entity Collection

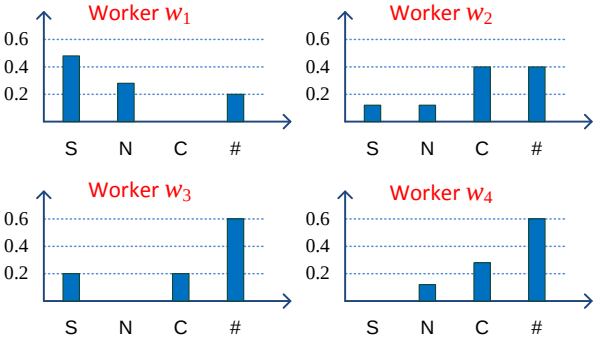
We employ crowdsourcing for entity collection. We publish *human intelligence tasks* (HITs) on existing crowdsourcing platforms, such as Amazon Mechanical Turk[2] (AMT) to solicit a set of *workers*, denoted by $W = \{w_1, w_2, \ldots, w_m\}$, where each HIT asks a worker to submit "one more" entity with a value of attribute A from domain Ω. An example of such HIT would be asking the worker to provide "a restaurant in NYC with attribute `Region`".

The crowdsourcing process works iteratively: at each time point t, some workers in W request for HITs which will be assigned by our worker selection framework (see Section 2.3). After they submit their entities, we pay them and proceed to the next time point $t+1$ till k entities have been collected. In such a way, we finally obtain a collection of entity sets, $S_1, S_2, \ldots, S_m$ from the workers, where S_j is the entities submitted by worker w_j. Then, we consolidate the entities from the workers to generate an integrated set S. Note that this paper assumes that data integration issues involved in generating S, e.g., entity resolution and attribute consolidation, can be perfectly addressed manually or by existing techniques [8], which is orthogonal to this paper.

Example 2. Figure 1(a) provides an example of crowdsourcing process with four workers, where each row is entities submitted by a worker and each column is a time point. Each *labelled* cell represents a submitted entity where the label is its attribute value (`S`, `N` and `C` represent `South`, `North` and `Center` respectively), while a *blank* cell means no entity is submitted. Considering to collect 10 entities, we terminate the crowdsourcing process at time point t_5 and obtain an integrated entity set S.



(a) Entities collected from example workers.



(b) Underlying distribution of the workers.

Fig. 1. The crowd model for crowdsourced entity collection (where `S`, `N`, `C` and # represent `South`, `North`, `Center` and *no answer*).

Based on the experiments on real datasets (Section 5.2), we have observed that crowdsourcing has its own characteristics for entity collection, which is discussed as follows.

1) *Diverse Bias*: As workers have different backgrounds, each of them may have her own *bias* of entities she knows, leading to quite *diverse* entity distributions across different workers. For example, worker w_1 in Figure 1(a) may have bias on restaurants in `South` and `North`, while w_2 know better for those in `Center`. In our experiments, we measure KL divergence for each pair of workers, and observe that most of the pairs have quite large divergence values.

2) *Uneven Contribution*: It is known that crowdsourcing workers usually have unevenly contributions [38], [6]: some workers, referred as *streakers* in [38], may provide significantly more entities than the other workers. In our example shown in Figure 1(a), worker w_1 is likely to be a streaker as she contributes more than the other ones. In our real dataset of collecting movie entities, one worker submits 1250 entities, while most of the workers only provide tens of entities (see Section 5.2 for more details).

The above observations imply that contributions from workers may be rather irregular, thus making it very challenging to model their behaviors. To address the challenge, we introduce a probabilistic model to model worker's behavior for entity collection. The basic idea of this model is to consider the generation of entity set S_j from w_j as a *sampling process* from an unknown entity distribution. Our model has a property of incorporating both diverse bias and uneven contribution into consideration.

1. Note that there may be more than one sets of collected entities that minimize the distribution difference, and any one of them can be regarded as the answer of the query in this paper.

2. https://www.mturk.com/

TABLE 2
Frequently Used Notations

e, A, Ω	entity, attribute, attribute value domain
Φ_A^S	distribution of entity set S w.r.t. A
Ψ_A	an expected entity distribution w.r.t. A
$D_{\mathrm{KL}}(\Psi\|\|\Phi^S)$	KL divergence of Φ^S from Ψ.
w_j, S_j	crowdsourcing worker, entity set of worker
$\mathcal{P}^{w_j}$	w_j's probabilistic model of entity collection

Formally, let $\mathcal{P}^{w_j}$ denote worker w_j's *underlying* probability distribution at a time point t. In particular, $\mathcal{P}^{w_j}$ is defined as $\{p_1^j, p_2^j, \ldots, p_n^j, p_\#^j\}$, where p_i^j is the probability that w_j provides an entity with attribute value a_i, $p_\#^j$ is the probability that w_j provides no entity and $\sum_i p_i^j + p_\#^j = 1$. Based on this, our model considers that entity set S_j is generated by sampling entities from distribution $\mathcal{P}^{w_j}$ in the following way. At any time point t, we throw a dice with $|\Omega|+1$ faces, $\{a_1, a_2, \ldots, a_n, \#\}$, where the probabilities of the result are $\mathcal{P}^{w_j}$. If the result of dice throwing is a_i, w_j provides an entity e with attribute value a_i, and otherwise (i.e., the result is $\#$), w_j does not provide any entity.

Example 3. Figure 1(b) provides the underlying entity distributions of the four workers in Figure 1(a). We can see that the model can formalize both diverse bias and uneven contribution of workers: On the one hand, the distributions of the workers are obviously quite different from each other. On the other hand, workers have various probability $p_\#^w$ that models "unwillingness" of contribution. For example, $p_\#^1$ of the streaker w_1 is much smaller than those of the other workers, which means w_1 is more likely to be zealous to provide more entities.

To facilitate our presentation, all the frequently used notations are listed in Table 2.

2.3 An Adaptive Worker Selection Framework

This section presents an adaptive worker selection framework to support distribution-aware crowdsourced entity collection. The framework mainly addresses two challenges. First, the underlying distribution $\mathcal{P}^{w_j}$ of worker w_j is actually *unknown*, resulting in a challenge of accurate $\mathcal{P}^{w_j}$ estimation. Second, as the diversity of $\mathcal{P}^{w_j}$ across workers is significant, it calls for an effective approach to finding a subset of W such that the "aggregated" distribution provided by these workers is as close to the expected Ψ as possible.

Algorithm 1 shows the pseudo-code of the framework, which takes as input an expected distribution Ψ coupled with a number k and outputs an entity set S. Initially, it constructs an empty entity set S and worker set W, and then starts to select workers *periodically in time*[3].

At each time point t, it first identifies the current *active* workers $\tilde{W}$ who are ready to work on the HITs (line 3). Like our previous work in [9], we may use different methods to this end. For example, if the span from the last time point when a worker requests for HITs to the current time point is smaller than a threshold (e.g., 15 minutes), we consider the worker is still active and can be assigned with HITs. For

3. We can define various granularity for time points, such as seconds, minutes, hours, etc. See more details of time point settings in our experiments.

Algorithm 1: CROWDDEC-FRAMEWORK (Ψ, k)

Input: Ψ: expected distribution; k: a number
Output: S: a set of collected entities

```
1  Initialize entity set S ← ∅, worker set W ← ∅ ;
2  for each time point t do
3      W̃ ← DETECTACTIVES (t) ;
4      for each w_j ∈ W̃ do
5          if w_j ∉ W then S_j ← QUALIFICATION (w_j) ;
6          P̂^{w_j} ← ESTIMATE (w_j, S_j) ;
7      W ← W ∪ W̃ ;
8      W* ← SELECTWORKERS (W̃, {P̂^{w_j}}, Ψ, S) ;
9      for each action from worker w_j do
10         if w_j ∉ W* then continue ;
11         if w_j.action = req then Assign w_j an HIT;
12         else if w_j.action = ans then
13             Insert entity e from w_j into S_j ;
14             INTEGRATEENTITY (S, e) ;
15             if |S| = k then return S ;
```

each active worker w_j, the framework checks if w_j is a *new* worker who has not answered our HITs yet (i.e., $w_j \notin W$). If so, it assigns w_j qualification HITs for collecting some initial entities S_j (line 5). Intuitively, a qualification HIT asks w_j to fill attribute values of a pre-defined entity list. The purpose of the qualification is two-fold. First, as we know the ground-truth attribute values of the entities in the list, if w_j provides too many wrong answers, we block this "low-quality" worker. Second, the qualification HIT can address the "cold-start" problem: it helps us to collect an initial set of entities from w_j to facilitate the following estimation.

Next, the framework estimates w_j's underlying probabilistic model $\mathcal{P}^{w_j}$ from the "historical" entities S_j submitted by w_j (line 6). Then, it selects a subset W^* of active workers such that the difference from Ψ is minimized after collecting entities from W^* (line 8). Based on W^*, it selectively assigns HITs to workers at time point t as follows. For each request from a worker w_j, the framework "blocks" the request[4] if $w_j \notin W^*$. If w_j is selected ($w_j \in W^*$), our framework assigns HITs and collects the entities from w_j.

Finally, it integrates the entities into S by using existing data integration techniques [8] (line 14) and examines if the number of collected entities reaches k (line 15). If enough entities have been collected, it terminates. Otherwise, it proceeds to the next time point for further collection.

From the above algorithm, we can see that the proposed framework is *adaptive* since it continuously estimates workers' probabilistic models and selects workers based on the current estimation. This allows us to on-the-fly utilize the most appropriate set of workers by finding better workers and eliminating worse ones. Note that the implementation of Algorithm 1 on the crowdsourcing platform AMT can utilize the "external question" mechanism that manages the HITs and assignment in our own web server (refer to [9] for more details of the mechanism).

Example 4. We use the example in Figure 1 to illustrate

4. For "blocking", we can actually provide user-friendly messages, e.g., "We currently do not have questions. Please try later..."

the framework. Given an expected entity distribution $\Psi = \{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\}$ and a number 12, the framework selects workers periodically as follows. At any time, say t_6, it first identifies active workers $\{w_1, w_2, w_4\}$. If any worker w_j is a new worker the framework uses qualification to initialize entity set S_j. Then, it estimates $\hat{\mathcal{P}^{w_1}}$, $\hat{\mathcal{P}^{w_2}}$ and $\hat{\mathcal{P}^{w_4}}$ based on the sets S_1, S_2 and S_4. Suppose that the current estimation is accurate to capture the underlying distribution in Figure 1(b). Then, it may only select workers w_2 and w_4 while blocking w_1, as the relative ratio of restaurants in `Center` needs to be increased. Finally, if 12 entities have been collected, the framework terminates and returns the integrated entity set S.

This paper focuses on developing techniques to address two challenging problems in the framework. First, Section 3 introduces effective approaches to estimating worker w_j's underlying probabilistic model $\mathcal{P}^{w_j}$ (i.e., function ESTIMATE in line 6). Second, Section 4 discusses optimal worker set selection to minimize the difference from expected distribution Ψ (i.e., function SELECTWORKERS in line 8).

3 Worker Probabilistic Model Estimation

This section presents an estimation problem for computing probabilistic model $\mathcal{P}^{w_j}$ of a worker w_j. The basic idea of the estimation is to utilize w_j's *historical* entities, i.e., the observed entity set S_j from w_j. Note that historical entity set S_j also includes $\#$ as a special entity which means no answer from w_j is provided. We consider that the observed S_j is also generated from the underlying model $\mathcal{P}^{w_j}$, and thus we can infer $\mathcal{P}^{w_j}$ from this observation. This estimation problem can be formally defined as follows.

Definition 3 (Worker Probabilistic Model Estimation). Given a worker w_j and her historical entity set S_j, it aims to estimate w_j's underlying probabilistic model $\hat{\mathcal{P}}^{w_j} = \{\hat{p}_1^j, \hat{p}_2^j, \ldots, \hat{p}_n^j, \hat{p}_\#^j\}$, where $\hat{p}_i^j$ (or $\hat{p}_\#^j$) is the *estimated* probability of sampling an entity e with attribute value a_i (or *no answer*) from w_j given observation of w_j's historical entity set S_j, i.e., $\hat{p}_i^j = Pr\{a_i|S_j\}$.

Take worker w_1 in Figure 1 as an example. The corresponding S_1 is the the first row of entities provided by w_1 in Figure 1(a). For instance, at time point t_5, $S_1 = \{$`S`,`#`,`N`,`S`,`S`$\}$. Then, the problem aims to estimate $\hat{\mathcal{P}}^{w_1}$ of w_1 in Figure 1(b) based on S_1. This section introduces estimation techniques to address the problem. We discuss an empirical probability method in Section 3.1, a *Good-Turing* estimation in Section 3.2, and a hybrid method in Section 3.3.

3.1 Empirical Probability Estimation

A simple method for estimating $\hat{\mathcal{P}}^{w_j}$ is to use the *empirical probability*, or *relative frequency* in S_j. For example, at time point t_5 in Figure 1(a), as worker w_1 has provided three entities of `South`, one of `North`, zero of `Center` and one of $\#$, the estimated probabilistic model $\hat{\mathcal{P}}^{w_1} = \{\frac{3}{5}, \frac{1}{5}, 0, \frac{1}{5}\}$. Similarly, we have $\hat{\mathcal{P}}^{w_2} = \{\frac{1}{5}, \frac{1}{5}, 0, \frac{3}{5}\}$, $\hat{\mathcal{P}}^{w_3} = \{\frac{1}{5}, 0, \frac{1}{5}, \frac{3}{5}\}$ and $\hat{\mathcal{P}}^{w_4} = \{0, \frac{1}{5}, \frac{1}{5}, \frac{3}{5}\}$ at time point t_5. However, empirical probability estimation is usually not accurate, as a sample S_j with limited size may not capture the underlying distribution. Moreover, it may also lead to the "sparsity" problem that assigns "zero" to many attribute values that are not observable in S_j, which actually deviates the true underlying probabilistic distribution.

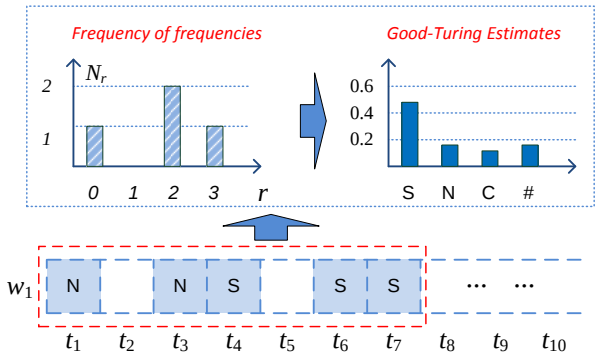


Fig. 2. Illustration of Good-Turing estimation.

To overcome the above drawbacks, a commonly used approach is *smoothing* that considers the fact that S_j does not represent the "whole-world" of entities from w_j. In our work, we adopt the well-known *Laplace smoothing* [23], i.e.,

$$\hat{p}_i^j = \frac{\sum_{e \in S_j} \mathbf{1}[e.A = a_i] + \alpha}{|S_j| + \alpha \cdot (|\Omega| + 1)}, \tag{3}$$

where $\alpha \geq 0$ is a *pseudocount* that is used for smoothing. If $\alpha = 1$, Equation (3) is also called *add-one smoothing*. Take w_2 as an example again: given $\alpha = 1$, the estimated worker model at t_5 after smoothing is $\hat{\mathcal{P}}^{w_2} = \{\frac{2}{9}, \frac{2}{9}, \frac{1}{9}, \frac{4}{9}\}$.

3.2 Good-Turing Estimation

Good-Turing estimation is also commonly used for estimating probabilistic distribution [13], [29], [30]. The basic idea of this technique is to utilize the "frequency of frequencies" in observed set S_j, i.e., the number of distinct attribute values having the same occurrence in S_j. As shown in Figure 2, there are two distinct values, e.g., `N` and $\#$, that occurs 2 times. Subsequently, we obtain the "frequency of frequencies" in the top left corner, which can help us estimate the underlying probabilistic distribution.

More formally, let $f_{S_j}(a_i)$ denote the frequency of attribute value a_i in S_j[5], i.e., the number of entities in S_j having attribute value a_i. For ease of presentation, we use $f(a_i)$ to represent $f_{S_j}(a_i)$ if the context is clear. Let N denote the size of S_j ($N = |S_j|$) and N_r denote the number of attribute values with frequency $f(a_i) = r$. Then, given the observation $f(a_i) = r$, the Good-Turing method estimates probability $\hat{p}_i^j$ as the *conditional expectation* of $\hat{p}_i^j$ for those attribute values which occur r times in S_j, i.e.,

$$\hat{p}_i^j = \mathbb{E}[p_i^j | f(a_i) = r] = \sum_i p_i^j \cdot Pr\{p_i^j | f(a_i) = r\}, \tag{4}$$

Now, we show the estimate in Equation (4) can be derived by using aforementioned N_r as follows.

Lemma 1. Given the observation $f(a_i) = r$, a Good-Turing estimate $\hat{p}_i^j$ can be derived as

$$\hat{p}_i^j = \frac{r+1}{N+1} \cdot \frac{\mathbb{E}_{N+1}[N_{r+1}]}{\mathbb{E}_N[N_r]}, \tag{5}$$

5. For ease of presentation, we consider $\#$ as a special case of a_i.

where $\mathbb{E}_N[N_r]$ is the expectation of the number of distinct attribute values which occur exact r times in entity set S_j of worker w_j with size N.

Proof 1. By applying the Bayesian formula, we can transform Equation (4) as follows.

$$\hat{p}_i^j = \sum_i p_i^j \cdot Pr\{p_i^j | f(a_i) = r\}$$
$$= \sum_i p_i^j \cdot \frac{Pr\{f(a_i) = r | p_i^j\} \cdot Pr\{p_i^j\}}{\sum_{i'} Pr\{f(a_{i'}) = r | p_{i'}^j\} \cdot Pr\{p_{i'}^j\}},$$

where $Pr\{p_i^j\}$ is a *prior* of attribute values. In our work, we assume uniform priors, i.e., $Pr\{p_i^j\} = 1/(|\Omega| + 1)$. $Pr\{f(a_i) = r|p_i^j\}$ is the probability that a_i occurs r times in S_j given its underlying probability p_i^j. Based on our sampling model mentioned in Section 2.2, it can be computed as $Pr\{f(a_i) = r|p_i^j\} = C_N^r \cdot (p_i^j)^r \cdot (1-p_i^j)^{N-r}$, under the assumption of independent sampling. Thus, the estimate $\hat{p}_i^j$ can be further derived as

$$\hat{p}_i^j = \sum_i p_i^j \cdot \frac{C_N^r \cdot (p_i^j)^r \cdot (1-p_i^j)^{N-r}}{\sum_{i'} C_N^r \cdot (p_{i'}^j)^r \cdot (1-p_{i'}^j)^{N-r}}$$
$$= \frac{r+1}{N+1} \cdot \frac{\sum_i C_{N+1}^{r+1} \cdot (p_i^j)^{r+1} \cdot (1-p_i^j)^{N-r}}{\sum_{i'} C_N^r \cdot (p_{i'}^j)^r \cdot (1-p_{i'}^j)^{N-r}}$$
$$= \frac{r+1}{N+1} \cdot \frac{\mathbb{E}_{N+1}[N_{r+1}]}{\mathbb{E}_N[N_r]}.$$

Hence, we prove the lemma. □

Now, we show the Good-Turing estimation satisfies the probabilistic distribution property, i.e.,

Theorem 1. The Good-Turing estimation in Equation (5) satisfies the property $\sum_i \hat{p}_i^j = 1$.

Proof 2. As the number of the a_i's that occurs exactly r times is N_r, we have

$$\sum_i \hat{p}_i^j = \sum_{r=0}^{N} \mathbb{E}_N[N_r] \cdot \frac{r+1}{N+1} \cdot \frac{\mathbb{E}_{N+1}[N_{r+1}]}{\mathbb{E}_N[N_r]}$$
$$= \frac{\sum_{r=0}^{N} (r+1) \cdot \mathbb{E}_{N+1}[N_{r+1}]}{N+1}.$$

Obviously, we have $\sum_{r=0}^{N} (r+1) \cdot \mathbb{E}_{N+1}[N_{r+1}] = N+1$. Thus, we prove the theorem. □

The non-trivial part in Equation (5) is $\mathbb{E}_N[N_r]$, which is the expected number of the attribute values which occur exact r times in S_j with size N. For example, given $r = 1$, $\mathbb{E}_N[N_1]$ is the *expected* number of *singletons* in S_j with $|S_j| = N$. The expectation is difficult to compute unless we know p_i^j, which is actually what we want to estimate. To address this issue, we adopt an existing method [29] to compute the Good-Turing estimates $\hat{p}_i^j$ as

$$\hat{p}_i^j = \frac{r+1}{N} \cdot \frac{N_{r+1}}{N_r}. \quad (6)$$

Obviously, N_r could be *zero* in Equation (6), and thus smoothing techniques also need to be applied. We use the smoothing technique introduced in [29] to use $N_r' = \max\{N_r, 1\}$, and thus the estimation becomes

$$\hat{p}_i^j = ((r+1) \cdot N_{r+1}')/(N' \cdot N_r'), \quad (7)$$

where $N_r' = \max\{N_r, 1\}$ and N' is a term used for normalization and $N' = \sum_{r:N_r>0} N_r \cdot (r+1) \cdot N_{r+1}'/N_r'$.

Example 5. Figure 2 shows an example to illustrate Good-Turing estimation. Given the observation of S_1 at time point t_7, it first generates the "frequency of frequencies" as shown in the top left corner, i.e., $\{N_0 = 1, N_2 = 2, N_3 = 1\}$. Then, it computes the estimates by using Equation (7) and obtains $\{0.5, 0.1875, 0.125, 0.1875\}$, compared with the estimates computed by the empirical estimation method, $\{0.36, 0.27, 0.1, 0.27\}$.

3.3 A Hybrid Estimation Strategy

As discussed in the previous work [29], [30], Good-Turing estimation performs well for attribute values with low frequencies, i.e., small values of r, while it may produce unsatisfactory results for larger r. This can be explained by the fact that N_{r+1} tends to be zero for large values of r, and thus it becomes inappropriate to approximate the expectation $\mathbb{E}_{N+1}[N_{r+1}]$ in Equation (5).

To address the above problem, we introduce a hybrid estimation method that combines Good-Turing and empirical estimation. The basic idea is to utilize Good-Turing for low values of r and empirical method for high values of r. Formally, by comparing r and N_{r+1}, the hybrid method produces the estimate $\hat{p}_i^j$ as

$$\hat{p}_i^j = \begin{cases} r/(N') & r > N_{r+1}, \\ ((r+1) \cdot N_{r+1}')/(N' \cdot N_r'), & 0 \le r \le N_{r+1}. \end{cases} \quad (8)$$

where N' is introduced for normalization, i.e., $N' = \sum_{r \le N_{r+1}} N_r \cdot (r+1) \cdot N_{r+1}'/N_r' + \sum_{r > N_{r+1}} N_r \cdot r$. In practice, we also interpolate a uniform frequency $1/(1 + |\Omega|)$ for smoothing. A recent theoretical study [30] has shown that the hybrid estimator is uniformly optimal for estimating every distribution in terms of KL-divergence. We also demonstrate its effectiveness via experiments in Section 5.

4 Optimal Worker Selection

This section presents an approach to optimal worker selection for minimizing divergence from the expected distribution in the query. We first formalize this problem in Section 4.1, prove its hardness in Section 4.2, and develop a best-effort algorithm for finding exact solution in Section 4.3 as well as an approximate solution in Section 4.4.

4.1 Formalization of Optimal Worker Selection

Intuitively, *optimal worker selection* aims to "approximate" the expected distribution Ψ_A at the best by incorporating entities of some judiciously selected workers. We slightly abuse the notations to also use W to denote the set of current active workers if the context is clear. Let W' be a subset of W and $\Phi_A^{W'}$ be distribution of entities to be collected by W'. The optimal worker selection problem is defined as follows.

Definition 4 (Optimal Worker Selection). Given an expected distribution Ψ_A and current active worker set W, it finds a set of workers $W^* \subseteq W$ that minimizes KL divergence from Ψ_A, i.e., $W^* = \arg_{W' \subseteq W} \min D_{\mathrm{KL}}(\Psi_A || \Phi_A^{W'})$.

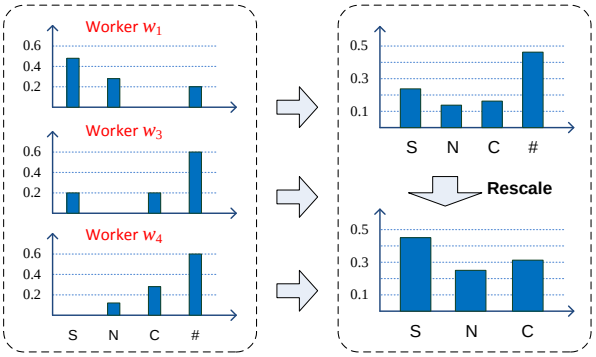


Fig. 3. Example of worker model aggregation.

Computation of Distribution $\Phi_A^{W'}$. The key in Definition 4 is to compute distribution $\Phi_A^{W'}$ of the entities to be provided by the workers in W'. Based on our worker model, this distribution mainly depends on how the entities are sampled from an *aggregated probabalistic model* of these workers. The basic idea is illustrated in Figure 3, which consists of the following two steps.

Step 1 - Aggregating Worker Models. We first compute an *aggregated probabalistic model* $\mathcal{P}^{W'}$ by combining the estimated $\hat{\mathcal{P}}^{w_j}$ of each individual worker w_j in W'. Intuitively, each probability in $\mathcal{P}^{W'}$ corresponds to the event of sampling an entity from W' with attribute value a_i (or $\#$). This process can be considered as throwing multiple dices, each of which corresponds to a worker w_j in W' and has $|\Omega|+1$ faces as $\{a_1, a_2, \ldots, a_n, \#\}$. After throwing, we count the result of how many dices with faces a_1, a_2, ..., $\#$ respectively. Then, we compute $\mathcal{P}^{W'}$ by normalizing the obtained numbers. As we already capture the "uneven contribution" in $\hat{\mathcal{P}}^{w_j}$, we can simply use $1/|W'|$ for the normalization, i.e., $p_i^{W'} = \sum_{w_j \in W'} \hat{p}_i^j \cdot \frac{1}{|W'|}$.

Step 2 - Rescaling to Entity Distribution. Notice that the aggregated model $\mathcal{P}^{W'}$ is not equivalent to entity distribution $\Phi_A^{W'}$, as the former contains probability $p_\#^{W'}$ of sampling *no entity* which obviously has no effect on $\Phi_A^{W'}$. Thus, we need to "rescale" $\mathcal{P}^{W'}$ to obtain $\Phi_A^{W'}$ by only considering the cases in which W' provides entities, i.e.,

$$\phi_i^{W'} = \frac{p_i^{W'}}{\sum_{l=1}^n p_l^{W'}} = \frac{\sum_{w_j \in W'} \hat{p}_i^j}{\sum_{w_j \in W'} \sum_{l=1}^n \hat{p}_i^j}. \quad (9)$$

Example 6. Consider the example shown in Figure 3 with three workers $W' = \{w_1, w_3, w_4\}$. To compute $\Phi_A^{W'}$, we first combine $\hat{\mathcal{P}}^{w_1}$, $\hat{\mathcal{P}}^{w_3}$ and $\hat{\mathcal{P}}^{w_4}$ to generate an aggregated model $\mathcal{P}^{W'}$, then rescale the probabilities by applying Equation (9), and finally obtain the entity distribution $\Phi_A^{W'} = \{\frac{7}{16}, \frac{4}{16}, \frac{5}{16}\}$.

Transforming Optimization Objective. Next, we transform the original optimization objective $D_{\mathrm{KL}}(\Psi_A||\Phi_A^{W'})$ into a more compact objective. Based on the definition of KL divergence in Equation (2), we have

$$D_{\mathrm{KL}}(\Psi_A||\Phi_A^{W'}) = \sum_{i=1}^n \psi_i \log \psi_i - \sum_{i=1}^n \psi_i \log \phi_i^{W'}. \quad (10)$$

As $\sum_{i=1}^n \psi_i \log \psi_i$ is a constant w.r.t. worker set W', minimization of $D_{\mathrm{KL}}(\Psi_A||\Phi_A^{W'})$ can be transformed into *maximizing* $\sum_{i=1}^n \psi_i \log \phi_i^{W'}$. For ease of presentation, we call it *impact* of W' and use $\mathcal{I}(W')$ to denote $\sum_{i=1}^n \psi_i \log \phi_i^{W'}$. Thus, the problem of optimal worker selection is now transformed to selecting a worker set $W^* \subseteq W$ that maximizes the *impact*, i.e., $W^* = \arg_{W'} \max \mathcal{I}(W') = \arg_{W'} \max \sum_{i=1}^n \psi_i \log \phi_i^{W'}$.

Discussion. Our method can be extended to the case that distribution Φ_A^S of the existing collected entity set S deviates the expected distribution Ψ_A. In this case, the objective of optimal worker selection is not minimizing $D_{\mathrm{KL}}(\Psi_A||\Phi_A^{W'})$ any more. Instead, it aims to "adjust" Φ_A^S to approximate Ψ_A by incorporating entities from worker set W' into S, which can be formalized as $W^* = \arg_{W'} \min D_{\mathrm{KL}}(\Psi_A||\Phi_A^{W'} \oplus \Phi_A^S)$, where $\Phi_A^{W'} \oplus \Phi_A^S$ is distribution of combining existing entities in S and those to be collected from W'. In the remainder of this section, for ease of presentation, we only consider the case that minimizes $D_{\mathrm{KL}}(\Psi_A||\Phi_A^{W'})$ without Φ_A^S.

4.2 Hardness of Optimal Worker Selection

We show that the optimal worker selection problem in Definition 4 is a NP-complete problem, and thus we cannot hope for a polynomial time algorithm.

Theorem 2. The worker selection problem is NP-Complete.

We reduce the *Subset Sum* problem to our problem. The input of the Subset Sum problem is a set S of integers and a target integer t, and the goal is to determine if there is any subset of S that sums up to t. To construct the reduction, we consider the *Fixed Size Subset Sum (FSSS)* problem, which essentially fixes the size of the subset.

Definition 5 (Fixed Size Subset Sum). Given a set S of n natural numbers $S = \{s_1, \ldots, s_n\}$, and a target number t, is there a subset S' of S such that $|S'| = k$ and $\sum_{s_i \in S'} s_i = t$?

It is easy to see that if we can solve the FSSS problem in polynomial time for arbitrary k, then we can run the algorithm for $k = 1, \ldots, n$ and solve the Subset Sum problem in polynomial time. We now show that if there is a polynomial time algorithm for the worker selection problem, then we can solve the FSSS problem in polynomial time for arbitrary k, thus proving the NP-completeness of the FSSS problem.

Given an instance of the FSSS problem, we construct an instance of the worker selection problem as follows. Let $s = \sum_{j=1}^n s_j$. The attribute domain Ω consists of two values, a_1 and a_2. We construct a set W of $n+1$ workers as follows. For the first n workers, we define the estimated distribution of worker w_j to be

$$\left\{\hat{p}_1^j = \frac{s_j}{2(k+1)s}, \hat{p}_2^j = 1 - \frac{s_j}{2(k+1)s}\right\},$$

for $j = 1, \ldots, n$. Note that under this construction, the probability of sampling worker w_j samples no entities at each step is $\hat{p}_\#^j = 0$. We define $P_n = \sum_{j=1}^n \hat{p}_1^j$ to be the summation of the estimated probabilities of the first entity over all workers. Note that $P_n = \sum_{j=1}^n \frac{s_j}{2(k+1)s} = \frac{1}{2(k+1)}$, which is a minor probability. We further construct an extra worker w_{n+1} with uniform estimated distribution $\{\hat{p}_1^{n+1} = \frac{1}{2}, \hat{p}_2^{n+1} = \frac{1}{2}\}$.

Moreover, the expected distribution is defined to be

$$\left\{\psi_1 = \frac{\frac{1}{2} + \frac{t}{2(k+1)s}}{(k+1)}, \psi_2 = 1 - \frac{\frac{1}{2} + \frac{t}{2(k+1)s}}{(k+1)}\right\}.$$

To get some intuitions of this construction, we observe that the probability mass for the first entity of the first n workers is dominated by that of the $(n+1)$-th worker. So if there is a subset W^* such that $\frac{\sum_{w_j \in W^*} \hat{p}_1^j}{|W^*|} = \psi_1$, then the $(n+1)$-th worker must reside in W^*. This will ensure that $|W^*|$ cannot deviates from $k+1$, and thus the rest k workers make up the optimal worker set. In particular, we have the following Lemma that states the algorithm that solves the worker selection problem can also solve the FSSS problem.

Lemma 2. Suppose the worker selection problem finds a subset W^* of W that maximizes the impact. Then, the original FSSS problem has a solution if and only if $\frac{\sum_{w_j \in W^*} \hat{p}_1^j}{|W^*|} = \psi_1$.

Proof 3. Recall that the worker selection problem finds a subset W' of W that maximizes the impact $\mathcal{I}(W')$ of worker set W':

$$\psi_1 \log \frac{\sum_{w_j \in W'} \hat{p}_1^j}{|W'|} + \psi_2 \log \frac{\sum_{w_j \in W'} \hat{p}_2^j}{|W'|}. \quad (11)$$

Since $\psi_1 + \psi_2 = 1$ and $\hat{p}_1^j + \hat{p}_2^j = 1$ for $j = 1 \ldots, n+1$, the impact $\mathcal{I}(W')$ equals to

$$\psi_1 \log \frac{\sum_{w_j \in W'} \hat{p}_1^j}{|W'|} + (1-\psi_1) \log \left(1 - \frac{\sum_{w_j \in W'} \hat{p}_1^j}{|W'|}\right). \quad (12)$$

Consider the function $f(x) = \psi_1 \log x + (1-\psi_1)\log(1-x)$. Jensen's inequality suggests that $f(x)$ takes maximum if and only if $x = \psi_1$. It follows that the impact $\mathcal{I}(W')$ takes maximum if and only if $\frac{\sum_{w_j \in W'} \hat{p}_1^j}{|W'|} = \psi_1$.
Now suppose there is a subset $W^* \in W$ that satisfies

$$\frac{\sum_{w_j \in W^*} \hat{p}_1^j}{|W^*|} = \psi_1. \quad (13)$$

The first observation is that the $(n+1)$-th worker w_{n+1} must be selected to W^*. For a proof, notice that if w_{n+1} is not in W^*, then $\frac{\sum_{w_j \in W^*} \hat{p}_1^j}{|W^*|}$ is bounded by the maximum $\hat{p}_1^j$ for $j = 1, \ldots, n$, which is in term bounded by the summation $\sum_{i=1}^{n} \hat{p}_1^j = \frac{1}{2(k+1)}$. Since $\psi_1 > \frac{1}{2(k+1)}$, this is contradicting to equation (13).
The second observation is that W^* contains exactly $(k+1)$ workers. For a proof, we assume $|W^*| \neq k+1$. If $|W^*| \leq k$, we have

$$\frac{\sum_{w_j \in W^*} \hat{p}_1^j}{|W^*|} \geq \frac{\hat{p}_1^{n+1}}{k} \geq \frac{1}{2k}.$$

Notice that

$$\psi_1 = \frac{\frac{1}{2} + \frac{t}{2(k+1)s}}{(k+1)} \leq \frac{\frac{1}{2} + \frac{s}{2ks}}{(k+1)} \leq \frac{1}{2(k+1)} + \frac{1}{2k(k+1)} < \frac{1}{2k},$$

which is contradicting to equation (13). On the other hand, if $|W^*| \geq k+2$, then

$$\frac{\sum_{w_j \in W^*} \hat{p}_1^j}{|W^*|} \leq \frac{\sum_{i=1}^{n} \hat{p}_1^j + \hat{p}_1^{n+1}}{k+2} = \frac{\frac{1}{2(k+1)} + \frac{1}{2}}{(k+2)} < \frac{1}{2(k+1)}.$$

Notice that

$$\psi_1 = \frac{\frac{1}{2} + \frac{t}{2(k+1)s}}{(k+1)} > \frac{1}{2(k+1)}.$$

This also implies a contradiction to equation (13).
Summarizing the two observations, we have $w_{n+1} \in W^*$ and $|W^*| = k+1$. Combining with equation (13), it follows that $\sum_{w_j \in W^*} \hat{p}_1^j = \psi_1(k+1)$, which implies that

$$\sum_{w_j \in W^*, j \neq n+1} \hat{p}_1^j = \psi_1(k+1) - \hat{p}_1^{n+1} = \frac{t}{2(k+1)s}.$$

Plugging $\hat{p}_1^j = \frac{s_j}{2(k+1)s}$, we have

$$\frac{\sum_{w_j \in W^*, j \neq n+1} s_j}{2(k+1)s} = \frac{t}{2(k+1)s},$$

that is,

$$\sum_{w_j \in W^*, j \neq n+1} s_j = t.$$

In other words, if we select the subset $S^* \in S$ such that $S^* = \{s_j \mid w_j \in W^* \setminus \{w_{n+1}\}\}$, then S^* forms a feasible solution of the FSSS problem.
On the other hand, if there is a subset $S^* \in S$ such that $\sum_{s_j \in S^*} s_j = t$, then by picking $W^* = \{w_j \mid s_j \in S^* \cup \{w_{n+1}\}\}$, we construct a worker subset with $\frac{\sum_{w_j \in W^*} \hat{p}_1^j}{|W^*|} = \psi_1$. This worker subset will maximize the impact function $\mathcal{I}(W')$.
This proves that the original FSSS problem has a solution if and only if $\frac{\sum_{w_j \in W^*} \hat{p}_1^j}{|W^*|} = \psi_1$. □

4.3 A Best-Effort Algorithm for Worker Selection

This section presents a best-effort algorithm to compute exact solution for optimal worker selection. The basic idea is to estimate the *upper bound* of any worker set and preferentially compute impact for the worker sets with larger upper bounds, so as to prune insignificant worker sets.

Algorithm 2 provides the pseudo-code. The algorithm initializes a max-heap $\mathcal{H}$ for supporting preferential access to worker sets, which considers that each worker set W^* in heap $\mathcal{H}$ has one of the following two states:

1) `bounded`: upper bound of impact of W^* is estimated;
2) `computed`: exact impact of W^* is computed.

It first inserts an empty worker set into $\mathcal{H}$, and then iteratively accesses elements in $\mathcal{H}$. In each iteration, it pops the top element $\langle W^*, \mathcal{I}(W^*), state\rangle$ from heap $\mathcal{H}$.

1) If the state is `bounded`, it expands W^* to W' by including every worker w in $W - W^*$. In particular, for each expanded worker set W', it estimates bound $\overline{\mathcal{I}}(W')$, which is the upper bound of any superset of W'. Then, it inserts W' with $\overline{\mathcal{I}}(W')$ as well as state `bounded` into $\mathcal{H}$. After that, it computes the exact impact for W^* and inserts it back to $\mathcal{H}$ with state `computed` if W^* is not empty (lines 6 - 12).

Algorithm 2: BESTEFFORTSELECT ($W, \{\hat{\mathcal{P}}^{w_j}\}, \Psi, S$)

Input: W: workers; $\{\hat{\mathcal{P}}^{w_j}\}$: estimated worker models; Ψ: expected distribution; S: collected entities
Output: W^*: a selected worker set

1 Initialize a max-heap $\mathcal{H}$;
2 Insert $\langle \emptyset, -\infty, \texttt{bounded} \rangle$ into $\mathcal{H}$;
3 **while** $\mathcal{H} \neq \emptyset$ **do**
4 $\langle W^*, \mathcal{I}(W^*), state \rangle \leftarrow \mathcal{H}.pop()$;
5 **if** $state = \texttt{bounded}$ **then**
6 **for** *each* $w \in W - W^*$ **do**
7 $W' \leftarrow W^* \cup \{w\}$;
8 $\overline{\mathcal{I}}(W') \leftarrow$ ESTIMATEUB $(W', \{\hat{\mathcal{P}}^{w_j}\}, \Psi)$;
9 Insert $\langle W', \overline{\mathcal{I}}(W'), \texttt{bounded} \rangle$ into $\mathcal{H}$;
10 **if** $W^* \neq \emptyset$ **then**
11 $\mathcal{I}(W^*) \leftarrow$ CALCIMPACT $(W^*, \{\hat{\mathcal{P}}^{w_j}\}, \Psi)$;
12 Insert $\langle W^*, \mathcal{I}(W^*), \texttt{computed} \rangle$ into $\mathcal{H}$;
13 **else if** $state = \texttt{computed}$ **then**
14 **if** $\mathcal{I}(W^*) > \sum_{i=1}^{n} \psi_i \log \phi_i^S$ **then return** W^* ;
15 **else return** $\emptyset$;

2) If the state is `computed`, it can safely ensure that impact $\mathcal{I}(W^*)$ of W^* is larger than upper bounds of any other worker sets. Then, the algorithm examines whether W^* can reduce the KL divergence of Φ^S from Ψ by comparing $\mathcal{I}(W^*)$ and $\sum_{i=1}^{n} \psi_i \log \phi_i^S$. If the former is larger, it returns W^* as the selected worker set. Otherwise, it gives up W^* and waits for the next time point (lines 14-15).

Upper Bound Estimation. Obviously, the success of Algorithm 2 relies on the performance of bound estimation.

Lemma 3. The impact of any worker set W is bounded by

$$\mathcal{I}(W) \leq \log \frac{\sum_{w_j \in W} \mu_j}{\sum_{w_j \in W} \nu_j}, \tag{14}$$

where $\mu_j = \sum_{i=1}^{n} \psi_i \cdot \hat{p}_i^j$ and $\nu_j = \sum_{i=1}^{n} \hat{p}_i^j$.

Proof 4. According to the definition of $\mathcal{I}(W)$ and Equation (9), we have

$$\mathcal{I}(W) = \sum_{i=1}^{n} \psi_i \log \frac{\sum_{w_j \in W} \hat{p}_i^j}{\sum_{w_j \in W} \sum_{l=1}^{n} \hat{p}_i^j}.$$

Then, as $\sum_i \psi_i = 1$, we can apply the Jensen's inequality to the above equation, i.e.,

$$\begin{aligned} \mathcal{I}(W) &\leq \log \frac{\sum_{w_j \in W} \sum_{i=1}^{n} \psi_i \cdot \hat{p}_i^j}{\sum_{w_j \in W} \sum_{l=1}^{n} \hat{p}_i^j} \\ &\leq \log \frac{\sum_{w_j \in W} \mu_j}{\sum_{w_j \in W} \nu_j}. \end{aligned}$$

Hence, we prove the lemma. □

Lemma 3 allows us to estimate the upper bound of any candidate worker set W^* as follows. For each active worker w_j, we precompute its $\mu_j = \sum_{i=1}^{n} \psi_i \cdot \hat{p}_i^j$ and $\nu_j = \sum_{i=1}^{n} \hat{p}_i^j$. Then, we sort μ_j for all w_j's in *descending* order and use $\mu^{(l)}$ to denote the one in the l-th position of the sorted list. Similarly, we sort all ν_j's in *ascending* order and use $\nu^{(l)}$ to denote the one in the l-th position. Recall that $\overline{\mathcal{I}}(W^*)$ is used to denote the upper bound of any *super set* of W^*. Then, we have the following estimate of $\overline{\mathcal{I}}(W^*)$.

$$\overline{\mathcal{I}}(W^*) = \max_{L=0\ldots(|W|-|W^*|)} \log \frac{\sum_{w_j \in W^*} \mu_j + \sum_{l=0}^{L} \mu^{(l)}}{\sum_{w_j \in W^*} \nu_j + \sum_{l=0}^{L} \nu^{(l)}}. \tag{15}$$

Lemma 4. Given active worker set W, $\overline{\mathcal{I}}(W^*)$ in Equation (15) is an upper bound for any super set of W^*.

Note that we can also apply some incremental computation and pruning techniques to compute Equation (15). We omit the details due to the space constraint.

Algorithm 3: LOCALSEARCHSELECT ($W, \{\hat{\mathcal{P}}^{w_j}\}, \Psi, S$)

Input: W: workers; $\{\hat{\mathcal{P}}^{w_j}\}$: estimated worker models; Ψ: expected distribution; S: collected entities
Output: W^*: a selected worker set

1 Initialize a worker set $W^* = \emptyset, \mathcal{I}(W^*) = 0$;
2 **while** *true* **do**
3 **for** *each* $w \in W - W^*$ **do**
4 $\mathcal{I} \leftarrow$ CALCIMPACT $(W^* \cup \{w\}, \{\hat{\mathcal{P}}^{w_j}\}, \Psi)$;
5 **if** $\mathcal{I} > \mathcal{I}(W^*)/(1+\varepsilon)$ **then**
6 $W^* \leftarrow W^* \cup \{w\}, \mathcal{I}(W^*) \leftarrow \mathcal{I}$;
7 **break**;
8 **if** *having insertion* **then continue** ;
9 **for** *each* $w \in W^*$ **do**
10 $\mathcal{I} \leftarrow$ CALCIMPACT $(W^* \setminus \{w\}, \{\hat{\mathcal{P}}^{w_j}\}, \Psi)$;
11 **if** $\mathcal{I} > \mathcal{I}(W^*)/(1+\varepsilon)$ **then**
12 $W^* \leftarrow W^* \setminus \{w\}, \mathcal{I}(W^*) \leftarrow \mathcal{I}$;
13 **break**;
14 **if** *having deletion* **then continue** ;
15 **for** *each* $w \in W - W^* \wedge w' \in W^*$ **do**
16 $\mathcal{I} \leftarrow$ CALCIMPACT $(W^* \cup \{w\} \setminus \{w'\}, \{\hat{\mathcal{P}}^{w_j}\}, \Psi)$;
17 **if** $\mathcal{I} > \mathcal{I}(W^*)/(1+\varepsilon)$ **then**
18 $W^* \leftarrow W^* \cup \{w\} \setminus \{w'\}$;
19 **break**;
20 **if** *having swap* **then continue** ;
21 **if** $\mathcal{I}(W^*) > \sum_{i=1}^{n} \psi_i \log \phi_i^S$ **then return** W^* ;
22 **else return** $\emptyset$;

4.4 A Local Search Algorithm for Worker Selection

Due to the hardness of the problem (see Theorem 2), to achieve better performance than the exact algorithm, we also develop a *local search* based algorithm to approximately solve the optimal worker selection problem.

Initially, we arbitrarily select a subset W^* from W. Then, we define three operations based on *local search scheme*: 1) ***add***: add a new worker $p \in (W \setminus W^*)$ to W^*; 2) ***remove***: remove a worker from W^*, if $|W^*| > 1$; 3) ***swap***: swap a worker in W^* with another one in $(W \setminus W^*)$. We repeatedly perform one of the three operations to improve the impact, and the search process terminates when no new operation can produce a better result. One technical issue is that the improvement of each operation may be small, leading to

TABLE 3
Statistics of Datasets

Datasets	attribute A	$\|\Omega\|$	# of workers	# of answers
movie	decade	7	74	5,000
car	body style	15	91	1,975

TABLE 4
Uneven Contributions across Workers

# answers / worker		$[0, 30)$	$[30, 60)$	$[60, 90)$	≥ 90
# workers	movie	50	10	5	9
	car	76	6	5	4

too many operations which prevent the algorithm from finishing in polynomial time. To avoid this, we perform one of the three operations when the impact is improved by a factor of $1 + \varepsilon$, where ε is an arbitrary small constant.

Algorithm 3 shows the pseudo-code. It starts from an empty worker set $W^* = \emptyset$, and examines the improvement on impact of each add / remove / swap operation. Specifically, if the impact $\mathcal{I}$ after an operation is improved by a factor of $1 + \varepsilon$, i.e., $\mathcal{I} > \mathcal{I}(W^*)/(1 + \varepsilon)$, it performs the operation and then examines the possibility of subsequent operations. The algorithm terminates when none of the three operations can improve the impact by a factor of $1 + \varepsilon$. Let $\mathcal{I}_{\max}$ be the maximum possible impact over all possible subsets, and then the number of operations is bounded by $\log_{1+\varepsilon} \mathcal{I}_{\max}$, which is polynomial in the input size.

5 Experiments

This section evaluates the performance of our approach. First, we report some observations on worker behaviors for entity collection. Then, we evaluate the methods for estimating the probabilistic model of each worker. Finally, we compare different strategies for worker selection.

5.1 Experiment Setup

Datasets. We conduct experiments on the well-known crowdsourcing platform Amazon Mechanical Turk (AMT) and evaluate the approaches on two real datasets collected from the workers on AMT. 1) The movie dataset contains a set of *movie* entities: each worker is asked to submit movies she knows, together with an attribute decade indicating the time a movie firstly publishes. The domain of this attribute contains 7 distinct values, ranging from 1950s to 2010s. 2) The car dataset contains a set of *car* entities from the workers, together with an attribute body style which has 15 distinct values, such as sedans, suvs, etc.

Specifically, these two datasets are collected from the workers in the following way. We publish a set of human intelligent tasks (HITs) on AMT, where each HIT asks a worker to submit 10 entities together with their attribute values (e.g., 10 cars with their body style). To ensure that all approaches are compared on the same set of workers, we ask workers to submit as many entities as they can. In particular, each record in a dataset consists of entity name, attribute value, worker ID and submission time. For example, a record in the movie dataset is { Black Swan, 2010s, A13FVM2C914A3H, 2016-04-15 19:54:22 }. Then, based on these records, we can run different approaches for worker selection and compare their performance. Table 3 shows statistics of the datasets and answers collected from AMT.

Compared approaches. We implement our approach and compare with baseline approaches. Note that we compare both worker model estimation and online worker selection.

For worker model estimation, we compare five methods: 1) Empirical applies the empirical estimator with smoothing. 2) GoodTuring exploits the Good-Turing estimator. 3) JelinekMercer utilizes the Jelinek-Mercer technique in [16]. 4) AbsoluteDisc uses the *absolute discounting* technique in [28]. 5) Hybrid is our hybrid approach combining empirical and Good-Turing estimators.

For adaptive worker selection, we implement the best-effort and the local-search algorithms proposed in Section 4, and compare with alternative worker selection methods. 1) NoSelect does not perform worker selection, that is, it accepts all the active workers and includes their entities. 2) RandomSelect applies a random selection strategy that randomly picks a subset of active workers. Note that we run RandomSelect multiple times and use the average as its result. 3) BestEffort utilizes our best effort algorithm that selects a subset of workers having minimum KL divergence from the expected distribution in the query. 4) LocalSearch employs the local search algorithm that approximately solves the worker selection problem.

Evaluation metrics. We evaluate approaches on both *effectiveness* and *efficiency*. Effectiveness is measured by the actual KL divergence of the collected entities from the expected distribution in a query, while efficiency is measured by the elapsed time of worker selection.

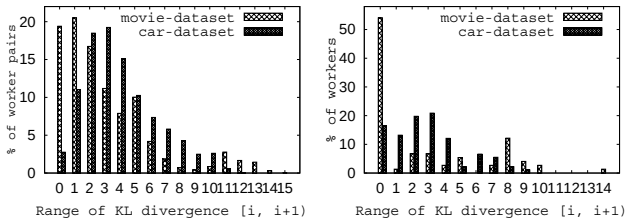
Experiment settings. To evaluate performance on effectiveness, we sort records in a dataset in ascending order of the submission time and access the records one by one. Then, given a query, we apply a worker selection approach to run the framework in Algorithm 1. In particular, we set the granularity of the "time point" in Algorithm 1 as 5 hours. Finally, when enough entities are collected, we measure the KL divergence of the collected entities from the expected distribution. On the other hand, to evaluate performance on efficiency, we implement all the programs in JAVA and run all the experiments on a Mac machine with an Intel Core i5 2.8 GHz processor and 8 GB memory.

5.2 Observations of Crowdsourced Entity Collection

We first investigate worker behaviors on crowdsourced entity collection by analyzing the collected answers on each dataset. Table 4 shows the *uneven contribution* of crowdsourcing workers. For example, on the movie dataset, 68% of workers provide less than 30 entities, while a few "streakers" provide more than 90 entities. In particular, the most zealous worker contributes 1250 entities, which are much more than that of other workers.

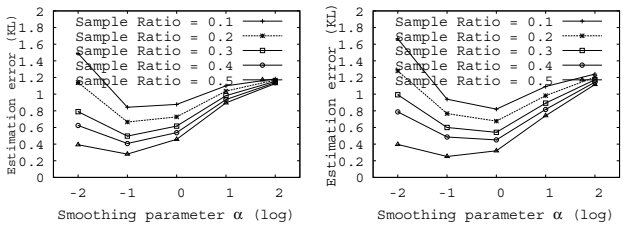
Next, we examine *diverse bias* of crowdsourcing workers, by using all entities of a worker to compute entity distribution of the worker. We first consider each pair of workers and compute KL divergence between distributions of their entities [6]. Figure 4(a) shows the percentage of worker pairs in various ranges of KL divergence (e.g., $[0, 1)$, $[1, 2)$, ...). It is obvious to see that most of the worker pairs have large values of KL divergence, e.g., more than 80% of worker

6. As KL divergence is not asymmetric, we compute KL divergence on both sides, and then use the average.



(a) Pairwise KL (b) KL w.r.t query

Fig. 4. Diverse entity distributions across workers.



(a) movie dataset (b) car dataset

Fig. 5. Effect of smoothing in empirical estimation.

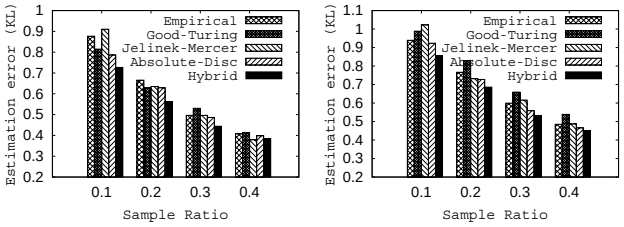
pairs on the movie dataset and more than 95% on the car dataset having KL divergence larger than 1. This validates our claim in Section 2 that different workers have quite diverse biases on the entities they know.

Moreover, we also examine the deviation of each individual worker's entity distribution from an expected distribution. On each dataset, given an expected query distribution, we compute the KL divergence according to Equation (2) for each worker, and then plot percentages of workers in various ranges of KL divergence in Figure 4(b). As seen in the figure, almost 50% of workers on the movie dataset and more than 80% of workers on the car dataset have KL divergence larger than 1 from the query. This experimental result shows that it is not effective to only rely on individual workers for entity collection. Thus, to achieve better performance, we need to select a set of workers and aggregate their entities, which validates the necessity of the worker selection problem introduced in this paper.

5.3 Evaluation on Worker Model Estimation

We evaluate the approaches to estimating worker probabilistic model in this section. For each worker, we select some percentage (e.g., 20%) of her entities as a "sample set" that feeds an estimator to estimate $\hat{\mathcal{P}}^w$. Then, we measure the KL divergence of the estimated distribution from the actual distribution of the worker, i.e., $D_{\mathrm{KL}}(\mathcal{P}^w||\hat{\mathcal{P}}^w)$. Next, we average values of KL divergence of all workers as *estimation error* to measure the performance of an estimator.

We first examine the empirical estimator mentioned in Section 3.1. In particular, we vary the smoothing parameter α in Equation (3) as 0.01, 0.1, 1, 10 and 100, and report the result in Figure 5. As seen in the figure, with the increase of smoothing parameter α, the estimation error first decreases and then increases. This is mainly attributed to the following reasons. The Laplace smoothing can improve the performance of probability estimation, because it adds a *pseudocount* α to empirical relative frequencies. This pseudocount can on the one hand estimate probabilities for the



(a) movie dataset (b) car dataset

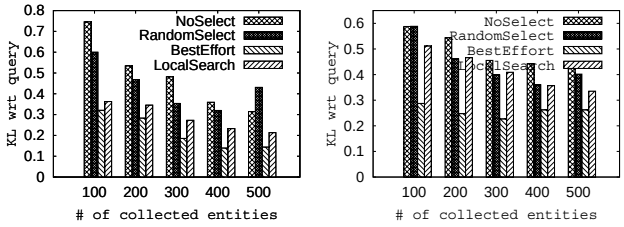
Fig. 6. Evaluation on worker model estimation.

attribute values not seen in the sample, and on the other hand smooth relative frequencies of those frequently occur in the sample. However, when α is too large, e.g., $\alpha = 10$ or 100 in Figure 5, this pseudocount will "dominate" relative frequencies, which fails to capture relative frequencies and thus brings damages to the estimation. In our experiment, we observe that $\alpha = 0.1$ performs well in most of cases. Thus, we set $\alpha = 0.1$ as a default smoothing parameter for the empirical estimator in the remainder of this section.

Next, we compare different approaches for worker model estimation. Note that parameters of approaches (e.g., discounting parameter in AbsoluteDisc) are tuned using held-out estimation [4]. The experimental result is reported in Figure 6. We have the following observations.

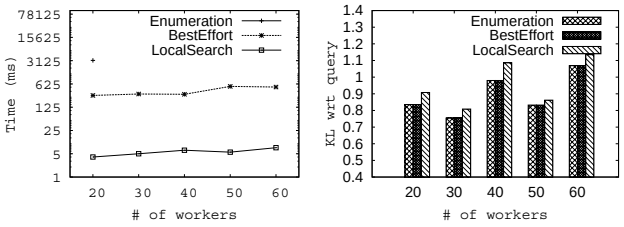
First, on the movie dataset, GoodTuring performs better than Empirical for small sample ratios. For example, GoodTuring can achieve about 7% improvement on estimation error when sample ratio is 0.1. However, when the sample ratio is large, e.g, 0.3 or 0.4, GoodTuring produces worse estimates. This is because GoodTuring is usually good for estimating attribute values with low frequencies in the sample [30]. Specifically, when sample ratio is small, many attribute values only have low frequencies or even do not occur in the sample. In this case, Empirical only adds a uniform pseudocount α for smoothing, while GoodTuring considers "frequency of frequencies" by applying Equation (5) which can better capture the underlying probability distribution than a uniform pseudocount. However, when sample ratio is large, many attribute values occur frequently in the sample. According to Equation (6), we can see that GoodTuring may not perform well in this case, because N_{r+1} is very likely to be zero. In addition, on the car dataset, GoodTuring performs worse than Empirical. This is because the dataset is relatively sparse, i.e., only having 1975 entities for 91 workers (the movie dataset has 5000 entities for 74 workers), and thus may increase the error of estimating $\mathbb{E}_N[N_r]$ in GoodTuring.

Second, the Hybrid estimator combining GoodTuring and Empirical achieves the lowest estimation error. For example, when sample ratio is 0.1, it achieves 17% improvement to Empirical and 14% improvement to GoodTuring on the movie and car datasets respectively. It also performs better than JelinekMercer and AbsoluteDisc, which are "common practical estimators" [30]. This result validates the practice that GoodTuring is often used in conjunction with Empirical [30]: for attribute values with low or zero frequencies, GoodTuring can capture the underlying probability distribution, while for those with high frequencies, Empirical can avoid the zero N_{r+1} problem.



(a) movie dataset (b) car dataset

Fig. 7. Effectiveness of worker selection methods.



(a) Efficiency (b) Approximation

Fig. 8. Evaluation of worker selection methods in simulation.

5.4 Evaluation on Adaptive Worker Selection

This section compares different approaches to worker selection on both effectiveness and efficiency.

Evaluation on Effectiveness. On each dataset, we run the framework in Algorithm 1 and apply different approaches for function SELECTWORKERS. We vary the number of collection k from 100 to 500, and report the KL divergence of the collected entities from the query.

Figure 7 shows the experimental result. First, approach `NoSelect` produces the largest KL divergence from the query on both datasets. This validates our claim in the Introduction that distribution of entities will become *unpredictable* if no effective strategy is utilized to control the collection process. Second, the random strategy `RandomSelect` only has limited improvement compared with `NoSelect`, and sometimes it may not be stable, e.g., producing worse result on the `movie` dataset when collecting 500 entities. This result justifies the motivation of our framework that is aware of current entity distribution and selects workers to reduce the KL divergence as much as possible. Third, our proposed exact solution `BestEffort` achieves the best performance in all the cases. For example, it achieves about 50% improvement on both datasets, which can significantly reduce the KL divergence of the collected entities from the query. The improvement is mainly attributed to our worker selection objective that minimizes the estimated KL divergence (or maximizes the impact) of a worker set. Moreover, coupled with the effective worker model estimator, `BestEffort` can finally output a proper set of entities that best approximates the expected distribution. Fourth, our approximate solution `LocalSearch`, although performing worse than the the exact solution `BestEffort`, also achieves good performance and outperforms the baselines, `NoSelect` and `RandomSelect`, with a large margin.

Evaluation on Efficiency. Next, we evaluate the efficiency of the proposed worker selection algorithms in Section 4. To this end, we build a simulation environment with two parameters, number of workers (i.e., m) and size of attribute value domain Ω (i.e., n). Considering a specified settings of m and n, we generate m workers by randomly assigning entity distributions of size n to them. Then, given an expected distribution, we run different worker selection algorithms and report the time. By default, we set $m = 20$ and $n = 20$. We compare three alternative algorithms: 1) `Enumeration` is a brute-force algorithm that enumerates all possible sets of workers, computes KL divergence for each of them, and selects the one with the minimum divergence; 2) `BestEffort` is our best-effort algorithm (Algorithm 2); 3) `LocalSearch` is the approximate algorithm (Algorithm 3).

We first consider the effect of number m of workers by varying the value of m. Figure 8 shows the experimental results. As seen in Figure 8(a), `Enumeration` is inefficient and it cannot select workers for the number of workers larger than 20 (we wait for 30 minutes and still cannot get the result of `Enumeration`). This result is not surprising due to the hardness of the worker selection problem as stated in Theorem 2. Our exact solution `BestEffort` is efficient and can select workers in hundreds of milliseconds. For example, when the number of workers is 60, it can produce the result within about 500 milliseconds. This is mainly due to our best effort framework and upper-bound estimation techniques, which can preferentially compute exact impact for the worker sets with larger upper bounds, so as to prune the insignificant worker sets. Our approximate solution `LocalSearch` achieves the best efficiency, and it can select workers within 10 ms, which is faster than `BestEffort` by one order of magnitude. Moreover, as seen in Figure 8(b), `LocalSearch` also has good approximate performance. This result shows that `LocalSearch` is also good enough to be used for worker selection, if we have real-time requirements on worker selection.

We also evaluate the efficiency against the size of attribute value domain Ω (i.e., n), and find that n only slightly affects the efficiency. This is because n only affects the functions of computing impact or KL divergence, which is not the bottleneck of the computation. Due to the space constraint, we omit this result in this paper.

Summarization. Finally, we summarize the main conclusions of our experiments as follows. 1) Our experimental result supports our claim about worker behaviors on crowdsourced entity collection, i.e., the two properties, *diverse bias* and *uneven contribution*. 2) Our hybrid estimation method that combines `Empirical` and `GoodTuring` achieves the best performance on estimating worker models. 3) Our exact best-effort algorithm `BestEffort` can significantly reduce the KL divergence of the collected entities from the expected distribution in query and is much more efficient on time than the brute-force algorithm. 4) Our approximate algorithm `LocalSearch` selects workers within 10 ms and thus can fulfill real-time worker selection requirements. Also, it has good approximation performance and overall effectiveness in our experiments.

6 RELATED WORK

The studies most related to our work are the recently proposed crowdsourced entity collection [38], [6], [36], [35]. Park and Widom developed a general framework CrowdFill that shows a partially filled table and asks the crowd to

contribute new entities, fill empty cells, and up-vote / down-vote existing entities [35]. However, CrowdFill neither pays attention to estimating worker behaviors on entity collection, nor considers statistical properties of the collected entities. Trushkowsky et al. is the first work to study worker behaviors on collecting entities [38] and utilized statistical approaches for reasoning completeness of the collected data. Chung et al. extended the estimation techniques to support aggregate queries, such as SUM, AVG, MAX / MIN, etc., by analyzing both coverage of the collected entities and the correlation between attributes and entities [6]. Rekatsinas et al. extended the underlying data model to a structured domain with hierarchical structure (e.g., restaurant with location and cuisine), and aimed to maximize collection coverage under a monetary budget. The key difference of our work from these previous studies is three-fold. First, the problem settings are different: we focus on the underlying distribution of the collected entities, which is often indispensable in data collection but ignored by previous studies. Second, the estimation tasks are different: the existing approaches focused on estimating overall statistics of the *entire entity set* collected from the crowd, such as the coverage and aggregate results. In contrast, we aim at estimating the underlying probability distribution of each *individual worker*. The previous estimation techniques cannot be adapted to solve our more fine-grained problem. Third, the optimization problems are different: as far as we know, we are the first to study the problem of selecting workers to minimize the difference from expected distributions.

Recently, many studies in the database community have aimed to leverage crowdsourcing to build database systems, such as CrowdDB [12], Qurk [25], Deco [34] and CDAS [22], [11], and develop various operators, such as filter [32], [22], join [26], [39], [10], sort / top-k [14], [7], graph search [33], [15], and counting [24]. To achieve effective crowdsourcing performance, existing studies investigated *quality control* strategies in crowdsourcing. Most of the strategies applied a redundancy-based approach which assigns a crowdsourcing task to multiple workers and aggregates worker answers by using weighted majority voting. Some approaches [22], [32] leveraged a small amount of crowdsourcing tasks with ground truth to estimate workers reliability as aggregation weight, while other approaches [37], [18] simultaneously estimated worker weights and predicted aggregated results using an Expectation-Maximization (EM) strategy. However, most of crowdsourcing-powered operators as well as quality control techniques only consider crowdsourced data evaluation, which asks the crowd to evaluate the existing data according to some criteria, such as filtering useless data, and joining data from various sources. In contrast, this paper focuses on studying crowdsourced entity collection which asks the crowd to collect data instead of evaluating existing data. The key difference and the main challenge of crowdsourced entity collection is the "open world" nature of crowdsourcing which may returns unbounded amount of answers. To address this challenge, we have devised novel estimation and quality control techniques for adaptive worker selection.

Crowdsourcing has many successful applications in different areas. Solyent [3] is a word processor that employs the Find-Fix-Verify interaction method. gMission is a general platform for supporting spatial crowdsourcing [5] Adrenaline [2] is a crowd-powered camera that supports real-time crowdsourcing by using an interaction method called rapid refinement. Our work also has large potentials to be utilized in the so-called data curation applications [27], such as knowledge base completion [19], domain-aware entity (e.g., points-of-interest in twitter) collection.

Probability estimation discussed in Section 3 is extensively studied in the area of statistical learning. Good-Turing is a well-known approach to this end [13], [29], [30]. Gale et. al [13] discussed the derivation of Good-Turing estimates and applied this method to the *smoothing* problem in natural language processing. The studies [29], [30] theoretically proved the effectiveness of Good-Turing and suggested to combine Good-Turing and empirical estimates. In this paper, we utilize the Good-Turing method to a new problem, i.e., estimating entity distribution of workers, and conduct extensive experiments to compare it with other estimation techniques. Recently, some recommendation studies [1], [21], [31] also considered to model and estimate user behaviors (e.g., reputation). Compared with these studies, our work focuses to estimate worker behaviors on entity collection, which are not well studied in the existing works.

7 Conclusion

In this paper, we studied the problem of distribution-aware crowdsourced entity collection that controls the crowdsourcing process to collect entities following an expected distribution from the crowd. We introduced an adaptive worker selection framework to estimate worker's distribution based on her historical entity set and select a subset of workers that minimizes the KL divergence from the expected distribution. We devised effective statistical estimation approaches to estimating worker's distribution with low estimation error. We proved that the problem of worker selection is NP-complete and developed a best-effort algorithm to find exact solution and an approximate local search algorithm for instant worker selection. We deployed the proposed approach on AMT and the experimental results on two real datasets show that the approach achieves superiority on both effectiveness and efficiency.

Acknowledgement

This work was partly supported by the National Natural Science Foundation of China (NSFC) under Grant No. 61602488, No. 61632016, No. 61502503, and No. 61602087, the 973 Program of China (Project No. 2012CB316205), the Start-up Research Grant of Renmin University of China (Project No. 16XNLF02), and Tencent.

References

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.*, 17(6):734–749, 2005.

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

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

[4] S. F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–393, 1999.

[5] Z. Chen, R. Fu, Z. Zhao, Z. Liu, L. Xia, L. Chen, P. Cheng, C. C. Cao, Y. Tong, and C. J. Zhang. gmission: A general spatial crowdsourcing platform. *PVLDB*, 7(13):1629–1632, 2014.

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

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

[8] 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.

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

[10] 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.

[11] 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.

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

[13] W. A. Gale and G. Sampson. Good-turing frequency estimation without tears. *Journal of Quantitative Linguistics*, 2(3):217–237, 1995.

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

[15] X. Huang, H. Cheng, R. Li, L. Qin, and J. X. Yu. Top-k structural diversity search in large networks. *VLDB J.*, 24(3):319–343, 2015.

[16] F. Jelinek and R. L. Mercer. Probability distribution estimation from sparse data. *IBM Technical Disclosure Bulletin*, 1985.

[17] S. Kajimura, Y. Baba, H. Kajino, and H. Kashima. Quality control for crowdsourced POI collection. In *PAKDD 2015*, pages 255–267, 2015.

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

[19] S. K. Kondreddi, P. Triantafillou, and G. Weikum. Combining information extraction and human computing for crowdsourced knowledge acquisition. In *ICDE 2014*, pages 988–999, 2014.

[20] S. Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[21] B. Li, R. Li, I. King, M. R. Lyu, and J. X. Yu. A topic-biased user reputation model in rating systems. *Knowl. Inf. Syst.*, 44(3):581–607, 2015.

[22] 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.

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

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

[25] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller. Demonstration of qurk: a query processor for humanoperators. In *SIGMOD 2011*, pages 1315–1318, 2011.

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

[27] R. J. Miller. Big data curation. In *COMAD 2014*, page 4, 2014.

[28] H. Ney, U. Essen, and R. Kneser. On structuring probabilistic dependences in stochastic language modelling. *Computer Speech & Language*, 8(1):1–38, 1994.

[29] A. Orlitsky, N. P. Santhanam, and J. Zhang. Always good turing: Asymptotically optimal probability estimation. In *FOCS 2003*, pages 179–188, 2003.

[30] A. Orlitsky and A. T. Suresh. Competitive distribution estimation: Why is good-turing good. In *NIPS 2015*, pages 2143–2151, 2015.

[31] W. Pan, S. Xia, Z. Liu, X. Peng, and Z. Ming. Mixed factorization for collaborative recommendation with heterogeneous explicit feedbacks. *Inf. Sci.*, 332:84–93, 2016.

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

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

[34] H. Park, R. Pang, A. G. Parameswaran, H. Garcia-Molina, N. Polyzotis, and J. Widom. Deco: A system for declarative crowdsourcing. *PVLDB*, 5(12):1990–1993, 2012.

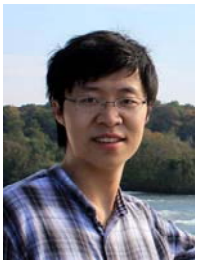
[35] H. Park and J. Widom. Crowdfill: collecting structured data from the crowd. In *SIGMOD 2014*, pages 577–588, 2014.

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

[37] V. S. Sheng, F. J. Provost, and P. G. Ipeirotis. Get another label? improving data quality and data mining using multiple, noisy labelers. In *SIGKDD 2008*, pages 614–622, 2008.

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

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



Ju Fan received the BEng degree in computer science from Beijing University of Technology, China in 2007 and the PhD degree in computer science from Tsinghua University, China in 2012. He worked as a research fellow in the School of Computing, National University of Singapore (NUS) from 2012 to 2015. He is currently an associate professor at Renmin University of China. His research interests include crowdsourcing data management, big data analytics, and database usability.

Zhewei Wei received the B.Sc. in Mathematics from Peking University, China in 2007 and the PhD degree in computer science and engineering from The Hong Kong University of Science and Technology, Hong Kong in 2012. He worked as a Postdoc in the Centre for Massive Data Algorithmics (MADALGO), Aarhus University, Denmark from 2012 to 2014. He is currently an associate professor at Renmin University of China. His research interests include massive data algorithms and database management.

Dongxiang Zhang is a Professor in the School of Computer Science and Engineering, University of Electronic Science and Technology of China. He received his B.Sc. degree from Fudan University, China in 2006 and the PhD degree from National University of Singapore in 2012. He worked as a research fellow at the NeXT center in Singapore from 2012 to 2014 and was promoted as a senior research fellow in 2015. His research interests include spatial databases, cloud computing and big data analytics.

Jingru Yang obtained the B.S. degree from Beijing Forestry University, Beijing, China, in 2016. She is currently a PhD student at the School of Information and the Key Lab of Data Engineering and Knowledge Engineering, Renmin University of China. Her current research interests include crowdsourced data collection and data visualization.

Xiaoyong Du obtained the B.S. degree from Hangzhou University, China, in 1983, the M.E. degree from Renmin University of China, in 1988, and the Ph.D. degree from Nagoya Institute of Technology, Japan, in 1997. He is currently a Professor with the School of Information, Renmin University of China. He has authored or coauthored more than 100 papers. His current research interests include databases and intelligent information retrieval.