DOCS: A Domain-Aware Crowdsourcing System Using Knowledge Bases

Yudian Zheng [†], Guoliang Li [#*], Reynold Cheng [†]
[†] Department of Computer Science, The University of Hong Kong
[#] Department of Computer Science, Tsinghua University
ydzheng2@cs.hku.hk, liguoliang@tsinghua.edu.cn, ckcheng@cs.hku.hk

ABSTRACT

Crowdsourcing is a new computing paradigm that harnesses human effort to solve computer-hard problems, such as entity resolution and photo tagging. The crowd (or workers) have diverse qualities and it is important to effectively model a worker's quality. Most of existing worker models assume that workers have the same quality on different tasks. In practice, however, tasks belong to a variety of diverse domains, and workers have different qualities on different domains. For example, a worker who is a basketball fan should have better quality for the task of labeling a photo related to '*Stephen Curry*' than the one related to '*Leonardo DiCaprio*'. In this paper, we study how to leverage domain knowledge to accurately model a worker's quality. We examine using *knowledge base* (KB), e.g., Wikipedia and Freebase, to detect the domains of tasks and workers. We develop *Domain Vector Estimation*, which analyzes the domains of a task with respect to the KB. We also study *Truth Inference*, which utilizes the domain-sensitive worker model to accurately infer the true answer of a task. We design an *Online Task Assignment* algorithm, which judiciously and efficiently assigns tasks to appropriate workers. To implement these solutions, we have built DOCS, a system deployed on the Amazon Mechanical Turk. Experiments show that DOCS performs much better than the state-of-the-art approaches.

1. INTRODUCTION

To tackle complex tasks that are hard for computers (e.g., entity resolution [43, 45] and sentiment analysis [54, 29]), many crowdsourcing platforms (e.g., Amazon Mechanical Turk (AMT) [3] and CrowdFlower [13]) have been recently deployed. These platforms allow tasks to be performed by a huge number of Internet users (or *workers*) with different backgrounds. In AMT [3], for instance, there are over 500K workers originated from 190 countries [1]. The increase in the importance of crowdsourcing has attracted a lot of research attention [27, 19, 16, 53, 15, 54, 23, 18, 8, 50, 24].

*Guoliang Li is the Corresponding Author.

Proceedings of the VLDB Endowment, Vol. 10, No. 4

However, workers may yield low quality and a core problem in crowdsourcing is to infer high-quality results from the workers' answers. Different workers may have diverse qualities, and it is important to accurately model a worker's quality. An effective worker model can benefit many important problems in crowdsourcing and we examine three crucial aspects addressed in existing works [19, 16, 53, 54, 15, 18, 30, 8] that help to infer high-quality results:

• *Worker Model*: How to represent the quality of a worker that effectively reflects her skills? Existing works simply treat it as a real value [19, 16, 53] or a matrix [15, 54].

• *Truth Inference*: How to obtain the true answer (called *truth*) of a task? To improve the quality, a task may be performed by one or more workers, and thus an important issue is to infer the truth through aggregating workers' answers [18, 30, 16, 15].

• *Task Assignment*: How to assign a task to appropriate workers? As pointed out by [54, 8, 18], this is often done based on worker model, which reflects her performance statistics shown in her previous tasks. Note that the assignment latency is crucial and *online* task assignment is required to achieve instant assignment.

A common drawback of existing solutions is that they often overlook the worker's ability in different aspects (or *domains*). As a matter of fact, workers have a variety of expertise, skills, and cultural backgrounds. Let us consider two workers (A, an *NBA* fan, and B, a frequent moviegoer) and two tasks t_1 and t_2 (which ask workers to select labels of two photos about *Stephen Curry* and *Leonardo DiCaprio*, respectively). Intuitively, A should do better than B in the *sports* domain, while B is a better candidate than A in doing tasks related to *films*. Thus, t_1 and t_2 should be assigned to A and B respectively. However, existing works often neglect the domain information (e.g., the qualities of A and B are modeled as the same values for different tasks [19, 16, 53]).

The issues of incorporating domain knowledge in the crowdsourcing process have only been recently studied [18, 30], where each worker has diverse qualities on different domains. These solutions, while promising, still have room for improvement:

• *Worker Model*: [18] examines the issues of inferring domains of workers and tasks. The solution relies on the text descriptions of tasks – tasks with large text similarity have a higher chance to be classified into the same domain. However, this solution can result in wrong domain classification. For example, the two tasks "*Is Stephen Curry a PF?*" and "*Has Golden State Warriors ever won championships?*" may not be similar (e.g., in terms of Jaccard similarity), yet they are in the *sports* domain. On the other hand, tasks "*Compare the height of Stephen Curry and Kobe Bryant.*" and "*Compare the height of Mount Everest and K2.*" may have a high text similarity, but they are in different domains (i.e., *sports* and *mountains*, respectively). In [30], machine-learning techniques are used to compute *latent domains* of tasks. However, due to the lack

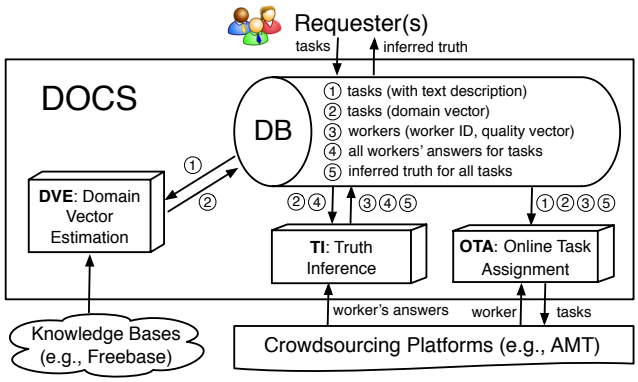

Figure 1: The Architecture of DOCS.

of semantics, these *latent domains* are hard to interpret, making it difficult to profile and understand a worker's ability.

• *Truth Inference*: In [18, 30], the problem of using domain information to infer truth has been studied. Typically they exploit a worker's diverse qualities on different domains, and then for a task, it will trust a worker's answer if the worker has high qualities on the domains in that task. However, the workers' qualities are either inaccurately estimated [30], or incorrectly leveraged to compute each task's truth [18]. For example, [30] estimates each task's *latent domains* and each worker's quality for those *latent domains* together, thus the estimation of worker's quality is highly affected by the inaccurate estimation of task's domains; [18] uses the weighted majority voting to infer each task's truth, which is easy to be misled by the answers given by multiple low-quality workers.

• *Task Assignment*: The only work that uses domain information in task assignment is [18]. However, it adopts a fairly simple task assignment method, in which each task is assigned to the *same* number of workers, and the difficulty level of a task is not considered. Moreover, it assigns tasks to a worker such that the worker has the highest qualities to accomplish, which omits the fact the assigned tasks may have already obtained confident and consistent answers.

Hence, there is a need of investigating how to make the best use of domain information in the crowdsourcing process. Our goal is to study an effective method of utilizing domain information to enhance the effectiveness of the three steps above. The main idea is to consult an existing knowledge base (or *KB*), such as Wikipedia [47] and Freebase [20] for obtaining domain information. These KBs are often associated with a large number of categories/topics information, organized in a systematic and hierarchical manner. For example, Freebase [20] contains over $57M$ concepts, encoded by $3G$ facts. We have developed a DOmain-Aware Crowdsourcing System, called DOCS, which taps into this large pool of information, learning the domains of workers and tasks explicitly.

Figure 1 shows the architecture of DOCS, which contains three main modules: **Domain Vector Estimation (DVE)**, **Truth Inference (TI)** and **Online Task Assignment (OTA)**. A requester (who publishes tasks) can specify a set of tasks (with text descriptions) and a budget in DOCS. Then DOCS interacts with knowledge bases and crowdsourcing platforms, respectively. Finally after consuming the budget, the inferred truth for all tasks are returned to the requester. Next, we show how the three modules in DOCS work upon receiving a requester's tasks.

• **DVE**. This module is responsible for estimating the related domains of each task, based on the domain information in a KB. Specifically, an "entity-linking" algorithm [39] can be used, which extracts entities from the text description of each task. A *domain vector* is then computed for these entities, in order to capture how likely a task belongs to each domain mentioned in a KB.

After computing each task's domain vector, the tasks are published to crowdsourcing platforms (e.g., AMT [3]). By interacting with the crowd workers, in general, DOCS needs to handle two types of requests from workers: (1) a worker accomplishes tasks and submits answers; (2) a worker comes and requests tasks.

• **TI**. When a worker accomplishes tasks and submits answers, the module first stores the worker's answers into database and then infers each task's truth and each worker's model based on two principles: (1) a worker's answer is trusted, if she is a domain expert on her submitted tasks; and (2) a worker is a domain expert if she often correctly answers tasks related to that domain.

• **OTA**. When a worker comes and requests new tasks, this module assigns tasks to her. A poor assignment may not only waste budget and time, but also hurt the quality of inference results which depend on workers' answers. To judiciously assign tasks, the module makes decisions based on three factors: (1) the worker's quality, (2) the domain vectors of tasks, and (3) how confident each task's truth can be inferred from previously received answers. Intuitively, we assign a task to the worker if the task's domains are the worker's expertise and its truth cannot be confidently inferred. The assignment is done *online*, i.e., tasks will be assigned to the worker instantly.

However, designing the three modules above is not straightforward. We technically address the above challenges as follows:

For **DVE**, although we can extract entities from a task based on existing entity linking algorithm [39], ambiguities exist when we link each extracted entity to real-world concepts (e.g., pages in Wikipedia). For example, the entity *Michael Jordan* can either refer to the famous basketball player (https://en.wikipedia.org/wiki/Michael_Jordan), or the computer scientist (https://en.wikipedia.org/wiki/Michael_I._Jordan) in Wikipedia. Suppose there are u entities in a task, and each entity can be linked to 3 concepts, then there are 3^u possible linkings from the entities to concepts, which is exponential. Thus deriving a domain vector for a task in a straightforward way involves aggregating an exponential number of such linkings. We propose an algorithm that can reduce the complexity from exponential to polynomial (Section 3).

For **TI**, it is challenging to infer each task's truth correctly, as it highly depends on workers' qualities (which are unknown). Intuitively, we exploit the inherent relations between workers' qualities and tasks' truth, and finally devise an iterative approach that collectively infers those parameters. We also study how to maintain each worker's quality in the long run and devise update policies for the incremental inference algorithms (Section 4).

Finally, for the **OTA** module, we have studied how to use the three factors above (that affect task assignment), in order to estimate the *benefit* of assigning each task to the worker, by considering *if* the task is answered by the worker; then we assign a set of k tasks that attain the highest benefits. There are several challenges. (1) How to define the *benefit* function? (2) How to estimate the answer given by the worker? (3) Typically for better user interaction, a set of k tasks (e.g., k = 20 in [54, 44]) will be batched together and assigned to the worker. To select the optimal k tasks out of all (say, n) tasks, there are $\binom{n}{k}$ possible combinations that have to be considered, then how to efficiently compute the optimal assignment? We have developed an optimal and linear algorithm to support this complex assignment process (Section 5).

To summarize, our main goal is to study the impact of using domain knowledge in the crowdsourcing process. We further examine how to use a knowledge base (KB) to realize this goal. We examine how to use a KB effectively and efficiently in the three key procedures of crowdsourcing, namely, (1) domain vector estimation (**DVE**), (2) truth inference (**TI**), and (3) online task assignment (**OTA**). To our understanding, no previous work has examined the

Table 1: Workers' Qualities and Answers for Task t_1.

Worker	Worker's Quality	Worker's Answer for Task t_1
w_1	$\boldsymbol{q^{w_1}} = [\,0.3, 0.9, 0.6\,]$	$v_1^{w_1} = 1$ ('*yes*')
w_2	$\boldsymbol{q^{w_2}} = [\,0.9, 0.6, 0.3\,]$	$v_1^{w_2} = 2$ ('*no*')
w_3	$\boldsymbol{q^{w_3}} = [\,0.6, 0.3, 0.9\,]$	$v_1^{w_3} = 2$ ('*no*')

use of domain knowledge in such a comprehensive manner. We present a simple architecture to integrate these processes, and our extensive experiments show that our solution outperforms state-of-the-art methods, i.e., [18, 30, 15, 16, 8, 54].

2. DATA MODEL

DEFINITION 1 (DOMAIN). *Let $\mathcal{D} = \{d_1, d_2, \ldots, d_m\}$ denote the domain set with $|\mathcal{D}| = m$ domains.*

An example domain set is $\mathcal{D}$ ={*politics*, *sports*, *films*}. The domain set is used to interpret tasks and profile workers, which can be obtained by existing knowledge bases or question answering systems, e.g., main topics in Wikipedia [47], domains in Freebase [20], or categories in Yahoo Answers [48]. The reason for using general topics is that they can interpret a task and profile a worker in a fine-grained manner. We record the worker's familiar domains, which can be further used when the same worker comes in the future.

DEFINITION 2 (TASK, DOMAIN VECTOR). *A requester publishes n tasks, denoted as $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$. Each task $t_i \in \mathcal{T}$ has a text description, followed by ℓ_{t_i} possible choices. Each task t_i is modeled as a domain vector $\boldsymbol{r^{t_i}} = [\, r_1^{t_i}, r_2^{t_i}, \ldots, r_m^{t_i}\,]$, where each $r_k^{t_i} \in [0, 1]$ $(1 \le k \le m)$ and $\sum_{k=1}^{m} r_k^{t_i} = 1$. The domain vector $\boldsymbol{r^{t_i}}$ represents the distribution that task t_i is related to each domain in $\mathcal{D}$. A higher value of $r_k^{t_i}$ means that task t_i is more related to domain d_k.*

In this paper, we focus on multiple-choice tasks. Now let us consider a task t_1: *"Does Michael Jordan win more NBA championships than Kobe Bryant?"*, and the same $\mathcal{D}$ as above. The task has $\ell_{t_1} = 2$ choices: {*yes*, *no*}. From the text description we know that the task is related to domains *sports* and *films* in $\mathcal{D}$ (note that *Michael Jordan* starred in the film "*Space Jam*" in 1996), and it is more relevant with *sports*, thus a reasonable domain vector for t_1 is $\boldsymbol{r^{t_1}} = [0, 0.78, 0.22]$ (we will show how to compute it in Section 3). We use bold font to represent a vector (e.g., $\boldsymbol{r^{t_1}}$) and the symbol $|\cdot|$ to get the size of a vector or set (e.g., $|\boldsymbol{r^{t_1}}| = |\mathcal{D}| = m$).

DEFINITION 3 (WORKER, QUALITY VECTOR). *Let $\mathcal{W}$ denote the worker set. Each worker $w \in \mathcal{W}$ is modeled as a quality vector $\boldsymbol{q^w} = [\, q_1^w, q_2^w, \ldots, q_m^w\,]$, where each $q_k^w \in [0, 1]$ indicates the expertise (accuracy) of worker w in answering tasks in domain d_k $(1 \le k \le m)$. A higher value q_k^w means that worker w has more expertise on domain d_k.*

Considering the same $\mathcal{D}$ above, if worker w is an enthusiastic sports-fan and movie-goer, while pays no attention to politics, then a proper quality vector for w is $\boldsymbol{q^w} = [0.3, 0.8, 0.8]$. Note that a worker can be an expert in multiple domains.

DEFINITION 4 (ANSWER, TRUTH). *Workers can come to the DOCS and answer tasks. Let ℓ_{t_i} denote the number of possible answers for task t_i, and v_i^w denote the answer given by worker w for task t_i, i.e., $1 \le v_i^w \le \ell_{t_i}$. We assume that a worker can answer a task at most once. Each task t_i has a (ground) truth, or true answer, denoted as v_i^* $(1 \le v_i^* \le \ell_{t_i})$.*

For the above example task t_1 and $\ell_{t_1} = 2$, suppose three workers (w_1, w_2 and w_3) give their answers in Table 1: $v_1^{w_1} = 1$ (*yes*), and $v_1^{w_2} = v_1^{w_3} = 2$ (*no*). The truth of t_1 is $v_1^* = 1$ (as *Michael* wins 6 *championships* while *Kobe* wins 5). Note that the truth v_1^* is unknown to us and we infer v_1^* based on workers' answers.

3. DOMAIN VECTOR ESTIMATION

We propose a two-step framework to compute $\boldsymbol{r^t}$ for task t.

Step 1: Extracting Entities, Concepts, and Indicator Vectors. Based on the advances in information retrieval [39, 42, 36, 10], we can leverage existing "entity linking" techniques [39] to detect entities in a task. Each detected entity can be linked to a set of possible concepts, which forms a probability distribution where each concept is associated with a probability that indicates the link from entity to concept is correct (by considering the semantic meanings in the text). For each concept, we can then use the hierarchical structure of a knowledge base to compute an indicator vector, expressing the domains in $\mathcal{D}$ that are related to the concept.

We use the task t_1 (denoted as t in this section) and $\mathcal{D}$ in Section 2 as an example. Table 2 shows the generated information. We denote $E_t = \{e_1, e_2, \ldots, e_{|E_t|}\}$ as the set of detected entities for task t, e.g., $|E_t| = 3$ and e_1 = *Michael Jordan*. For an entity $e_i \in E_t$, the distribution of all its possible correct concepts is denoted as $\boldsymbol{p_i} = [\, p_{i,1}, p_{i,2}, \ldots, p_{i,|\boldsymbol{p_i}|}]$, where each $p_{i,j}$ $(1 \le j \le |\boldsymbol{p_i}|)$ is the probability that the link from e_i to its j-th concept is correct. For example, for e_2 (*NBA*), we get $\boldsymbol{p_2} = [0.8, 0.2]$ for its two concepts. For the j-th concept in e_i, the computed indicator vector is denoted as $\boldsymbol{h_{i,j}} = [\, h_{i,j,1}, h_{i,j,2}, \ldots, h_{i,j,m}\,]$, where each $h_{i,j,k} = 1$ (0) means that the j-th concept in e_i is related (unrelated) to domain d_k. For example, as *Michael_B._Jordan* is an American actor, thus it is only related to domain *films* (i.e., d_3), and $\boldsymbol{h_{1,3}} = [0, 0, 1]$.

Step 2: Computing Domain Vector. We aggregate all entities of a task to compute its domain vector, by considering the correctness probability from an entity to a concept and the indicator vector of each concept in an entity. However, it is prohibitively expensive to compute the best domain vector (see Section 3.1).

3.1 Challenges in Computing Domain Vector

For a task t, based on step 1 we have all detected entities E_t, the distribution $\boldsymbol{p_i}$ of concepts for an entity e_i, and each concept's indicator vector $\boldsymbol{h_{i,j}}$. To compute the domain vector $\boldsymbol{r^t}$, we consider all correct linkings between entities and concepts. For example, in Table 2, one possible linking from the three entities in E_t to concepts is: e_1–"*Michael_B._Jordan*", e_2–"*National_Basketball_Association*", e_3–"*Kobe_Bryant*". The correctness of the linking is $p_{1,3} \cdot p_{2,1} \cdot p_{3,1} = 0.08$. Under this linking, the aggregated indicator vector is $\boldsymbol{h_{1,3}} + \boldsymbol{h_{2,1}} + \boldsymbol{h_{3,1}} = [0, 2, 1]$, which counts the number of related concepts in each domain, by considering all entities in task t. As the domain vector is a distribution, it is then normalized as $\frac{\boldsymbol{h_{1,3}}+\boldsymbol{h_{2,1}}+\boldsymbol{h_{3,1}}}{\sum_{k=1}^{m}(h_{1,3,k}+h_{2,1,k}+h_{3,1,k})} = [0, \frac{2}{3}, \frac{1}{3}]$. From the above analysis, we know that for a possible linking, we can derive its corresponding correctness probability and normalized vector.

For ease of presentation, we use $\boldsymbol{\pi} = [\pi_1, \pi_2, \ldots, \pi_{|E_t|}]$ to denote a possible linking, which means that e_i $(1 \le i \le |E_t|)$ is linked to the π_i-th possible concept of e_i. For example, the above linking corresponds to $\boldsymbol{\pi} = [3, 1, 1]$. Let $\Omega = \{\boldsymbol{\pi}\}$ denote a set containing all possible linkings, so $|\Omega| = \prod_{i=1}^{|E_t|} |\boldsymbol{p_i}|$. In this paper, we assume the entity is linked into different concepts independently. We will consider the issues of correlation among concepts in the future. Then for each linking $\boldsymbol{\pi} \in \Omega$, we can derive its corresponding correctness probability $\Pr(\boldsymbol{\pi}) = \prod_{i=1}^{|E_t|} p_{i,\pi_i}$ and normalized vector $\boldsymbol{v_\pi} = (\sum_{i=1}^{|E_t|} \boldsymbol{h_{i,\pi_i}})/(\sum_{k=1}^{m}\sum_{i=1}^{|E_t|} h_{i,\pi_i,k})$. As the normalized vector $\boldsymbol{v_\pi}$ is a distribution that reflects the degree of relatedness of task t to each domain w.r.t. the linking $\boldsymbol{\pi}$, thus by considering all possible $\boldsymbol{\pi} \in \Omega$, we define the domain vector $\boldsymbol{r^t}$ as the expected normalized vector, i.e.,

$$\boldsymbol{r^t} = \sum_{\boldsymbol{\pi}\in\Omega} \boldsymbol{v_\pi} \cdot \Pr(\boldsymbol{\pi}) = \sum_{\boldsymbol{\pi}\in\Omega} \frac{\sum_{i=1}^{|E_t|} \boldsymbol{h_{i,\pi_i}}}{\sum_{k=1}^{m}\sum_{i=1}^{|E_t|} h_{i,\pi_i,k}} \cdot \prod_{i=1}^{|E_t|} p_{i,\pi_i} \,. \quad (1)$$

Table 2: The Information Generated for Task t: *"Does Michael Jordan win more NBA championships than Kobe Bryant?"*.

Entity	Concept (Page in Wikipedia)	Linking Probability	Indicator Vector (of size m)
e_1: *Michael Jordan*	https://en.wikipedia.org/wiki/Michael_Jordan	$p_{1,1}$ = 0.7	$\boldsymbol{h_{1,1}}$ = [0, 1, 1]
	https://en.wikipedia.org/wiki/Michael_I._Jordan	$p_{1,2}$ = 0.2	$\boldsymbol{h_{1,2}}$ = [0, 0, 0]
	https://en.wikipedia.org/wiki/Michael_B._Jordan	$p_{1,3}$ = 0.1	$\boldsymbol{h_{1,3}}$ = [0, 0, 1]
e_2: *NBA*	https://en.wikipedia.org/wiki/National_Basketball_Association	$p_{2,1}$ = 0.8	$\boldsymbol{h_{2,1}}$ = [0, 1, 0]
	https://en.wikipedia.org/wiki/National_Bar_Association	$p_{2,2}$ = 0.2	$\boldsymbol{h_{2,2}}$ = [0, 0, 0]
e_3: *Kobe Bryant*	https://en.wikipedia.org/wiki/Kobe_Bryant	$p_{3,1}$ = 1.0	$\boldsymbol{h_{3,1}}$ = [0, 1, 0]

Algorithm 1: Domain Vector Computation

Input: E_t, $\boldsymbol{p_i}$ $(1 \le i \le |E_t|)$, $\boldsymbol{h_{i,j}}$ $(1 \le i \le |E_t|, 1 \le j \le |\boldsymbol{p_i}|)$
Output: $\boldsymbol{r^t}$

1 $x_{i,j} = \sum_{k=1}^{m} h_{i,j,k}$ for $1 \le i \le |E_t|, 1 \le j \le |\boldsymbol{p_i}|$; // pre-computation
2 $\boldsymbol{r^t} = [0, 0, \ldots, 0]$; // a vector of size m with all 0 elements
3 M = hash-map(); // we use M[key] to visit the value of the key (a 2-tuple)
4 **for** $k = 1$ to m (iterate over all domains) **do**
5 M[$(0, 0)$] = 1; // initialize the hash-map M
6 **for** $i = 1$ to $|E_t|$ (iterate over all entities) **do**
7 tmpM = hash-map(); // another hash-map, similar to M
8 **for** (nm, dm) $\in$ M (iterate over all keys in M) **do**
9 *value* = M[(nm, dm)]; // get the value of the key: (nm,dm)
10 **for** $j = 1$ to $|\boldsymbol{p_i}|$ (iterate over all concepts for entity e_i) **do**
11 **if** (nm + $h_{i,j,k}$, dm + $x_{i,j}$) $\notin$ tmpM **then**
12 tmpM[(nm + $h_{i,j,k}$, dm + $x_{i,j}$)] = 0;
13 tmpM[(nm + $h_{i,j,k}$, dm + $x_{i,j}$)] += *value* $\cdot\ p_{i,j}$;
14 M = tmpM; // assign tmpM to M for next iteration
15 **for** (nm, dm) $\in$ M (iterate over all keys in M to compute the value of r_k^t) **do**
16 **if** dm $\neq$ 0 **then**
17 r_k^t += (nm/dm) $\cdot$ M[(nm, dm)]; // aggregate the value of r_k^t
18 **return** $\boldsymbol{r^t}$;

Take the example in Table 2. By enumerating all possible $\boldsymbol{\pi} \in \Omega$ ($|\Omega|$=3 · 2 · 1=6) as Equation 1, the domain vector is computed as $\boldsymbol{r^t} = [0, 0.78, 0.22]$. However, directly computing $\boldsymbol{r_t}$ via Equation 1 is expensive. Let $c = \max_{1 \le i \le |E_t|} |\boldsymbol{p_i}|$, i.e., the maximum number of concepts in all entities, then it takes $\mathcal{O}(|\Omega| \cdot |E_t| \cdot m) = \mathcal{O}(c^{|E_t|} \cdot |E_t| \cdot m)$ time, which is exponential.

3.2 Our Solution

We devise our solution in Algorithm 1, which computes the domain vector accurately by reducing the complexity from $\mathcal{O}(c^{|E_t|} \cdot |E_t| \cdot m)$ (exponential) to $\mathcal{O}(c \cdot m^2 \cdot |E_t|^3)$ (polynomial). The basic idea is that although the number of possible linkings ($|\Omega|$) is exponential, the number of possible normalized vectors is bounded. For example, for the k'-th element in the normalized vector, i.e., $(\sum_{i=1}^{|E_t|} h_{i,\pi_i,k'})/(\sum_{k=1}^{m} \sum_{i=1}^{|E_t|} h_{i,\pi_i,k})$, as each $h_{*,*,*} \in \{0, 1\}$, if we consider its numerator and denominator respectively, there are at most $(|E_t| + 1) \cdot (m \cdot |E_t| + 1)$ possible values for that element. This inspires us to compute $\boldsymbol{r^t}$ from the perspective of normalized vectors in Algorithm 1.

To be specific, Algorithm 1 takes m iterations, where for the k-th iteration (lines 5-17), it computes the k-th element of $\boldsymbol{r^t}$, i.e., r_k^t. To achieve this, we use a hash-map (M), whose *keys* are the possible (numerator, denominator) combinations (denoted by (nm,dm)), and the corresponding *value* for a *key* (nm,dm) is the aggregated probability for nm/dm. In order to compute r_k^t, we consider each entity iteratively. In the i-th iteration (lines 7-14), it derives an M, whose *keys* are the possible (nm,dm) by considering the first i entities (i.e., from e_1 to e_i). In doing so, we leverage the derived M in last iteration (i.e., considering the first $i - 1$ entities) and directly applies the $\boldsymbol{p_i}$ and $\boldsymbol{h_{i,*}}$ of the i-the entity e_i to generate a new temporary hash-map tmpM. To be specific, for each *key* (nm,dm) in M, it is updated based on concepts in e_i: for the j-th concept, the *key* (nm,dm) becomes new *key* (nm+$h_{i,j,k}$, dm+$x_{i,j}$), note $x_{i,j} = \sum_{k=1}^{m} h_{i,j,k}$, which is initially stored in line 1, and the *value* (or aggregated probability) is multiplied by $p_{i,j}$ and added to the *value* of new *key* in tmpM. At last tmpM is assigned to M for the next iteration (line 14). After all entities are considered ($|E_t|$ iterations), we can finally use the information in the derived M to compute r_k^t. (lines 15-17).

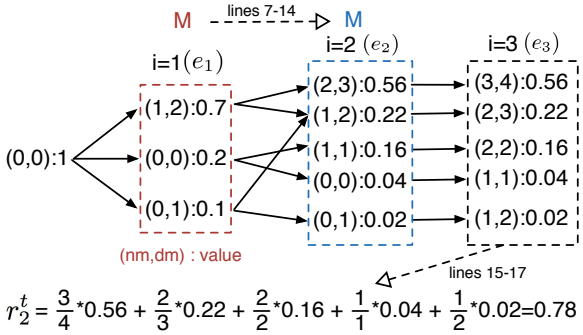

Figure 2: A Run of Computing r_2^t by Algorithm 1.

Running Example. Figure 2 shows how to compute r_2^t in Table 2. The i-th layer shows the derived M after considering e_i. We use '*key*:*value*' to represent each data in the hash-map. Initially for $i = 1$ (e_1), its three concepts (with $\boldsymbol{p_1}$, $\boldsymbol{h_{1,*}}$) form M. For e_2, based on the derived M in last iteration, it is updated to a new one. For example, (1, 2):0.7 is updated to (1+$h_{2,1,2}$, 2+$x_{2,1}$):0.7·$p_{2,1}$=(2,3):0.56 and (1+$h_{2,2,2}$, 2+$x_{2,2}$):0.7·$p_{2,2}$=(1,2):0.14. Moreover, as (0,1):0.1 can also be similarly updated as (1,2):0.08, thus the *value* for the *key* (1,2) is aggregated as 0.14+0.08=0.22. After all 3 entities are considered, r_2^t = 0.78 is finally derived based on the final M. Following this, we can compute $\boldsymbol{r^t} = [0, 0.78, 0.22]$.

Time Complexity of Algorithm 1. First the calculation of $x_{*,*}$ takes $\mathcal{O}(c \cdot m \cdot |E_t|)$ time, where $c = \max_{1 \le i \le |E_t|} |\boldsymbol{p_i}|$. Then it computes each element in $\boldsymbol{r^t}$ iteratively. To compute a r_k^t $(1 \le k \le m)$, we iteratively consider $|E_t|$ entities: for the i-th iteration, we leverage the already derived M to further update itself to a new one, by considering all concepts in e_i. In each iteration, as the number of *keys* in M is $\mathcal{O}(m \cdot |E_t|^2)$, then it takes $\mathcal{O}(c \cdot m \cdot |E_t|^2)$. So computing a r_k^t takes $\mathcal{O}(c \cdot m \cdot |E_t|^3)$, and the total time complexity of deriving $\boldsymbol{r^t}$ is $\mathcal{O}(c \cdot m^2 \cdot |E_t|^3)$.

The Implementations of DVE in DOCS. We adopt Freebase [20], a large, reliable, and curated knowledge base. We construct $\mathcal{D}$ based on 26 domains in Yahoo Answers [48], which consists of a wide range of topics, such as *Sports*, *Politics*. We manually map each of the 26 domains to the respective domain(s) in Freebase. Next we discuss how to compute E_t, $\boldsymbol{p_i}$, $\boldsymbol{h_{i,j}}$ and $\boldsymbol{r^t}$.

We use an open-source entity linking tool Wikifier [36, 10], which can detect entities E_t in a task t. For each entity e_i, it links to top 20 possible concepts (or pages) [36] in Wikipedia, by considering features such as the frequency of the linking and the string similarity between concepts and e_i. Then it computes a probability distribution $\boldsymbol{p_i}$ (of size 20), which indicates how probable each possible concept is correctly linked to e_i in the task. For each concept (in Wikipedia), which can be redirected to the corresponding Freebase concept using API, we compute the indicator vector $\boldsymbol{h_{i,j}}$ (of size 26) by considering whether or not each domain in $\mathcal{D}$ is related to the given concept in Freebase. Note that this can be obtained directly from the corresponding concept page in Freebase. Finally, based on E_t, $\boldsymbol{p_i}$ and $\boldsymbol{h_{i,j}}$ $(1 \le i \le |E_t|, 1 \le j \le |\boldsymbol{p_i}|)$, Algorithm 1 is leveraged to compute the task t's domain vector $\boldsymbol{r^t}$.

4. TRUTH INFERENCE

In this section, we study the truth inference problem. The **Input** of the problem includes: (1) tasks' domain vectors ($\boldsymbol{r^{t_i}}$ for $1 \leq i \leq n$) and (2) all workers' answers. The **Output** is each task's inferred truth (and we can also derive each worker's quality vector). To solve it, we first introduce an iterative approach (Section 4.1), and then discuss how to use it in practice (Section 4.2).

4.1 Iterative Approach

We observe that there are two kinds of relations between workers' qualities and tasks' truth: (1) *given a task t, if the worker's quality values for t's related domains are high, then her answer is likely to be the truth for t*; (2) *given a worker w, if w often answers tasks correctly related to a domain, then w has a high quality for that domain*. Based on these intuitions, we develop an iterative approach, which updates the sets of parameters for tasks and workers until convergence is reached. Here, we use $\boldsymbol{q^w}$ to denote a worker w's quality; $\boldsymbol{s_i} = [\, s_{i,1}, s_{i,2}, \ldots, s_{i,\ell_{t_i}}\,]$ is task t_i's probabilistic truth, where $s_{i,j}$ ($1 \leq j \leq \ell_{t_i}$) is the chance that the j-th choice is the truth for task t_i. We use $V^{(i)}$ to denote the set of workers' answers for task t_i. For example, in Table 1, $V^{(1)} = \{v_1^{w_1}, v_1^{w_2}, v_1^{w_3}\}$. We denote o_i as task t_i's true domain. Based on the domain vector $\boldsymbol{r^{t_i}}$, we have $\Pr(o_i = k) = r_k^{t_i}$.

To be specific, in our iterative approach, each iteration contains two steps: in step 1, each task's probabilistic truth $\boldsymbol{s_i}$ is inferred based on workers' qualities ($\boldsymbol{q^w}$); then step 2 reversely infers each worker's quality based on the tasks' probabilistic truth. It will iterate until convergence. Finally, we infer the truth for each task t_i as $v_i^* = \arg\max_{1 \leq j \leq \ell_{t_i}} s_{i,j}$ (detailed algorithm can be found in technical report [40]). We first detail the two steps, and then analyze the *Initialization* and *Time Complexity*, respectively.

Step 1: Inferring the Truth ($\boldsymbol{q^w \rightarrow s_i}$). In general, the probabilistic truth $\boldsymbol{s_i}$ can be expressed by considering all domains:

$$s_{i,j} = \Pr(v_i^* = j \mid V^{(i)}) = \sum_{k=1}^{m} r_k^{t_i} \cdot \Pr(v_i^* = j \mid o_i = k, V^{(i)}). \quad (2)$$

For simplicity, we denote $\mathcal{M}_{k,j}^{(i)} = \Pr(v_i^* = j \mid o_i = k, V^{(i)})$. Note that $\mathcal{M}^{(i)}$ is a matrix of size $m \times \ell_{t_i}$, where each row $\mathcal{M}_{k,\bullet}^{(i)}$ in it represents the distribution of truth computed for the k-th domain, then $\boldsymbol{s_i}$ can be computed by considering t_i's domain vector: $\boldsymbol{s_i} = \boldsymbol{r^{t_i}} \times \mathcal{M}^{(i)}$ via matrix multiplication. In order to compute $\mathcal{M}_{k,j}^{(i)}$, we adopt two typical assumptions used in existing works [30, 16, 29]: (1) workers give their answers independently and (2) the priors are uniform (i.e., $\Pr(v_i^* = j) = 1/\ell_{t_i}$). Then we can derive

$$\mathcal{M}_{k,j}^{(i)} = \frac{\prod_{v_i^w \in V^{(i)}} \Pr(v_i^w \mid o_i = k, v_i^* = j)}{\sum_{j'=1}^{\ell_{t_i}} \prod_{v_i^w \in V^{(i)}} \Pr(v_i^w \mid o_i = k, v_i^* = j')}. \quad (3)$$

Since there are ℓ_{t_i} possible answers for task t_i, to capture worker w's ability, similar to [54, 53], we compute the probability that the worker answers *incorrectly* (i.e., $1 - q_k^w$) for those ($\ell_{t_i} - 1$) *incorrect* answers using a uniform distribution. Let $\mathbb{1}_{\{\cdot\}}$ denote an indicator function which returns 1 if the argument is true; 0 otherwise. For example, $\mathbb{1}_{\{2=5\}} = 0$ and $\mathbb{1}_{\{5=5\}} = 1$. Then we have

$$\Pr(v_i^w \mid o_i = k, v_i^* = j) = (q_k^w)^{\mathbb{1}_{\{v_i^w = j\}}} \cdot \left(\frac{1 - q_k^w}{\ell_{t_i} - 1}\right)^{\mathbb{1}_{\{v_i^w \neq j\}}}. \quad (4)$$

This means that the probability that w answers correctly for a task with domain k is q_k^w, and likewise, the probability that w gives a specific incorrect answer for that task is $(1 - q_k^w)/(\ell_{t_i} - 1)$.

Running Example. We use an example to show that step 1 can satisfy the 1st relation. We compute $\boldsymbol{s_1}$ for task t_1 in Table 2. As the domain vector $\boldsymbol{r^{t_1}} = [0, 0.78, 0.22]$, and we take workers' qualities ($\boldsymbol{q^{w_1}}, \boldsymbol{q^{w_2}}, \boldsymbol{q^{w_3}}$) and answers ($V^{(1)}$) in Table 1. We first compute each vector $\mathcal{M}_{k,\bullet}^{(1)}$, e.g., for $\mathcal{M}_{2,\bullet}^{(1)} = [\mathcal{M}_{2,1}^{(1)}, \mathcal{M}_{2,2}^{(1)}]$, we derive $\mathcal{M}_{2,1}^{(1)} = \frac{q_2^{w_1}(1-q_2^{w_2})(1-q_2^{w_3})}{q_2^{w_1}(1-q_2^{w_2})(1-q_2^{w_3}) + (1-q_2^{w_1})q_2^{w_2}q_2^{w_3}} = 0.93$ and $\mathcal{M}_{2,2}^{(1)} = 0.07$. Similarly we derive $\mathcal{M}_{1,\bullet}^{(1)} = [0.03, 0.97]$ and $\mathcal{M}_{3,\bullet}^{(1)} = [0.28, 0.72]$. Then we compute $s_{1,j}$ ($1 \leq j \leq 2$) as $s_{1,1} = \sum_{k=1}^{3} r_k^{t_1} \cdot \mathcal{M}_{k,1}^{(1)} = 0.79$, and similarly $s_{1,2} = 0.21$. Note that although two workers w_2, w_3 answer "*no*" to t_1, and only one worker w_1 answers "*yes*" to t_1, the computed truth $\boldsymbol{s_1} = [0.79, 0.21]$ tends to be "*yes*". The reason is that (1) the task t_1 is more related to domain "*sports*" (0.78 in $\boldsymbol{r^{t_1}}$), and (2) w_1 has a high quality (0.9) for domain "*sports*" while w_2 and w_3 have low qualities (0.6 and 0.3) for it, making w_1's answer more reliable.

Step 2: Estimating Worker Quality ($\boldsymbol{s_i \rightarrow q^w}$). We now estimate $\boldsymbol{q^w}$ based on the computed $s_{i,j}$ in step 1. As q_k^w denotes worker w's quality for the k-th domain, which is formally defined as the fraction of tasks in domain d_k that w has answered correctly: $q_k^w = \frac{\sum_{t_i \in \mathcal{T}^{(w)}} \mathbb{1}_{\{o_i = k\}} \cdot \mathbb{1}_{\{v_i^w = v_i^*\}}}{\sum_{t_i \in \mathcal{T}^{(w)}} \mathbb{1}_{\{o_i = k\}}}$, where $\mathcal{T}^{(w)}$ denotes the set of tasks answered by worker w, e.g., in Table 1, $\mathcal{T}^{(w_1)} = \{t_1\}$. However, q_k^w is hard to compute directly, as task t_i's true domain (o_i) and truth (v_i^*) are unknown. Fortunately we have their probabilistic representations, which facilitate us to compute their expected values, i.e., $\mathbb{E}[\mathbb{1}_{\{o_i = k\}}] = r_k^{t_i} \cdot 1 + (1 - r_k^{t_i}) \cdot 0 = r_k^{t_i}$, and similarly $\mathbb{E}[\mathbb{1}_{\{v_i^w = v_i^*\}}] = s_{i,v_i^w}$. Then we take the expectation of the numerator and denominator of q_k^w and derive

$$q_k^w = \Big(\sum_{t_i \in \mathcal{T}^{(w)}} r_k^{t_i} \cdot s_{i,v_i^w}\Big) / \Big(\sum_{t_i \in \mathcal{T}^{(w)}} r_k^{t_i}\Big). \quad (5)$$

Intuitively, q_k^w is computed by considering the tasks answered by worker w. To be specific, it considers (1) how much each answered task is related to domain d_k, i.e., $r_k^{t_i}$; and (2) how probable each task is answered correctly by worker w, i.e., s_{i,v_i^w}.

Running Example. We use an example to show that step 2 can satisfy the 2nd relation. Suppose a worker w_1 answers two tasks: t_1 and t_2 ($\ell_{t_1} = \ell_{t_2} = 2$), both with the first answer. Assume $s_{1,1} = 0.95$, and $s_{2,1} = 0.3$; for domain vectors, assume $r_2^{t_1} = 0.9$ and $r_2^{t_2} = 0.05$. Then we get $q_2^{w_1} = (r_2^{t_1} \cdot s_{1,1} + r_2^{t_2} \cdot s_{2,1})/(r_2^{t_1} + r_2^{t_2}) = 0.92$ (Equation 5). Note that although w_1 answers poorly for t_2 ($s_{2,1} = 0.3$), the worker's quality for domain d_2 is still very high ($q_2^{w_1} = 0.92$). The reason is that t_2 is merely related to d_2 ($r_2^{t_2} = 0.05$); moreover, w_1 answers accurately to t_1 ($s_{1,1} = 0.95$), which is highly related to d_2 ($r_2^{t_1} = 0.9$), making q_2^w very high.

Initialization. To initialize all workers' qualities, we can leverage each worker's answering performance for *golden tasks* (Section 5.2), i.e., tasks with known ground truth, which are used to test a worker's quality before a worker answers real published tasks.

Time Complexity. In step 1, let $\ell = \max_{1 \leq i \leq n} \ell_{t_i}$, then for each task t_i, computing each $\mathcal{M}_{k,j}^{(i)}$ takes $\mathcal{O}(\ell \cdot |V^{(i)}|)$, thus this step takes $\mathcal{O}(m\ell^2 \cdot \sum_{i=1}^{n} |V^{(i)}|)$ time in all; in step 2, to compute each q_k^w, it considers all the tasks answered by w, thus this step takes $\mathcal{O}(m \cdot \sum_{w \in \mathcal{W}} |\mathcal{T}^{(w)}|) = \mathcal{O}(m \cdot \sum_{i=1}^{n} |V^{(i)}|)$. Suppose it takes u iterations to converge, then the time complexity is $\mathcal{O}(um\ell^2 \cdot \sum_{i=1}^{n} |V^{(i)}|)$ in total. In practice, m and ℓ are constants, and $u \leq 20$, thus the time complexity is linear to the number of answers.

4.2 Practical Truth Inference

We now study the practical issues about how to maintain workers' qualities for future use, and how to accelerate truth inference.

Maintenance of Workers' Qualities. As different requesters may publish different tasks to DOCS, the workers who have previously answered tasks may come again in the future. Thus we need to maintain workers' previous answering performance, which can be

further used (e.g., initializing worker's quality) in tasks published by new requesters. Obviously, it is ineffective to store all of workers' previous completed tasks and answers. A straightforward way is to store each worker w's quality $\boldsymbol{q^w}$. However, this is insufficient, as each q_k^w $(1 \le k \le m)$ is derived (Equation 5) based on w's answers for tasks in domain d_k, and if w answers very few tasks related to d_k, the computed q_k^w is not confident. Thus besides q_k^w, we also maintain another statistic u_k^w, i.e., the number of tasks w has answered that are related to d_k.

Specifically, in order to update a worker w's quality $\boldsymbol{q^w}$, e.g., q_k^w $(1 \le k \le m)$, DOCS maintains two statistics in database: the quality q_k^w and its weight u_k^w, which is the expected number of tasks answered by w that are related to domain d_k, i.e., $u_k^w = \sum_{t_i \in \mathcal{T}^{(w)}} r_k^{t_i}$. Suppose worker w came to DOCS before and answered tasks. Let $\widehat{q}_k^w$ and $\widehat{u}_k^w$ $(1 \le k \le m)$ denote two statistics stored for previous tasks, then by considering those computed for newly answered tasks (i.e., q_k^w and u_k^w for $1 \le k \le m$), Theorem 1 states how to update these two parameters correctly in DOCS.

THEOREM 1 (WORKER QUALITY UPDATE). *If q_k^w and u_k^w are updated as $(\widehat{q}_k^w \cdot \widehat{u}_k^w + q_k^w \cdot u_k^w)/(\widehat{u}_k^w + u_k^w)$, and $(\widehat{u}_k^w + u_k^w)$, respectively, then the quality of worker w is updated correctly.*

PROOF. See technical report [40] for the detailed proof. □

Accelerating Truth Inference. When a worker submits answers, the **TI** module is run and the parameters are updated and stored in the database (Figure 1). As **TI** takes an iterative approach, it *could* be expensive to run until convergence. To alleviate this issue, we develop an incremental approach. The challenges are three-fold: (1) What are the parameters to update upon receiving an answer? (2) How can we update those parameters instantly? (3) What parameters should we store in order to facilitate such updates?

W.l.o.g., assume worker w answers a task t_i with the a-th choice. Upon receiving the answer, the basic idea is that we only update the parameters that are *most* related to task t_i and worker w, i.e., task t_i's truth and the qualities of workers who have answered task t_i. To facilitate such updates, we store the following parameters: (1) for a worker w, based on Theorem 1, we store its quality q_k^w and weight u_k^w $(1 \le k \le m)$; (2) for a task t_i, we store its matrix $\mathcal{M}^{(i)}$ and the probabilistic truth $\boldsymbol{s_i}$. The update policy is as follows:

• **Step 1: Inferring the Truth**. In this step, we only update task t_i's parameters, i.e., $\mathcal{M}^{(i)}$ (Equations 3-4) and $\boldsymbol{s_i} = \boldsymbol{r^{t_i}} \times \mathcal{M}^{(i)}$. In fact, the process can be further accelerated by storing another parameter $\widehat{\mathcal{M}}_{k,j}^{(i)}$, which records the numerator in Equation 3.

• **Step 2: Estimating Worker Quality**. In this step, we update the qualifies of worker w and those who have answered t_i before. To be specific, (1) for worker w, $q_k^w = (q_k^w \cdot u_k^w + s_{i,a} \cdot r_k^{t_i})/(u_k^w + r_k^{t_i})$ and $u_k^w = u_k^w + r_k^{t_i}$; (2) if worker w' ever answered task t_i with j-th answer before, then $q_k^{w'} = (q_k^{w'} \cdot u_k^{w'} - \widetilde{s}_{i,j} \cdot r_k^{t_i} + s_{i,j} \cdot r_k^{t_i})/u_k^{w'}$, where $\widetilde{s}_{i,j}$ is the previous $s_{i,j}$ before update (i.e., step 1 above).

The complete algorithm is shown in technical report [40]. It is not hard to prove that the above two steps are bounded in time $\mathcal{O}(m \cdot |V^{(i)}|)$, which is more efficient than the iterative approach. Note that the incremental approach may not achieve as high quality as the iterative one; however, its advantage is that the parameters can be updated instantly, which fits to the scenario when workers' answers arrive in a high velocity. In practice, we can run **TI** in a *delayed manner*, that is, the iterative approach will be run in every z submissions of answers (z = 100 in DOCS).

5. ONLINE TASK ASSIGNMENT

When a worker comes to crowdsourcing platforms such as AMT [3], it instantly interacts with DOCS for task assignment. Specifically, AMT will pass the unique worker ID to us, then we dynamically assign a set of k tasks (e.g., k = 20 in [54, 44]) to the coming worker in AMT. In this section, we address two problems in **OTA**. First, if the worker has already completed some tasks, we select k tasks and assign them to the worker (Section 5.1). Second, if the worker is new, we investigate how to select representative *golden tasks* to test the worker's quality (Section 5.2).

5.1 Online Task Assignment

If the coming worker w has answered tasks before, we can retrieve related parameters from database. The **Input** of the problem includes (1) worker w's quality ($\boldsymbol{q^w}$), and (2) tasks' current information (i.e., $\mathcal{M}^{(i)}$ and $\boldsymbol{s_i}$ for $1 \le i \le n$). The **Output** of the problem is to select k tasks for worker w, from the task set $\mathcal{T} - \mathcal{T}^{(w)}$, i.e., the set of tasks not answered by worker w before.

To assign tasks, on one hand, we assign tasks with domains that the worker is good at; on the other hand, we have to evaluate if a task is really beneficial to be assigned. For example, for a task t_i that is confident enough based on previous answers (e.g., $\boldsymbol{s_i}$ = $[0.99, 0.01]$), then even if the coming worker is a domain expert for the task, it is of very minor benefit to assign the task.

Following the above discussions, we design an assignment framework, where for each task t_i, it estimates the *benefit* of assigning it to the coming worker, i.e., $\mathtt{B}(t_i)$; we then select k tasks that attain the highest benefits to the worker. In the following, we first focus on the problem of assigning $k = 1$ task, and then address the general problem of assigning k tasks.

Task Assignment for One Task ($k = 1$)

To address this, an important problem is how to estimate the benefit of a task, by considering *if* the task is answered by the worker? Recall that for a task t_i, we use a distribution $\boldsymbol{s_i}$ of size ℓ_{t_i} to capture its truth. Intuitively, the more concentrated $\boldsymbol{s_i}$ is (e.g., for a certain choice j $(1 \le j \le \ell_{t_i})$, the value $s_{i,j}$ is close to 1 while other $s_{i,j'}$ for $j' \ne j$ are close to 0), the more confident to derive the truth; otherwise, if $\boldsymbol{s_i}$ tends to be a uniform distribution, then the truth is ambiguous. By capturing this idea, we apply entropy [38] as the measure to define the ambiguity of a distribution $\boldsymbol{s_i}$, i.e., $\mathcal{H}(\boldsymbol{s_i}) = -\sum_{j=1}^{\ell_{t_i}} s_{i,j} \cdot \ln s_{i,j}$. The higher the value $\mathcal{H}(\boldsymbol{s_i})$ is, the more ambiguous $\boldsymbol{s_i}$ is. We then define the benefit function $\mathtt{B}(t_i)$.

DEFINITION 5 (BENEFIT FUNCTION $\mathtt{B}(\cdot)$). *For a task t_i, when a worker w comes, let $\widehat{\boldsymbol{s}}_{\boldsymbol{i}}$ denote the distribution after w answers the task, then the benefit of assigning t_i to worker w is defined as how much ambiguity can be reduced based on the assignment, i.e., $\mathtt{B}(t_i) = \mathcal{H}(\boldsymbol{s_i}) - \mathcal{H}(\widehat{\boldsymbol{s}}_{\boldsymbol{i}})$.*

However, the computation of $\mathtt{B}(t_i)$ is challenging, as $\widehat{\boldsymbol{s}}_{\boldsymbol{i}}$ is unknown before t_i is really answered by w. Then to estimate $\widehat{\boldsymbol{s}}_{\boldsymbol{i}}$, we have to consider the following two questions:

Q1: what answer the worker *may* give to the task, and
Q2: how the truth *will change* if the worker gives an answer.
Next we solve Q1 and Q2, respectively.

Solutions to Q1. In estimating the answer that will be given by w, we denote it as a random variable v_i^w $(1 \le v_i^w \le \ell_{t_i})$ and estimate it based on the collected answers, i.e., $\Pr(v_i^w = a \mid V^{(i)})$. By considering all possible true domain o_i and truth v_i^* for task t_i, we can express $\Pr(v_i^w = a \mid V^{(i)})$ as follows:

$$\sum_{k=1}^{m} \Pr(o_i = k) \sum_{j=1}^{\ell_{t_i}} \Pr(v_i^w = a \mid o_i = k, v_i^* = j) \Pr(v_i^* = j \mid o_i = k, V^{(i)}).$$

Then we can derive the following theorem that solves Q1.

THEOREM 2. *The probability that worker w will give the a-th choice to task t_i is*

$$\Pr(v_i^w = a \mid V^{(i)}) = \sum_{k=1}^{m} r_k^{t_i} \cdot \Big[q_k^w \cdot \mathcal{M}_{k,a}^{(i)} + \frac{1 - q_k^w}{\ell_{t_i} - 1} \cdot (1 - \mathcal{M}_{k,a}^{(i)}) \Big]. \quad (6)$$

PROOF. See technical report [40] for the detailed proof. □

Solutions to Q2. We talk about how the truth $\boldsymbol{s_i}$ will be updated. Suppose worker w gives the a-th choice to task t_i, and let $\mathcal{M}^{(i)|a}$ denote the updated matrix of $\mathcal{M}^{(i)}$. Based on Equations 3 and 4, we can derive the formula of each element in $\mathcal{M}^{(i)|a}$ in Theorem 3.

THEOREM 3. *If worker w gives the a-th choice to task t_i, then*

$$\mathcal{M}_{k,j}^{(i)|a} = \frac{\mathcal{M}_{k,j}^{(i)} \cdot (q_k^w)^{\mathbb{1}\{j=a\}} \cdot \left(\frac{1-q_k^w}{\ell_{t_i}-1}\right)^{\mathbb{1}\{j\neq a\}}}{\sum_{j'=1}^{\ell_{t_i}} \mathcal{M}_{k,j'}^{(i)} \cdot (q_k^w)^{\mathbb{1}\{j'=a\}} \cdot \left(\frac{1-q_k^w}{\ell_{t_i}-1}\right)^{\mathbb{1}\{j'\neq a\}}}. \quad (7)$$

PROOF. See technical report [40] for the detailed proof. □

Then we can update the truth $\boldsymbol{s_i}$ as $\widehat{\boldsymbol{s}}_{\boldsymbol{i}} = \boldsymbol{r}^{\boldsymbol{t_i}} \times \mathcal{M}^{(i)|a}$, by considering that worker w gives the a-th choice to t_i.

The solutions to Q1 and Q2 tell us how to compute the probability that w gives the a-th choice to t_i, and the updated truth $\widehat{\boldsymbol{s}}_{\boldsymbol{i}}$ if the answer is really given. Considering all possible answers, we define $\mathcal{H}(\widehat{\boldsymbol{s}}_{\boldsymbol{i}})$ in an expected manner, i.e.,

$$\mathcal{H}(\widehat{\boldsymbol{s}}_{\boldsymbol{i}}) = \sum_{a=1}^{\ell_{t_i}} \mathcal{H}(\, \boldsymbol{r}^{\boldsymbol{t_i}} \times \mathcal{M}^{(i)|a}\,) \cdot \Pr(v_i^w = a \mid V^{(i)}). \quad (8)$$

Then for each task t_i, we can compute $\mathcal{H}(\widehat{\boldsymbol{s}}_{\boldsymbol{i}})$ via Equations 6, 7, 8, and derive $\mathtt{B}(t_i)$ in Definition 5. We select the task with the highest benefit, i.e., $\arg\max_{t_i \in \mathcal{T}-\mathcal{T}^{(w)}} \mathtt{B}(t_i)$.

Task Assignment for k Tasks

To select k tasks out of the set $\mathcal{T} - \mathcal{T}^{(w)}$, it has two challenges:

Challenge I. For a fixed set of k tasks, denoted as $\mathcal{T}_k$ ($\mathcal{T}_k \subseteq \mathcal{T} - \mathcal{T}^{(w)}$ and $|\mathcal{T}_k| = k$), we need to consider all possible answers given by worker w. W.l.o.g., we assume $\mathcal{T}_k$ contains the first k tasks in $\mathcal{T}$ (i.e., $t_i \in \mathcal{T}_k$ for $1 \le i \le k$). Let $\boldsymbol{\phi} = [\,\phi_1, \phi_2, \ldots, \phi_k\,]$ denote one possible combination of answers given by w for $\mathcal{T}_k$, and $1 \le \phi_i \le \ell_{t_i}$. Then upon receiving the answers $\boldsymbol{\phi}$, following Definition 5, the benefit of k tasks is changed to

$$\mathtt{B}_{\boldsymbol{\phi}}(\mathcal{T}_k) = \sum_{i=1}^{k} \left[\, \mathcal{H}(\boldsymbol{s_i}) - \mathcal{H}(\boldsymbol{r}^{\boldsymbol{t_i}} \times \mathcal{M}^{(i)|\phi_i}) \,\right]. \quad (9)$$

Let all possible combinations of answers for $\mathcal{T}_k$ form a set $\Phi = \{\boldsymbol{\phi}\}$ and $|\Phi| = \prod_{i=1}^{k} \ell_{t_i}$. Then if we consider all $\boldsymbol{\phi} \in \Phi$, the expected benefit of assigning $\mathcal{T}_k$, denoted as $\mathtt{B}(\mathcal{T}_k)$ is expressed as

$$\mathtt{B}(\mathcal{T}_k) = \sum\nolimits_{\boldsymbol{\phi}\in\Phi} \mathtt{B}_{\boldsymbol{\phi}}(\mathcal{T}_k) \cdot \prod\nolimits_{i=1}^{k} \Pr(v_i^w = \phi_i \mid V^{(i)}). \quad (10)$$

From the above analysis we know that even for a fixed k-task set $\mathcal{T}_k$, computing its benefit (Equation 10) needs to require exponential number of combinations in Φ (as $|\Phi| = \prod_{i=1}^{k} \ell_{t_i}$).

Challenge II. Moreover, we need to select the optimal k tasks (i.e., with the highest value $\mathtt{B}(\mathcal{T}_k)$) out of the set $\mathcal{T} - \mathcal{T}^{(w)}$. This process requires enumerating all $\binom{n}{k}$ possible cases in the worst case.

To address the first challenge, fortunately we can prove the following theorem, which reduces the complexity from exponential to linear. The basic idea is that if we consider all possible answers for one task (e.g., t_1), then it can be safely proved that $\mathtt{B}(t_1)$ can be extracted from $\mathtt{B}(\mathcal{T}_k)$, and similarly other $\mathtt{B}(t_i)$ for $t_i \in \mathcal{T}_k - \{t_1\}$ can also be extracted and added independently.

THEOREM 4. $\mathtt{B}(\mathcal{T}_k) = \sum_{t_i \in \mathcal{T}_k} \mathtt{B}(t_i)$.

PROOF. See technical report [40] for the detailed proof. □

Theorem 4 implies that to compute the benefit for k tasks, we can compute each $\mathtt{B}(t_i)$ (Definition 5 and Equations 6, 7, 8) and add up individual benefit. Then in order to address the second challenge, i.e., to select the k tasks with highest $\mathtt{B}(\mathcal{T}_k)$, we only need to select top k tasks with the highest values of $\mathtt{B}(t_i)$, from the set $\mathcal{T} - \mathcal{T}^{(w)}$.

Time Complexity. To compute the benefit for each task (Equation 8), it should compute Equations 6, 7, which take $\mathcal{O}(m\ell^2)$ in all, where $\ell = \max_{1\le i\le n} \ell_{t_i}$. Then computing benefits for all tasks take $\mathcal{O}(nm\ell^2)$. As selecting top k values in a size n list can be addressed linearly (e.g., PICK algorithm in [7]), the time complexity for task assignment is $\mathcal{O}(nm\ell^2)$. Considering that m and ℓ are often constants, the complexity is linear to the number of tasks.

5.2 Selecting Golden Tasks

For a new worker, to test the worker's quality for different domains, we select some tasks with ground truth, called *golden tasks*. The golden tasks are selected after **DVE** is done. Then for each new worker from AMT [3], we assign the same golden tasks to her, and initialize her quality by comparing her answers with the ground truth for these tasks. However, we can manually label the ground truth (or refer to experts) for only a limited number of tasks. Thus given n tasks and a number $n' \ll n$, the problem is how to select the most representative n' tasks (out of n tasks) as *golden tasks*?

In order to profile each worker in a fine granularity w.r.t. n tasks, we consider two intuitive guidelines. (1) Each selected golden task should accurately capture a certain domain. For example, for the k-th domain, the selected task t_i should guarantee that its domain vector $\boldsymbol{r}^{\boldsymbol{t_i}}$ has a high value of $r_k^{t_i}$. (2) The distribution of selected golden tasks w.r.t. domains should approximate the distribution of all tasks' aggregated domain vectors. Let n'_k denote the number of tasks selected for the k-th domain, and $\sum_{k=1}^{m} n'_k = n'$. Then the distribution $\boldsymbol{\sigma} = [\frac{n'_1}{n'}, \frac{n'_2}{n'}, \ldots, \frac{n'_m}{n'}]$ should approximate the distribution $\boldsymbol{\tau} = [\frac{\sum_{i=1}^{n} r_1^{t_i}}{n}, \frac{\sum_{i=1}^{n} r_2^{t_i}}{n}, \ldots, \frac{\sum_{i=1}^{n} r_m^{t_i}}{n}]$.

Suppose we know n'_k for $1 \le k \le m$, then following the first guideline, for each domain d_k, we can select top n'_k tasks with the highest values of $r_k^{t_i}$, i.e., highly related to d_k. Then the remaining problem is how to decide each n'_k for $1 \le k \le m$?

Following guideline 2, to define the similarity of two distributions $\boldsymbol{\sigma}$ and $\boldsymbol{\tau}$, a widely-used metric is *KL-divergence* [26]: $D(\boldsymbol{\sigma}, \boldsymbol{\tau}) = \sum_i \sigma_i \cdot \ln(\sigma_i/\tau_i)$. It can be proved in [26] that $D(\cdot,\cdot) \ge 0$ and the lower the value is, the similar the two distributions are. Thus we aim to minimize $D(\boldsymbol{\sigma}, \boldsymbol{\tau})$ w.r.t. constraints as follows:

$$\min_{\{n'_k\}} \sum_{k=1}^{m} \frac{n'_k}{n'} \cdot \ln \frac{n'_k \cdot n}{n' \cdot \sum_{i=1}^{n} r_k^{t_i}} \quad \text{s.t. } \sum_{k=1}^{m} n'_k = n' \text{ and } n'_k \in \mathbb{N} \text{ for } 1 \le k \le m. \quad (11)$$

It is not hard to prove that solving Equation 11 is NP-hard, due to the fact that it is in general an "Integer Programming Problem" [33]. Despite its hardness, we devise an approximation algorithm (the detailed algorithm can be found in technical report [40]).

The general idea is to let each $n'_k/n' \approx \sum_{i=1}^{n} r_k^{t_i}/n$ for $1 \le k \le m$ w.r.t. constraints in Equation 11. To do this, we first derive a lower-bound for each n'_k and set $n'_k = \lfloor \sum_{i=1}^{n} r_k^{t_i}/n \cdot n' \rfloor$. Then a procedure is run for $n' - \sum_{k=1}^{m} n'_k$ times, and each time it conducts $n'_{ind} = n'_{ind} + 1$, where ind ($1 \le ind \le m$) is the choice of the domain with the minimum objective value if selected, i.e.,

$$ind = \min_k \{ \tfrac{n'_k+1}{n'} \cdot \ln \tfrac{(n'_k+1)\cdot n}{n' \cdot \sum_{i=1}^{n} r_k^{t_i}} + \sum_{j\neq k} \tfrac{n'_j}{n'} \cdot \ln \tfrac{n'_j \cdot n}{n' \cdot \sum_{i=1}^{n} r_j^{t_i}} \}.$$

Finally for each domain d_k, we select top n'_k tasks with the highest values of $r_k^{t_i}$, and then obtain all our selected n' golden tasks.

Time Complexity. To solve Equation 11, first computing the lower-bounds take $\mathcal{O}(m)$ time. For a real number x, we know $x \le \lfloor x \rfloor + 1$, then $n' = \sum_{k=1}^{m} \frac{\sum_{i=1}^{n} r_k^{t_i}}{n} \cdot n' \le \sum_{k=1}^{m} \lfloor \frac{\sum_{i=1}^{n} r_k^{t_i}}{n} \cdot n' \rfloor + m$, thus the procedure is run at most m times, which takes $\mathcal{O}(m^2 \cdot n)$ time. After solving Equation 11, selecting top n'_k tasks for different domains takes $\mathcal{O}(m \cdot n)$ time. So in total it takes $\mathcal{O}(m^2 \cdot n)$ time.

6. EXPERIMENTS

We evaluate DOCS on both real and simulated datasets. The settings are introduced in Section 6.1. Unless stated otherwise, real datasets are used to evaluate both the effectiveness and efficiency of the three modules: **DVE** (Section 6.2), **TI** (Section 6.3), and **OTA** (Section 6.4). We implement DOCS in Python 2.7 with the Django web framework on a 16GB memory Ubuntu server.

6.1 Settings

Real-World Datasets. We conduct experiments on AMT [3] with four real-world datasets.

ItemCompare Dataset (Item). It [18] contains 360 tasks with 4 domains: *NBA*, *Food*, *Auto* and *Country*, where each domain has 90 tasks. For each task, it asks workers to compare between two items. The task descriptions in each domain are highly similar, for example, in domain *Food*, each task compares which food (e.g., 'Chocolate' and 'Honey') contains more calories.

4-Domain Dataset (4D). It contains 400 tasks, which cover 4 domains: *NBA*, *Car*, *Film* and *Mountain*, where each domain has 100 tasks. Different from dataset Item, in 4D the task descriptions in each domain are not that similar, e.g., in domain *NBA*, the tasks vary in different forms: we ask the position of a player; compare the height (or age) of two players; compare which team wins more championships, etc. We manually label the ground truth.

Yahoo QA Dataset (QA). It [35] includes queries to a search engine in 2012-2014, and each query has a *best answer* in Yahoo Answers [48]. We select 1000 queries from the dataset, where for each query, we generate tasks related to the *best answer*. For example, "*Where does chili originate from, Texas or Turkey?*", where the *best answer* ('*Texas*') and the correct domain ('*Food&Drink*') can be extracted from the corresponding webpage [2].

SFV Dataset (SFV). It [30] contains 328 tasks, where each task asks the attribute of a person (e.g., the age of *Bill Gates*), and each task also shows a set of choices collected from different QA systems [25, 55], from which the workers select one as the correct choice. The ground truth is provided by [30].

Answer Collection. We publish tasks for the four datasets on AMT [3], which interacts with workers using Human Intelligence Task (HIT). When a worker comes, we batch $k = 20$ tasks in a HIT (same as [54, 44]) to the worker, and pay \$0.1 for the worker upon finishing the HIT. We assign each task to 10 different workers, so each dataset costs $\frac{360\times10}{20} \times \$0.1 = \$18$, \$20, \$50 and \$16.4, respectively. We select 20 *golden tasks* (Section 5.2) for each dataset. (1) In **TI** (Section 6.3), to make a fair comparison, we collect workers' answers as above and compare our solution (Section 4.1) with the existing methods on the *same* collected answers for each dataset. (2) In **OTA** (Section 6.4), as the assigned tasks for each coming worker may be totally different for different methods, to ensure that the same set of workers are used in comparisons, similar to [54], we assign tasks to a coming worker *in parallel* using different assignment methods. To be specific, there are 6 methods (see below) in comparison for task assignment, and when a worker comes, we use each method to assign 3 tasks, so $3 \times 6 = 18$ tasks are batched in a HIT (in random order) and assigned to the coming worker. We ensure that each method collects the same number of answers (e.g., 360×10 for dataset Item) in total. Then we compare with them on *respective* collected answers for each dataset.

Comparisons. We compare DOCS with existing methods: iCrowd (IC) [18], FaitCrowd (FC) [30], Majority Vote (MV), ZenCrowd (ZC) [16], David&Skene (DS) [15], AskIt! [8], QASCA [54]. As different methods focus on different perspectives, we compare with different methods in different modules.

(1) In **DVE** (Section 6.2), we compare with IC [18] and FC [30], which try to exploit the domain(s) of each task.

(2) In **TI** (Section 6.3), we compare with MV, ZC [16], DS [15], IC [18] and FC [30], which study truth inference.

(3) In **OTA** (Section 6.4), we perform end-to-end comparisons with methods AskIt! [8], IC [18], QASCA [54] and two baseline methods (Baseline and D-Max) that study online task assignment.

Evaluation Metric. For effectiveness, we use *Accuracy* to evaluate the quality of a method, and it measures the percentage of tasks whose truth are inferred correctly by the method. For efficiency, we measure the *Execution Time* of a method.

6.2 Evaluating Domain Vector Estimation

Evaluating the Accuracy of Domain Detection. We compare with two methods IC [18] and FC [30], which exploit diverse domains in a task. To be specific, IC uses Latent Dirichlet Analysis (LDA [6]) to model diverse domains in a task and compute the cosine similarity between pairwise tasks. It first manually sets the number of latent domains (m'), and then uses a generative model to capture how each task's text can be generated. Finally it learns a distribution (size m' vector) for each task, which indicates how it is related to each domain. Similarly, FC uses TwitterLDA [51], which is an adaptation of LDA [6] and focuses more on short texts (e.g., Tweets). It also sets the number of latent domains (m'') and then learns a specific domain for each task. To summarize, the two models used by IC and FC (1) only consider the text in each task, and (2) manually set the number of latent domains and learn a domain vector for each task w.r.t. latent domains. Different from them, DOCS (1) considers knowledge base (i.e., Freebase [20] with rich contextual and semantic information), and (2) learns a domain vector that captures the *explicit* domains (i.e., 26 domains in Yahoo Answers [48]), rather than the *latent* domains for each task. Next, we compare DOCS with IC and FC on four datasets.

• Datasets Item & 4D. For dataset 4D, there are 4 domains, and we manually set the latent m' and m'' as 4 to favor them. DOCS uses the default 26 explicit domains. After computing the domain vector for each task, we regard the domain with the highest probability as the detected domain for each task. Then we manually check for IC and FC on how each latent domain can be mapped to the 4 domains *NBA*, *Car*, *Film* and *Mountain*: for the domain vectors in tasks with true domain (say *NBA*), if we find that the probabilities in a latent dimension is very high, then we map that latent dimension to *NBA*. In DOCS, we can verify that the 4 domains are mapped to *Sports*, *Cars*, *Entertain* and *Science* in Yahoo Answers [48], respectively. Finally, we compute the domain detection accuracy per domain (the percentage of tasks that are detected correctly in the domain). Figures 3(a)(b) show the domain detection accuracy for each domain, by comparing with different methods. We also record their overall domain detection accuracy in Figure 3(e). It can be observed that in Item, the accuracy is very high ($\sim$100%) in all domains for three methods; however, in dataset 4D, our method DOCS performs much better compared with IC and FC, and the reason is that they use topic model-based methods (LDA [6] and TwitterLDA [51]), which perform well if the tasks in each domain have high string similarities (e.g., in Item, tasks in each domain compare two given items on the same metric); however, in 4D, as task descriptions vary in each domain, they cannot detect the correct domain. For example, both of the two methods detect the tasks that compare the heights between two basketball players and two mountains in the same domain, mainly because they have high string similarities. However, DOCS can detect the difference between them based on the *semantic* information of players and mountains, yielding the overall detection accuracy above 95%.

• Datasets QA & SFV. For dataset QA, although there are 26 domains in Yahoo Answers, most of the queries are related to *Entertain*, *Science*, *Sports* and *Business* (as most collected search engine queries [35] are related to them), so we only focus on tasks in those four domains and set the latent m' and m'' as 4. For dataset SFV, since each task asks a specific attribute for a person (e.g., age of *Bill Gates*), we manually label the ground truth of the person as his/her most renowned domain (e.g., *Business*). As most tasks are

Table 3: The Efficiency of Different Heuristics on DVE.

Dataset	Top-20 (Default)		Top-10		Top-3	
	Alg. 1	Enum.	Alg. 1	Enum.	Alg. 1	Enum.
Item	27.3s	>1 day	7.5s	>1 day	0.6s	1.3s
4D	28.6s	>1 day	8.8s	>1 day	0.8s	1.4s
QA	58.1s	>1 day	23.8s	>1 day	2.6s	264.7s
SFV	46.3s	>1 day	17.7s	>1 day	1.9s	406.8s

related to domains *Entertain*, *Business*, *Sports* and *Politics*, we focus on those tasks and set $m' = m'' = 4$. Similarly we compute the domain detection accuracy for the three methods as above in Figures 3(c)(d) and record the overall domain detection accuracy in Figure 3(e). It can be seen that in real-world question answering tasks, as tasks in each domain are not that similar in strings syntactically, IC and FC perform very bad (FC performs better than IC as it favors more on short texts); however, DOCS can capture the semantics in each domain and derive the correct domain accurately. As a result, it achieves over 20% improvement in overall accuracy.

Analysis on Multiple Domains. Note that for each task, the ground truth is only one domain. However, in real-world datasets (e.g., QA), each task can be related to multiple domains. Based on the computed domain vectors, we pick out those tasks whose domain vectors have more than one mode (or peak). Among them, we find some interesting cases. For example, in the task *"Is there a name for the whistle song that the Harlem Globetrotters are known for?"*, it is both related to domains '*Entertain*' and '*Sports*', and our computed domain vector has both high probabilities on those two domains. Similarly, we find that the task *"Who owns the Atalanta calcio (soccer/football) team in italy?"* is related to both '*Business*' and '*Sports*'; while the task *"What is the name of Simpson's episod, where Russia becomes Soviet Union?"* is related to both '*Entertain*' and '*Politics*'. Note that our framework also considers the relatedness of each domain to each task. In the future, it might be interesting to develop metrics on evaluating how a method can compute a task's multiple domains correctly.

Evaluating the Efficiency of DVE. We next compare the efficiency of *Enumeration* ($\mathcal{O}(c^{|E_t|} \cdot |E_t| \cdot m)$) and Algorithm 1 ($\mathcal{O}(c \cdot m^2 \cdot |E_t|^3)$) on different heuristics. In DOCS, we set $m = 26$, and E_t is the set of detected entities by Wikifier [36], which extracts top $c = 20$ related concepts for each entity. The efficiency of different methods on different datasets is shown in Table 3. It can be seen that on all four datasets, Algorithm 1 completes within one minute; on the other hand, *Enumeration* needs more than 1 day to finish. We have also compared the efficiency with two heuristics, which remove low-probability concepts, and only extract top $c = 10$ and $c = 3$ concepts for each entity. It can seen in Table 3 that although *Enumeration* performs fine on small datasets with few entities (e.g., 4D, Item), it is outperformed significantly (more than 100 times) by Algorithm 1 on QA and SFV. The reason is that *Enumeration* takes more time in QA-based tasks with a large number of entities.

6.3 Evaluating Truth Inference

In this section we evaluate **TI** in DOCS. We first exploit different aspects of our **TI**, and then compare our **TI** with other competitors. Finally we perform a case study on a dataset (Item).

Convergence. We run the iterative **TI** on the collected answers and identify the change of parameters between adjacent iterations. Let $\bar{s}_{i,j}$ and $\bar{q}_k^w$ denote the probabilistic truth and worker quality in the last iteration, then the change of parameters Δ between the adjacent two iterations is defined as $\Delta = \sum_{i=1}^{n} \sum_{j=1}^{\ell_{t_i}} \frac{|s_{i,j} - \bar{s}_{i,j}|}{n \cdot \ell_{t_i}} + \sum_{w \in \mathcal{W}} \sum_{k=1}^{m} \frac{|q_k^w - \bar{q}_k^w|}{|\mathcal{W}| \cdot m}$. We vary the number of iterations and record Δ in Figure 4(a). It can be seen that Δ drops significantly in the first 10 iterations and remains steady (convergence) ever since. In practice, we can terminate **TI** within a few (say 20) iterations.

Varying #Golden Tasks. As we initialize each worker's quality using golden tasks, in Figure 4(b) we vary the number of golden tasks in $[0, 40]$ and observe the change of accuracy on different datasets. It can be seen that the quality is significantly improved with a small number of golden tasks, as the iterative approach requires a good initialization. However, when the golden tasks are aplenty (say 20), the accuracy remains steady. For practical usage, we set the number of golden tasks as 20, which works well in experiments.

Varying #Collected Answers. As we collect each dataset by assigning each task to exactly 10 workers, in Figure 4(c), we vary the number of collected answers in $[1, 10]$ for each task and observe the accuracy of **TI** on different datasets. It can be seen that the accuracy becomes better as more answers are collected. However, for some dataset such as Item, it remains stable as ≥ 8 answers are collected. We will study the estimation of stable point in future.

Worker Quality Estimation. We next examine, when the worker answers more tasks, whether a worker's quality is closer to the worker's true quality. We first compute each worker w's true quality $\widetilde{\boldsymbol{q}}^{\boldsymbol{w}}$ by comparing the worker's answers with the ground truth. Here, $\widetilde{q}_k^w$ $(1 \leq k \leq m)$ is the fraction of the number of correctly answered tasks by worker w, among all her answered tasks in domain d_k. Then, we vary the number of collected answers (in $[1, 50]$) for each worker, and run **TI** to compute each worker w's quality $\boldsymbol{q}^{\boldsymbol{w}}$. Finally, we record the average deviation, defined as $\sum_{w \in \mathcal{W}} \sum_{k=1}^{m} \frac{|\widetilde{q}_k^w - q_k^w|}{m \cdot |\mathcal{W}|}$, in Figure 4(d). As workers answer more tasks, the average deviation decreases. Also, when 80 or more tasks are answered, the deviation becomes consistently low.

Scalability of TI (Simulation). We create n tasks with $m = 20$. Then the worker set $\mathcal{W}$ is generated, and each task is assigned to 10 randomly selected workers from $\mathcal{W}$. We vary $n \in$ [0, 10K], $|\mathcal{W}| \in \{10, 100, 500\}$ and run **TI** on randomly generated workers' answers. Figure 4(e) records the time. It can be seen that the time linearly increases with n, and the worker size is invariant with time, which corresponds to the complexity $\mathcal{O}(cm\ell^2 \cdot \sum_{i=1}^{n} |V^{(i)}|)$. Given that truth inference can be done offline, it is efficient, as it takes less than 15s with large data ($n = 10$K and $|\mathcal{W}| = 500$).

Comparing DOCS with Competitors on TI. We compare with other five competitors (i.e., MV, ZC, DS, IC and FC) on truth inference: (1) MV regards the truth of a task as the answer given by the highest number of workers; (2) ZC [16] models each worker's quality as a value, and adapts EM framework [17] to iteratively compute worker's quality and truth; (3) DS [15] models each worker's quality as a matrix, and also adapts EM framework [17] to iteratively compute worker's quality and truth; (4) IC [18] models a worker's quality for each task, and derives the truth using weighted majority voting; (5) FC [30] models a worker's quality as a vector of size m'', indicating the worker's quality for different *latent* domains, and it iteratively drives truth and worker's quality.

To make a fair comparison, we initialize the workers' qualities of all other competitors using the *same golden tasks*. Note that as IC and FC exploit the domains of each task, and the domain detection accuracy is not satisfactory (Figure 3), to do a more challenging job, we initially assign the *ground truth* of each task's domain to IC and FC, and then compute the truth of each task by them. We show the comparison results on both the effectiveness (*Accuracy*) and efficiency (*Execution Time*) of all datasets in In Figures 5(a)(b). We have the following observations: (1) MV is surpassed by other competitors, as it does not model a worker's quality and regard each worker as equal. (2) ZC and DS model a worker as a value or matrix, which does not consider a worker's quality for different domains, and that is why they are outperformed by more advanced methods. (3) IC, FC and DOCS model a worker's qualities for dif-

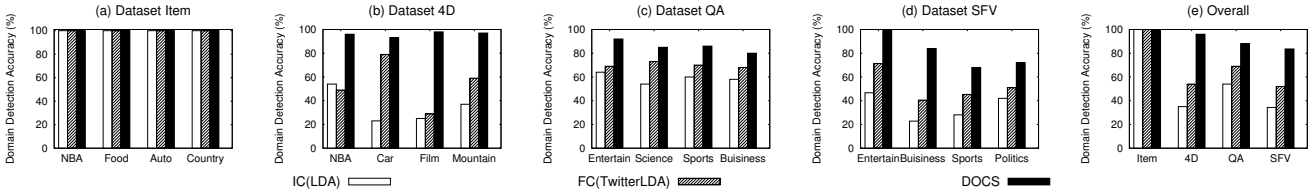

Figure 3: The Domain Detection Accuracy of Different Methods Per Domain (a-d) and the Overall Domain Detection Accuracy (e).

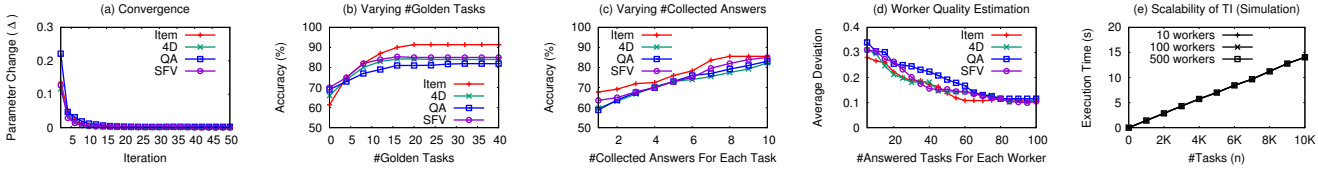

Figure 4: Exploiting Different Aspects of the Truth Inference (TI) in DOCS.

ferent domains, and DOCS outperforms other competitors on all datasets. Note that although IC and FC have been assigned with the ground truth of each task's domain in truth inference, we still outperform them a lot because our designed approach can capture the inherent relations between workers' qualities and tasks' truth accurately; while for FC, the modeled relations cannot capture the inherent relations very well, and for IC, it uses weighted majority voting, whose result is easy to be misled by low-quality workers. (4) For efficiency (Figure 5(b)), MV is the fastest as it can directly compute the truth of each task, while others adopt an iterative approach. IC is the least efficient as it first takes a preliminary offline computation, which will then facilitate online inference. As truth inference can be done offline, all methods can work well in practice. Dataset QA costs more time as it is larger than others.

We perform a case study on Item to show workers' qualities.

Worker's True Quality Across Domains. Similar to *Worker Quality Estimation*, we first compute each worker w's *true quality* $\widetilde{\boldsymbol{q}}^{\boldsymbol{w}}$ for four domains (q_k^w). In each domain ($1 \le k \le 4$), we split each worker w's quality q_k^w into 10 bins: it falls into the i-th bin ($1 \le i \le 9$) if $q_k^w \in [\frac{i-1}{10}, \frac{i}{10})$, and in the 10-th bin if $q_k^w \in [0.9, 1.0]$. Then for each domain, we record the number of workers that fall into each bin in Figure 6(a). It can be seen that most workers are good at answering tasks related to *Auto* (as there are $\ge$15 workers with quality ≥ 0.9); while workers have relatively low qualities on answering tasks related to *Food*. This means that it is necessary to select domain experts in all workers.

Worker Quality Calibration. Similar to the above, we first compute each worker w's *true quality* $\widetilde{\boldsymbol{q}}^{\boldsymbol{w}}$. Then we study whether the estimated quality q_k^w by DOCS is close to the true quality $\widetilde{q}_k^w$. In Figure 6(b), we select 3 workers who have answered the highest number of tasks and study their qualities. Specifically, the three workers are identified by labels '×', '⊡', '⊙'; each worker w corresponds to 4 points, where each point ($\widetilde{q}_k^w$, q_k^w) corresponds to a domain d_k ($1 \le k \le 4$). In the ideal case, all points should lie on the line $Y = X$, which means that the quality is estimated the same as the true quality. We observe that (1) a worker has diverse qualities in different domains. For example, the worker with label '⊙' has high qualities on two domains, and low qualities on other domains. (2) We can estimate a worker's quality accurately, as the points drawn in the figure lie very close to the line $Y = X$. For the domain *NBA* (d_1), we further plot the points for all workers who have performed more than one HIT (i.e.,> 20 tasks) in Figure 6(c). We can observe that in general, q_1^w is close to $\widetilde{q}_1^w$.

6.4 Evaluating Online Task Assignment

We first evaluate golden tasks selection, and then compare with other competitors in task assignment on respective collected datasets.

Evaluating Golden Tasks Selection (Simulation). The key of selecting golden tasks (Section 5.2) is to solve Equation 11, which enumerates all possible vectors $[n'_1, n'_2, \ldots, n'_m]$ such that $\sum_{k=1}^{m} n'_k = n'$ and $n'_k \in \mathbb{N}$ ($1 \le k \le m$). This is called "*Compositions of n' with size m (0 is allowed)*" [12], and it consists of $\binom{n'+m-1}{m-1}$ possible cases. By enumerating all cases, we can derive the optimal vector which obtains the minimum *KL-divergence*, i.e., D_{opt}. We can also compute D based on our solution (Section 5.2). We set $m = 10$, vary $n' \in [0, 20]$, and for each n', a distribution of size m, i.e., $\boldsymbol{\tau}$ is randomly generated. Then we record the time of both methods in Figure 7(a). It can be seen that the time for Enumeration increases exponentially with n', and when $n' = 20$, it takes more than 600s; while DOCS is efficient, which takes <0.01s. We also compute the approximation ratio, defined as $\gamma = |D - D_{opt}|/D_{opt}$ over all experiments, and the average γ is within 0.1%, which means that our computed results are very close to optimum. Next, we evaluate the scalability of our solution in Figure 7(b). We vary $n' \in$ [1K, 10K], $m \in \{10, 20, 50\}$ and observe the execution time. It can be seen that for a given m, the time is invariant with n', because the method that solves Equation 11 takes $\mathcal{O}(m^2 \cdot n)$, i.e., independent of n'.

Comparing DOCS with Competitors on OTA. We compare with other five competitors (i.e., Baseline, AskIt!, IC, QASCA, and D-Max) that address task assignment. Note that task assignment also requires truth inference method to derive worker's quality and infer task's truth: (1) Baseline uses MV to infer truth and randomly selects k tasks to assign to the coming worker; (2) AskIt! [8] uses MV to infer truth and leverages an entropy-like method to select k tasks for assignment; (3) IC [18] uses weighted majority voting to infer truth, and intuitively, it selects k tasks for assignment such that the coming worker has the highest quality to answer, with the constraints that each task should be answered with the same number of times (in our scenario, exactly 10 times) in the end; (4) QASCA [54] uses DS [15] to infer truth, and it selects k tasks such that the estimated quality (Accuracy in our scenario) can be improved most, and assigns them to the worker. (5) D-Max uses **TI** (Section 4) to infer truth, and it selects k tasks for assignment such that the coming worker has the matching domain to answer.

When comparing with different methods, we follow the instructions in Section 6.1 and assign $k = 3$ tasks using each method in parallel. There are 360×10 (3.6K), 4K, 10K, and 3.28K assignments in total for the four datasets. We show the Accuracy after all assignments and record the worst case assignment time in Figures 8(a)(b). We have the following observations: (1) Baseline performs worst as it randomly assigns tasks and does not consider the tasks' truth information or the coming worker's quality; (2) AskIt! considers tasks' truth in assignment, but does not take worker's quality into account; (3) QASCA performs better as it considers both the tasks' truth and worker's quality in assignment, but it does not take the domain information of tasks and workers into account; (4) IC captures a worker's quality for answering different tasks; however, it

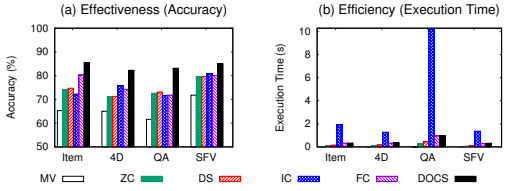

Figure 5: Comparisons on TI.

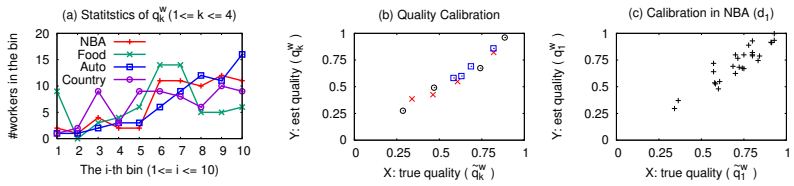

Figure 6: Case Studies of Worker Qualities on Dataset Item.

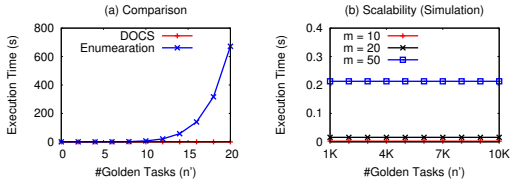

Figure 7: Golden Tasks Selection (Simulation).

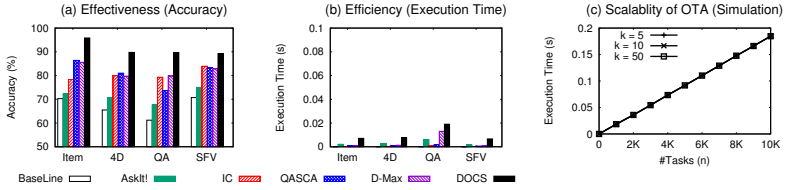

Figure 8: Online Task Assignment (OTA) Comparisons.

selects k tasks such that the worker has the highest quality to answer, which may assign tasks that are already confident enough. Furthermore, it restricts that each task should be answered with the same times, which does not consider that the assignments for the easy tasks can be saved for hard tasks; (5) although D-Max uses the **TI** (Section 4) to infer truth, it selects k tasks with the matching domain to the coming worker, which may assign tasks that are already confident enough; (6) DOCS performs the best, outperforming the best of other competitors consistently on all datasets. The reason is that we consider three factors: tasks' truth, worker's quality and the domain information. We estimate the benefit *if* a task will be assigned and answered by the worker, and selects the optimal k tasks which lead to the highest benefits; (7) all methods can finish the assignment efficiently, as it can be seen in Figure 8(b) that the worst case assignment is within 0.02s.

Scalability of OTA (Simulation). We generate n tasks with $m = 20$. Then we randomly generate the coming worker w's quality and the matrix $\mathcal{M}^{(i)}$ for each task t_i. Finally k tasks are assigned to worker w by running methods in Section 5.1. We vary $n \in [0, 10K]$, $k \in \{5, 10, 50\}$ and record the time in Figure 8(c). It can be seen that the assignment time increases linearly with n, which corresponds to the complexity $\mathcal{O}(nm\ell^2)$. We can also observe that the assignment time is independent of k. This is because that we use PICK algorithm [7] to select top k tasks with highest benefits, which is slightly affected by k. The assignment is efficient, and it can be finished within 0.2s with large data (n = 10K and k = 50).

7. RELATED WORKS

Crowdsourced Data Management. With the development in crowdsourcing (see a survey [27]), several platforms are built (e.g., AMT [3], CrowdFlower [13], ChinaCrowd [4]), enabling users to publish tasks and crowd workers to perform tasks. To facilitate query processing, crowdsourced databases (e.g., CrowdDB [19], Qurk [31]) are built on top of crowdsourcing platforms to support crowdsourced queries, e.g., join [43, 45, 9], selection [34], top-k [14], etc.

Task Model. Most existing crowdsourcing works [16, 15, 54, 8, 34] do not differentiate between tasks. Recently, [46] models the difficulty in tasks, while [18] and [30] exploit the diverse domains in each task using topic models (i.e., LDA [6] and TwitterLDA [51]). However, [18, 30] require a user to input the number of *latent* domains and cannot capture a task's related domain(s) explicitly and correctly, without considering the semantics in texts. We leverage the knowledge base (i.e., Freebase [20]), which has rich contextual and semantics information and can capture a task's diverse domain(s) in an explicit and accurate way. Note that there are other works [52, 49] that model a task and worker's diverse domains, but they do not consider knowledge base and leverage the external information, e.g., an answer with thumbs-up and thumbs-down voted by other workers. However, in reality the external information is hard to get, thus we do not assume them available and only leverage tasks' text descriptions and workers' answers. There are also some crowdsourcing works [11, 5] that consider knowledge bases, but they focus on different perspectives, e.g., [11] studies data cleaning and [5] focuses on crowd mining.

Truth Inference. Existing works [16, 15, 18, 30, 46, 56, 28] often model a worker's quality in order to enable truth inference. The basic idea is that the answer given by high quality workers is trusted in deriving each task's truth. A worker's quality can be modeled in different ways. Majority Voting does not consider a worker's quality and regards each worker as equal. ZC [16] and [46] treat a worker's quality as a value, while DS [15] uses a matrix to model a worker's quality. [56] leverages the idea of minimax entropy and models the answer given by a worker to a task as a probability distribution. IC [18] captures diverse *latent* domains in tasks, and models the worker's quality in answering each task explicitly; similarly, FC [30] models a worker as a vector whose size is equal to the number of *latent* domains, and captures a worker's quality in answering tasks related to each *latent* domain. DOCS exploits the inherent relations between tasks' truth and workers' qualities. As shown in our experiments, IC and FC are outperformed by DOCS, even if tasks' true domains have been assigned to them in advance.

Our iterative truth inference algorithm (Section 4.1) is related to EM solutions [16, 15], which also iteratively updates parameters until they converge. However, as shown in our experiments (Section 6), our method outperforms EM solutions significantly. This is mainly because neither the task's domain vector nor the worker's expertise on different domains are considered in those works. Also, this algorithm converges quickly (less than 20 iterations) in our experiments (Section 6), and is thus highly practical.

Online Task Assignment. This problem of how to assign k tasks to a worker is of fundamental importance in crowdsourcing. In [43, 19, 53], a task is randomly selected; also, each task has to be answered by the same number of workers. Recently, AskIt! [8] uses previous tasks' answers to assign the k most uncertain tasks to the worker. In IC [18], these k tasks are chosen in such a way that a worker has the k highest quality values of answering them. It also requires each task to be answered by the same number of workers. QASCA [54] selects k tasks that attain the highest improvement in quality. In [21, 22, 41], the target is to maximize the overall utility given a fixed budget: [21] models a bipartite graph between workers and tasks, and designs an incentive-compatible mechanism for assignment; [22] extends the online adwords problem and sets the number of times that each task should be answered; [41] models the problem by proposing a multi-armed bandit model and devises efficient solutions to solve it. Different from the above works, DOCS considers three factors in assignment: truth estimation of tasks, di-

versity of domains, and a worker's quality. Intuitively, DOCS assigns a task to the worker if its domains are within the worker's expertise, and its truth cannot be confidently inferred yet.

Knowledge-intensive crowdsourcing solutions require external information, e.g., workers' wages and acceptance ratio in [37], and a complete skill taxonomy tree, a worker's exact skills and the required skills of tasks in [32], which are hard to obtain in real crowdsourcing platforms (e.g., AMT [3]). While our work considers typical crowdsourcing settings used in existing platforms.

8. CONCLUSIONS

In this paper we build a system DOCS, which contains three main modules: **DVE**, **TI** and **OTA**. After a requester submits tasks, **DVE** leverages the KB to interpret the domains for each task, and then DOCS interacts with AMT [3] adaptively. When a worker submits answers, **TI** is run to infer workers' qualities and tasks' truth, by exploring their inherent relations. When a worker requests for new tasks, **OTA** dynamically assigns k tasks with the highest *benefits* to the worker. We conduct experiments to test the effectiveness and efficiency, showing that DOCS outperforms existing state-of-the-art methods on the three modules.

Acknowledgement. We would like to thank the reviewers for the insightful comments. Guoliang Li was supported by 973 Program of China (2015CB358700), NSF of China (61373024, 61632016, 61422205, 61472198), Shenzhou, Tencent, FDCT/116/2013/A3, and MYRG105 (Y1-L3)-FST13-GZ. Reynold Cheng and Yudian Zheng were supported by the Research Grants Council of Hong Kong (RGC Projects HKU 17229116 and 17205115) and the University of Hong Kong (Projects 102009508 and 104004129).

9. REFERENCES

[1] https://docs.aws.amazon.com/AWSMechTurk/latest/RequesterUI/amt-ui.pdf.
[2] http://answers.yahoo.com/question/index?qid=20071211155603AAKwtyr.
[3] Amazon mechanical turk. https://www.mturk.com/.
[4] Chinacrowd. http://www.chinacrowds.com.
[5] Y. Amsterdamer, S. B. Davidson, T. Milo, S. Novgorodov, and A. Somech. Oassis: query driven crowd mining. In *SIGMOD*, pages 589–600, 2014.
[6] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3(Jan):993–1022, 2003.
[7] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973.
[8] R. Boim, O. Greenshpan, T. Milo, S. Novgorodov, N. Polyzotis, and W. C. Tan. Asking the right questions in crowd data sourcing. In *ICDE*, pages 1261–1264, 2012.
[9] 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.
[10] X. Cheng and D. Roth. Relational inference for wikification. In *EMNLP*, pages 1787–1796, 2013.
[11] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *SIGMOD*, pages 1247–1261, 2015.
[12] Compositions. http://mathworld.wolfram.com/Composition.html.
[13] CrowdFlower. http://crowdflower.com/.
[14] S. B. Davidson, S. Khanna, T. Milo, and S. Roy. Using the crowd for top-k and group-by queries. In *ICDT*, pages 225–236, 2013.
[15] A. P. Dawid and A. M. Skene. Maximum likelihood estimation of observer error-rates using the em algorithm. *Applied statistics*, pages 20–28, 1979.
[16] G. Demartini, D. E. Difallah, and P. Cudré-Mauroux. Zencrowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In *WWW*, pages 469–478, 2012.
[17] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.
[18] J. Fan, G. Li, B. C. Ooi, K.-l. Tan, and J. Feng. icrowd: An adaptive crowdsourcing framework. In *SIGMOD*, pages 1015–1030, 2015.
[19] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. Crowddb: answering queries with crowdsourcing. In *SIGMOD*, pages 61–72, 2011.
[20] Freebase. https://www.freebase.com/.
[21] G. Goel, A. Nikzad, and A. Singla. Allocating tasks to workers with matching constraints: truthful mechanisms for crowdsourcing markets. In *WWW*, pages 279–280, 2014.
[22] C.-J. Ho and J. W. Vaughan. Online task assignment in crowdsourcing markets. In *AAAI*, pages 45–51, 2012.
[23] H. Hu, G. Li, Z. Bao, Y. Cui, and J. Feng. Crowdsourcing-based real-time urban traffic speed estimation: From trends to speeds. In *ICDE*, pages 883–894, 2016.
[24] H. Hu, Y. Zheng, Z. Bao, G. Li, J. Feng, and R. Cheng. Crowdsourced POI labelling: Location-aware result inference and task assignment. In *ICDE*, pages 61–72, 2016.
[25] H. Ji, R. Grishman, H. T. Dang, K. Griffitt, and J. Ellis. Overview of the tac 2010 knowledge base population track. In *TAC*, 2010.
[26] S. Kullback and R. A. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.
[27] G. Li, J. Wang, Y. Zheng, and M. J. Franklin. Crowdsourced data management: A survey. *TKDE*, 28(9):2296–2319, 2016.
[28] Y. Li, J. Gao, C. Meng, Q. Li, L. Su, B. Zhao, W. Fan, and J. Han. A survey on truth discovery. *SIGKDD Explorations*, 17(2):1–16, 2015.
[29] 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.
[30] F. Ma, Y. Li, Q. Li, M. Qiu, J. Gao, S. Zhi, L. Su, B. Zhao, H. Ji, and J. Han. Faitcrowd: Fine grained truth discovery for crowdsourced data aggregation. In *KDD*, pages 745–754, 2015.
[31] A. Marcus, E. Wu, S. Madden, and R. C. Miller. Crowdsourced databases: Query processing with people. In *CIDR*, pages 211–214, 2011.
[32] P. Mavridis, D. Gross-Amblard, and Z. Miklós. Using hierarchical skills for optimized task assignment in knowledge-intensive crowdsourcing. In *WWW*, pages 843–853, 2016.
[33] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience, 1988.
[34] A. G. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom. Crowdscreen: algorithms for filtering data with humans. In *SIGMOD*, pages 361–372, 2012.
[35] QA. https://webscope.sandbox.yahoo.com/catalog.php?datatype=l&did=76.
[36] L. Ratinov, D. Roth, D. Downey, and M. Anderson. Local and global algorithms for disambiguation to wikipedia. In *ACL*, pages 1375–1384, 2011.
[37] S. B. Roy, I. Lykourentzou, S. Thirumuruganathan, S. Amer-Yahia, and G. Das. Task assignment optimization in knowledge-intensive crowdsourcing. *VLDBJ*, 24(4):467–491, 2015.
[38] C. E. Shannon. A mathematical theory of communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 5(1):3–55, 2001.
[39] W. Shen, J. Wang, and J. Han. Entity linking with a knowledge base: Issues, techniques, and solutions. *TKDE*, 27(2):443–460, 2015.
[40] Technical Report. http://i.cs.hku.hk/~ydzheng2/docs_full.pdf.
[41] L. Tran-Thanh, S. Stein, A. Rogers, and N. R. Jennings. Efficient crowdsourcing of unknown experts using bounded multi-armed bandits. *Artificial Intelligence*, 214:89–111, 2014.
[42] S. Trani, D. Ceccarelli, C. Lucchese, S. Orlando, and R. Perego. Dexter 2.0-an open source tool for semantically enriching data. In *ICWS*, pages 417–420, 2014.
[43] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *PVLDB*, 5(11):1483–1494, 2012.
[44] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng. Leveraging transitive relations for crowdsourced joins. In *SIGMOD*, pages 229–240, 2013.
[45] S. E. Whang, P. Lofgren, and H. Garcia-Molina. Question selection for crowd entity resolution. *PVLDB*, 6(6):349–360, 2013.
[46] J. Whitehill, T.-f. Wu, J. Bergsma, J. R. Movellan, and P. L. Ruvolo. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *NIPS*, pages 2035–2043, 2009.
[47] Wikipedia. https://en.wikipedia.org/wiki/Category:Main_topic_classifications.
[48] Yahoo Answers. https://answers.yahoo.com/dir/index.
[49] L. Yang, M. Qiu, S. Gottipati, F. Zhu, J. Jiang, H. Sun, and Z. Chen. Cqarank: jointly model topics and expertise in community question answering. In *CIKM*, pages 99–108, 2013.
[50] X. Zhang, G. Li, and J. Feng. Crowdsourced top-k algorithms: An experimental evaluation. *PVLDB*, 9(8):612–623, 2016.
[51] W. X. Zhao, J. Jiang, J. Weng, J. He, E.-P. Lim, H. Yan, and X. Li. Comparing twitter and traditional media using topic models. In *ECIR*, pages 338–349, 2011.
[52] Z. Zhao, F. Wei, M. Zhou, W. Chen, and W. Ng. Crowd-selection query processing in crowdsourcing databases: A task-driven approach. In *EDBT*, pages 397–408, 2015.
[53] Y. Zheng, R. Cheng, S. Maniu, and L. Mo. On optimality of jury selection in crowdsourcing. In *EDBT*, pages 193–204, 2015.
[54] 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.
[55] S. Zhi, B. Zhao, W. Tong, J. Gao, D. Yu, H. Ji, and J. Han. Modeling truth existence in truth discovery. In *KDD*, pages 1543–1552, 2015.
[56] D. Zhou, S. Basu, Y. Mao, and J. C. Platt. Learning from the wisdom of crowds by minimax entropy. In *NIPS*, pages 2195–2203, 2012.