Crowdsourced Data Management: A Survey

Guoliang Li Jiannan Wang Yudian Zheng Michael J. Franklin

Abstract—Any important data management and analytics tasks cannot be completely addressed by automated processes. These tasks, such as entity resolution, sentiment analysis, and image recognition can be enhanced through the use of human cognitive ability. Crowdsouring platforms are an effective way to harness the capabilities of people (i.e., the crowd) to apply human computation for such tasks. Thus, crowdsourced data management has become an area of increasing interest in research and industry.
We identify three important problems in crowdsourced data management. (1) Quality Control: Workers may return noisy or incorrect results so effective techniques are required to achieve high quality; (2) Cost Control: The crowd is not free, and cost control aims to reduce the monetary cost; (3) Latency Control: The human workers can be slow, particularly compared to automated computing time scales, so latency-control techniques are required. There has been significant work addressing these three factors for designing crowdsourced tasks, developing crowdsourced data manipulation operators, and optimizing plans consisting of multiple operators. In this paper, we survey and synthesize a wide spectrum of existing studies on crowdsourced data management. Based on this analysis we then outline key factors that need to be considered to improve crowdsourced data management.

Index Terms—Crowdsourcing, Human Computation, Data Management, Quality Control, Cost Control, Latency Control

1 Introduction

Existing algorithms cannot effectively address computer-hard tasks such as entity resolution [118], [121], [116], [122], sentiment analysis [139], [74], [82], and image recognition [124], [104], [129], which can benefit from the use of human cognitive ability. Crowdsourcing is an effective way to address such tasks by utilizing hundreds of thousands of ordinary workers (i.e., the crowd). Furthermore, access to crowd resources has been made easier due to public crowdsourcing platforms, such as Amazon Mechanical Turk (AMT) [1], CrowdFlower [2] and Upwork [3]. As a result, crowdsourcing has become an active area of research in the data management community.

There are many successful applications that utilize crowdsourcing to solve computer-hard tasks. For example, Von Ahn et al. digitized printed material by getting Internet users to transcribe words from scanned texts [117]. Their method achieved accuracy exceeding 99% and has transcribed over 440 million words. As another example, Eiben et al. utilized a game-driven crowdsourcing method to enhance a computationally designed enzyme [31].

Crowdsourcing can also benefit data management applications, such as data cleaning [119], [93], data integration [58], [127], [73], knowledge construction [9], [12]. Consider entity resolution as an example. Suppose a user (called the "requester") has a set of objects and wants to find the objects that refer to the same entity, perhaps using different names. Although this problem has been studied for decades, traditional algorithms are still far from perfect [118]. Alternatively, s/he can harness the crowd's ability to identify the same entity. To this end, the requester first designs the tasks (e.g., a task for every pair of objects that asks workers to indicate whether the two objects refer to the same entity). Then the requester publishes their tasks on a crowdsourcing platform such as AMT. Crowd workers who are willing to perform such tasks (typically for pay or some other reward) accept the tasks, answer them and submit the answers back to the platform. The platform collects the answers and reports them to the requester. As the crowd has contextual knowledge and cognitive ability, crowdsourced entity resolution can improve the quality [118], [121], [122]. We will discuss more about how a typical crowdsourcing platform (i.e., AMT) works in Section 2.

There are several important problems in crowdsourced data management as shown in Figure 1.

(1) Quality Control. Crowdsourcing may yield relatively low-quality results or even noise. For example, a malicious worker may intentionally give wrong answers. Workers may have different levels of expertise, and an untrained worker may be incapable of accomplishing certain tasks. To achieve high quality, we need to tolerate crowd errors and infer high-quality results from noisy answers. The first step of quality control is to characterize a worker's quality (called worker modeling). Then based on the quality model of workers, there are several strategies to improve quality. We can eliminate the low-quality workers (called worker elimination), assign a task to multiple workers and aggregate their answers (called answer aggregation), or assign tasks to appropriate workers (called task assignment). We discuss quality-control methods in Section 4.

(2) Cost Control. The crowd is not free, and if there are large numbers of tasks, crowdsourcing can be expensive. For example, in entity resolution, if there are $10,000$ objects, there will be about 50 million pairs. Even if the price per pair is 1 cent, it still takes lots of money. There are several effective cost-control techniques. The first is pruning, which first uses computer algorithms to remove some unnecessary tasks and then utilizes the crowd to answer only the nec-

- *Guoliang Li is with the Department of Computer Science, Tsinghua National Laboratory for Information Science and Technology (TNList), Tsinghua University, Beijing, China. E-mail: liguoliang@tsinghua.edu.cn.*
- *Jiannan Wang is with the School of Computing Science, Simon Fraser University, Burnaby, Canada. E-mail: jnwang@sfu.ca.*
- *Yudian Zheng is with the Department of Computer Science, University of Hong Kong, Pokfulam, Hong Kong. E-mail: zhydhkcws@gmail.com.*
- *Michael J. Franklin is with the AMPLab at UC Berkeley, Berkeley, CA. E-mail: franklin@berkeley.edu.*

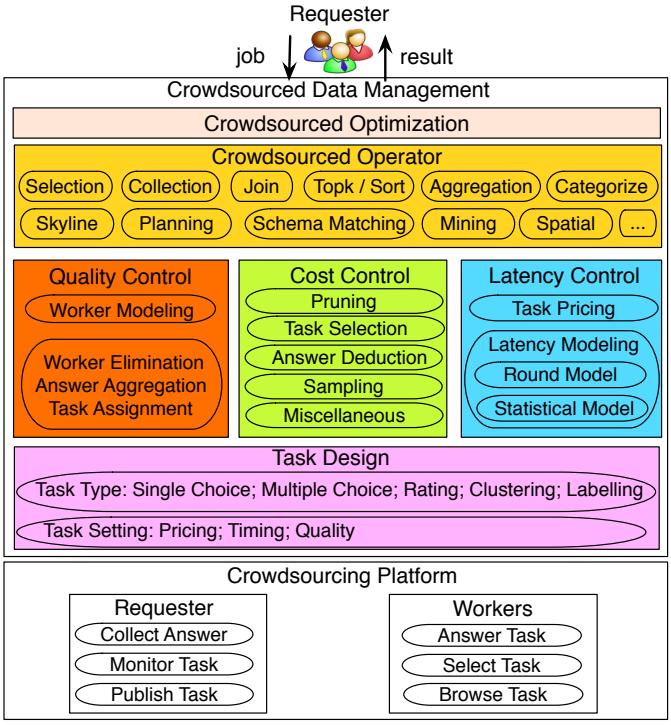


Fig. 1. Overview of Crowdsourced Data Management.

essary tasks. The second is task selection, which prioritizes which tasks to crowdsource. The third is answer deduction, which crowdsources a subset of tasks and based on the answers collected from the crowd, deduces the results of other tasks. The fourth is sampling, which samples a subset of tasks to crowdsource. There are also some specialized cost-control techniques mainly designed to optimize for specific operators. We review cost-control methods in Section 5.

(3) Latency Control. Crowd answers may incur excessive latency for several reasons: for example, workers may be distracted or unavailable, the tasks may not be appealing to enough workers, or the tasks might be difficult for most workers. If the requester has a time constraint it is important to control latency. There are several strategies for latency control. The first is pricing [44], [39]. Usually a higher price attracts more workers and can reduce the latency. The second is latency modeling [115], [103]. There are mainly two latency models: the round model and the statistical model. (a) The round model leverages the idea that tasks can be published in multiple rounds. If there are enough active workers on the crowdsourcing platform, the latency of answering tasks in each round can be regarded as constant time. Thus the overall latency is modeled as the number of rounds. (b) The statistical model is also used to model latency, which leverages the collected statistics from previous crowdsourcing tasks to build statistical models that can capture the workers' arrival time, the completion time, etc. These derived models can then be used to predict and perhaps adjust for expected latency. We review latency-control methods in Section 6.

Three additional components of crowdsourced data management are: task design, crowdsourced operator design, and optimization. Given a task (e.g., entity resolution), task design aims to design effective task types (e.g., devising a YES/NO question and asking workers to select an answer). Task design also needs to set the properties of tasks, e.g., deciding the prices, setting the time constraint, and choosing quality-control methods. We discuss task-design issues in Section 3. In addition, a crowdsourcing system can provide specialized operators for certain purposes. For example, entity resolution can use a crowdsourced join to find objects referring to the same entity. In data extraction, we need to use crowdsourced selection to select relevant data. In subjective comparison scenarios we need to use crowdsourced sort to rank the results. Many operator-specific techniques have been proposed to optimize cost, quality, or latency in crowdsourcing environments. We discuss crowdsourced operators in Section 7.

Furthermore for complicated queries with multiple operators an optimizer that chooses operators and decides their execution order is required. We review crowdsourced optimization techniques and crowdsourcing systems in Section 8. We then survey some widely-used crowdsourcing platforms in Section 9. We propose research challenges in Section 10 and finally conclude in Section 11.

To summarize, in this paper we survey a wide spectrum of work on crowdsourced data management. We also provide some key factors that need to be considered to improve the crowdsourced data management.

2 Background

In terms of granularity, tasks can be classified into macro-tasks (e.g., translating a paper) and micro-tasks (e.g., labeling an image). A micro-task usually takes several seconds to finish while a macro-task may take several hours. As the majority of existing crowdsourced data management works are focused on micro-tasks, we restrict the scope of this survey to micro-tasks as well. But we indeed believe that the studies of macro-tasks should be an important research topic in the future (see Section 10). In a crowdsourcing platform (e.g., AMT [1]), there are two types of users, called workers and requesters, who will deal with tasks. Requesters publish tasks to the platform; Workers perform tasks and returns the results. In the following, we describe the life-cycle of a task from the individual perspective of requesters and workers.

Suppose a requester has an entity resolution problem to solve, which aims to find the same entity from 1000 products. The requester needs to first design the user interface of a task (e.g., showing a pair of products and asking the crowd to choose between "the same" and "different"), and then set up some properties of the tasks, e.g., the price of a task, the number of workers to answer a task, the time duration to answer a task. After that, the requester publishes the tasks to the platform, and collects the answers from the crowd. We will discuss task design and task settings in Section 3.

From workers' perspective, they can browse and select the available tasks published by requesters. When accepting a task, they have to finish the task within the time duration specified by requesters. If a worker has accomplished a task, the requester who publishes the task can approve or disapprove the worker's answers. The approved workers will get paid from the requester.

3 Task Design

We discuss how to design task types (Section 3.1) and set up task settings (Section 3.2).

3.1 Task Types

There are several important task types that are widely used in real-world crowdsourcing platforms.

Single Choice. Workers select a single answer from multiple options. For example, in entity resolution, given a pair of objects, it asks the worker to select whether or not they refer to the same entity (options: Yes or No). In sentiment analysis, given a review, it asks the worker to assess the sentiment of the review (options: Positive, Neutral, Negative).
Multiple Choice. Workers select multiple answers from multiple options. For example, given a picture, workers select from a list, the objects (e.g., Monkey, Tree, Banana) that appear in the picture.
Rating. Workers rate multiple objects. For example, given several restaurants, it asks the crowd to give the ratings of these restaurants.
Clustering. Workers group a set of objects into several clusters. For example, in entity resolution, given several objects, the task is to create groups of objects that refer to the same entity.
Labeling. Workers provide a label for an object. For example, given a picture of an object, workers are asked to label the object, e.g., Apple, Banana. Note that labeling is different from other task types, as it is an "open task" and the workers can report any label instead of selecting some given labels.

In Section 7, we show that most operators can be implemented using these task types.

3.2 Task Settings

The requester also needs to determine some task settings based on her/his requirements. There are three main factors that the requester needs to consider.

3.2.1 Pricing

The requester needs to price each task. Usually task prices vary from a few cents to several dollars. Note pricing is a complex game-theoretic problem. Usually, high prices can attract more workers, thereby reducing the latency; but paying more does not always improve answer quality [39].

3.2.2 Timing

The requester can set time constraints for a task. For each task, the requester can set the time bound (e.g., 60 seconds) to answer it, and the worker must answer it within this time bound. The requester can also set the expiration time of the tasks, i.e., the maximum time that the tasks will be available to workers in the platform (e.g., 24 hours).

3.2.3 Quality Control

The requester can select the quality-control techniques provided by the crowdsourcing platform, or design her/his own quality-control methods. We will discuss quality-control techniques in the next section.

4 Quality Control

The results collected from the crowd are inherently dirty and ambiguous. Existing studies propose various quality-control techniques to address these issues. The basic idea of these works is to first characterize worker quality using a certain type of worker model (Section 4.1), and then based on the worker model, adopt different quality-control strategies such as eliminating low quality workers and spammers (Section 4.2), assigning a task to multiple workers and aggregating their answers (Section 4.3), or assigning informative tasks to high-quality workers (Section 4.4).

4.1 Worker Modeling

The first step of quality control is to characterize a worker's quality. We next discuss how existing works model a worker's quality (Section 4.1.1) and compute the parameters defined in the worker's model (Section 4.1.2).

4.1.1 Modeling a Worker

Existing works propose different ways to model the quality (or characteristic) of a worker. We summarize the proposed models as follows.

Worker Probability [49], [74], [133], [22], [138], [66]. *Worker probability* models each worker's quality as a single parameter $q \in [0, 1]$, indicating the probability that the worker correctly answers a task, i.e.,

$$q = \Pr(\text{ the worker's answer} = \text{task's true answer }).$$

For example, if a worker has a probability of 70% to correctly answer a task, the worker's quality is modeled as $q = 0.7$.

As q is defined as a probability (i.e., constrained in a range of $[0, 1]$), some works extend it in two ways:
(1) *Wider Range* [17], [64], [127] extends $q \in [0, 1]$ to a score in $(-\infty, +\infty)$. A higher score means a higher ability to correctly answer a task.
(2) *Confidence* [62], [61] extends the model by introducing another parameter for a worker: the confidence interval (e.g., [0.4,0.8]), which captures how confident the computed q is. Intuitively, as a worker answers more tasks, the worker's derived quality q becomes more confident (i.e., confidence interval becomes tighter, e.g., [0.75,0.8]).

Confusion Matrix [15], [58], [127], [112], [100]. *Confusion matrix* is commonly used to model a worker's capability for single choice tasks (with ℓ possible answers). For example, a sentiment analysis task asks workers to answer the correct sentiment (Positive, Neutral or Negative) of a given sentence, thus the task contains $\ell = 3$ possible answers. A confusion matrix is an $\ell \times \ell$ matrix:

$$Q = \begin{bmatrix} Q_{1,1}, & Q_{1,2}, & \ldots, & Q_{1,\ell} \\ Q_{2,1}, & Q_{2,2}, & \ldots, & Q_{2,\ell} \\ \vdots & \vdots & \ddots & \vdots \\ Q_{\ell,1}, & Q_{\ell,2}, & \ldots, & Q_{\ell,\ell} \end{bmatrix},$$

where the j-th ($1 \le j \le \ell$) row, denoted as $Q_j = [\, Q_{j,1}, Q_{j,2}, \ldots, Q_{j,\ell} \,]$, represents the probability distribution of the worker's possible answers for a task if the true answer of the task is the j-th answer. Then $Q_{j,k}$ ($1 \le j, k \le \ell$) indicates that given the fact that the true answer of a task is the j-th answer, the probability that the worker gives the k-th answer, i.e., $Q_{j,k} =$

$$\Pr(\text{ the worker's answer is } k \mid \text{task's true answer is } j \,).$$

For example, suppose we model a worker's quality for sentiment analysis tasks as $Q = \begin{bmatrix} 0.6 & 0.2 & 0.2 \\ 0.3 & 0.6 & 0.1 \\ 0.1 & 0.1 & 0.8 \end{bmatrix}$. If the first answer is Positive, the second is Neutral, and the third is Negative, then $Q_{3,1} = 0.1$ means that if the true answer of a task is Negative, the probability that the worker answers the sentiment as Positive is 0.1.

Bias and Variance [87]. *Bias and variance* are two statistical parameters that model a worker's capability of doing quantitative tasks. A quantitative task asks some quantitative

truth, which is commonly a real value rather than a predefined choice, e.g., counting the number of birds in an image. Bias, denoted as $\epsilon \in (-\infty, +\infty)$, represents the average estimation error of a worker. For example, if a worker usually overestimates a quantity, then $\epsilon > 0$; otherwise, if the quantity is usually underestimated by a worker, then $\epsilon < 0$. Variance, denoted as $\sigma^2 \in [0, +\infty)$, represents the variation of errors around the bias. If the true quantity of a task is t, then the worker's quantity (denoted as o) follows the Gaussian distribution: $o \sim \mathcal{N}(\ t + \epsilon\ ,\ \sigma^2\ )$. Intuitively, if worker's answers are stable around the mean (i.e., $t + \epsilon$), σ is close to 0; otherwise, σ deviates from 0.

Diverse Skills Across Tasks [35]. A worker may have diverse skills (or accuracies) across different tasks. For example, a worker may have high accuracies in answering tasks related to *Books* while having low accuracies in answering tasks related to *Fitness*. Inspired by this idea, Fan et al. [35] assume that each worker has diverse accuracies on different tasks and models a worker's quality $\vec{q}$ as: $\vec{q} = [q_1, q_2, \ldots, q_n]$, where n is the number of tasks. Each parameter in $\vec{q}$, i.e., q_i $(1 \leq i \leq n)$ models the probability that the worker correctly answers the i-th task, i.e., $q_i \in [0, 1]$.

Diverse Skills Across Domains [137], [136], [54], [55]. Different from the above model, Ho et al. [54], [55] model each worker's quality across different domains. Suppose there are $n = 100$ translation tasks, where 20 tasks are to translate from German to English, while the remaining 80 tasks are to translate from French to Chinese. This example has $K = 2$ known task domains. Generally, a worker's quality $\vec{q}$ is modeled with $K \leq n$ parameters: $\vec{q} = [q_1, q_2, \ldots, q_K]$, where each parameter $q_i \in [0, 1]$ represents the probability that the worker correctly answers tasks in the i-th domain.

The model in [54], [55] assumes that each task is related to a specific domain, which is known in advance. And a worker is modeled as K parameters with known domains. However, for complex tasks such as document review and text analysis, the specific domain of a given task is hard to be predefined, and a task may be related to multiple domains. Thus Zhao et al. [137], [136] model latent domains of tasks and the skills of workers for these latent domains. Then, suppose there are K' latent domains (K' is predefined), and each task is related to one latent domain (or combinations of several latent domains). A worker's quality is modeled as $\vec{q} = [\ q_1, q_2, \ldots, q_{K'}\ ]$, where $q_i \geq 0$ $(1 \leq i \leq K')$ indicates the worker's ability in the i-th latent domain. Unlike the above model, here the i-th domain cannot be explicitly explained. Then, Zhao et al. [137], [136] develop methods to iteratively derive each task's relevance to these K' latent domains (by exploiting their extracted features) and each worker's ability on these K' latent domains (by considering their answers to the tasks).

4.1.2 Computation of Worker Model Parameters

To derive the parameters in worker model (e.g., parameters in Confusion Matrix), the techniques used in existing works can be generally classified into the following categories:

Qualification Test [1]. Qualification test contains a set of golden tasks (tasks with known true answers). The workers must pass the qualification test (e.g., achieving a good quality on the golden tasks) before they can answer the real tasks. Based on worker's answers, the quality of the worker can be derived. For example, if a worker is modeled as Worker Probability and the worker correctly answers 8 out of 10 golden tasks, then the quality is $q = 8/10 = 0.8$.

Gold-Injected Method [74], [2]. The gold-injected method mixes golden tasks with real tasks. For example, suppose each time a worker is assigned with 10 tasks, where 20% of the tasks (i.e., 2 tasks) are set as golden tasks. Based on the worker' answers for the golden tasks, the worker's quality can be computed. Different from qualification test, workers do not know that some golden tasks are injected.

EM-Based Methods [139], [58], [15], [79], [118], [100], [127], [112], [41], [70], [104]. The EM (Expectation Maximization) algorithm is firstly proposed in [29], which leverages all workers' answers for all tasks, and iteratively updates each worker's quality and each task's true answer until convergence. The intuition is that the workers who frequently give trustworthy answers will be assigned with high qualities, and an answer supported by workers with high qualities will be selected as the true answer. General EM framework adopts an iterative approach, where each iteration takes E-step and M-step as follows:

E-step: it computes the distribution of *latent variables* using *parameters* derived from the previous M-step;

M-step: it computes *parameters* that maximize the expected log-likelihood function. The function is constructed using *latent variables* derived from the E-step.

For example, in [58], the *latent variables* contain the probability distribution of each task's true answers, and the *parameters* contain workers' qualities and the priors[1]. In the E-step, the probability of each task's true answers are computed based on the workers' qualities and the priors (computed from the previous M-step); in the M-step, the workers' qualities and the priors are updated based on the probability distribution of each task's true answers (derived from E-step).

Some existing works [139], [58], [15], [79], [118] directly apply the EM algorithm to derive parameters in workers' models. Other works [100], [127], [112], [41], [70], [84] study the variants of the EM algorithm. For example, Whitehill et al. [127] integrate tasks' difficulties into the EM algorithm to iteratively compute each worker's model.

Graph-Based Methods [64], [26], [28], [137], [136], [112], [22]. Existing works model a worker's quality as a node (or combinations of several nodes) in a defined graphical model[2] and leverage graphical model inference [68] to iteratively derive workers' models, where a graphical model is a graph, containing nodes and (directed or undirected) edges between pairs of nodes. Each node represents a random variable, which can be unknown parameters or observed data, and each edge represents the possible relationship (e.g., conditional dependency) between the linked pair of nodes. For example, Karger et al. [64] model workers and tasks using a bipartite graph, and propose an iterative approach based on Belief Propagation [94] to derive workers' models. Liu et al. [73] model workers and tasks using a directed graph, and apply variational inference methods to iteratively derive workers' models.

1. The priors are defined as a probability distribution of *all* tasks' true answers, which are global information.

2. Note that most models [127], [112], [124] solved by EM-based methods can also be represented as a graphical model.

4.2 Worker Elimination

Worker elimination, which eliminates low-quality workers or spammers based on worker models, is a very common strategy to improve quality [99], [58], [78]. There are different ways to detect low-quality workers or spammers.

A simple way is to use qualification test [1] or gold-injected method [2]. For example, before a worker answers real tasks, a qualification test that consists of golden tasks should be accomplished by the worker. Based on the worker's answers on the golden tasks, the worker's quality can be computed. Then, the workers with low qualities (e.g., Worker Probability < 0.6) are blocked to answer real tasks.

Other than worker probability model, there are some spammer detection methods based on more complex worker models. For example, Raykar et al. [99] and Ipeirotis et al. [58] study how to detect spammers if each worker is modeled as a Confusion Matrix (an $\ell \times \ell$ matrix). The basic idea is that for each worker, they compute a score based on the worker's Confusion Matrix, and the score represents how indicative the worker's answer is on a task's true answer. The higher the worker's score is, the more indicative her/his answer is on a task's true answer, and the more reliable s/he is. Then, they block the workers whose scores are below a predefined threshold.

In addition to detect spammers based on worker models, Marcus et al. [78] propose a detection method that is based on inconsistent answers given by different workers. They first compute the deviation of a worker's answer from the majority of other workers' answers. If the worker's answer deviates a lot from the majority of other workers' answers, the worker will be blocked.

4.3 Answer Aggregation

As a single worker may be biased for some tasks, most existing works [22], [69], [74], [139], [15], [58], [138] often employ a quality-control strategy, called answer aggregation (or voting strategy). The basic idea is to assign each task to multiple workers, and infer its result by aggregating all of its answers from these workers. Specifically, voting strategy can be seen as a function that takes two sets of parameters as input: (1) a task's answers; (2) the quality of each worker who answers the task. The output is the task's inferred result. Typical voting strategies include, Majority Voting [22], [69], Weighted Majority Voting [72], [54], Bayesian Voting [74], [139], [58], etc.

Majority Voting selects the result that receives the highest number of votes. Suppose a task is to ask workers to identify whether or not "IBM" is equal to "BIG BLUE". If the answers given by three workers (with qualities 0.2, 0.6 and 0.9) are Yes, Yes, and No, respectively, Majority Voting returns Yes as the result since it receives the highest number of votes (2 Yes vs. 1 No). A similar strategy, Weighted Majority Voting, also takes each worker' quality into account. A high-quality worker will be assigned with a higher voting weight. The answer with the highest aggregated weight will be returned as the result. For the same example, the aggregated weights of Yes and No are respectively 0.2+0.6=0.8 and 0.9, thus No is returned as the result. Another voting strategy, Bayesian Voting, which not only considers each worker's quality but also leverages Bayes' Theorem to compute the probability distribution of each answer being the true answer. In this example, assuming uniform priors, we have: $\Pr(\text{true answer is Yes}) \propto 0.2 \cdot 0.6 \cdot (1-0.9) = 0.012$, $\Pr(\text{true answer is No}) \propto (1-0.2) \cdot (1-0.6) \cdot 0.9 = 0.288$. By normalization, the probability distribution of each answer (Yes or No) being the true answer is $(\frac{0.012}{0.012+0.288}, \frac{0.288}{0.012+0.288}) = (4\%, 96\%)$, and Bayesian voting strategy will return No as the result as it has a higher probability to be the true answer.

According to whether or not the returned result has some randomness, Zheng et al. [138] classify voting strategies into two categories: deterministic and randomized voting strategies (A similar classification is also proposed in [89]). *Deterministic Voting Strategy* returns an answer as the result without any degree of randomness; *Randomized Voting Strategy* returns an answer as the result with some probability. In the above example, based on the computed probability distribution (4%, 96%), Bayesian Voting, a deterministic voting strategy, will deterministically return No as the result; while the Randomized Bayesian Voting, a randomized voting strategy, will return Yes with probability 0.04 and No with probability 0.96. As reported in [72], randomized voting strategies may improve the error bound for worst-case analysis.

4.4 Task Assignment

Since workers have diverse qualities on different tasks, a better task assignment strategy may lead to a higher quality result. Existing works studied two task assignment scenarios: (1) worker-based, i.e., given a task, which subset of workers should be selected to answer the task; (2) task-based, i.e., when a worker comes, which subset of tasks should be assigned to the worker.

4.4.1 Worker-based

In the worker-based scenario, given a task and a set of workers (with known qualities), intuitively the workers with high qualities (or having matching skills [137], [136] to the task) should be selected. In addition to these factors, *worker cost* is another key factor for the worker-based task assignment [138], [22], which is the monetary cost that each worker requires to answer a task. The cost can be indicated by the worker, or learned from the worker's profiles [22] (e.g., registration date, academic degree). Considering worker budget, Cao et al. [22] propose the Jury Selection Problem:

Given a task, a set of workers (with known qualities and costs) and an overall budget, how to select a subset of workers in order to maximize the task's quality without exceeding the overall budget?

To address this problem, Cao et al. [22] first studies how to compute the quality of a given subset of workers (before anyone gives an answer), called Jury Quality (or JQ). To characterize a worker's quality, a model such as Worker Probability can be used, indicating the worker's probability of correctly answering a task. However, to capture a set of workers' quality, it requires to aggregate all workers' possible answers. Thus the definition of JQ is specific to a voting strategy:

Given a set of workers with known qualities, how to compute the Jury Quality (or JQ), i.e., the probability of correctly returning a result w.r.t. a voting strategy?

Note that as workers' answers are unknown, to solve the problem, we need to consider all possible cases of workers' answers. For example, given three workers' qualities, we

aim at computing the JQ of the three workers w.r.t. the Majority Voting strategy. As in this case the Majority Voting strategy returns a task's result as the answer that receives at least 2 votes (out of all 3 votes), so in order to compute the JQ, i.e., the probability of correctly returning a result based on the three workers' answers w.r.t. Majority Voting strategy, it can be computed as the probability that at least 2 workers (out of 3) correctly answer the task, by considering all $\binom{3}{3}+\binom{3}{2}$=4 cases.

Cao et al. [22] propose an algorithm to compute JQ w.r.t. the Majority Voting strategy. The algorithm has a time complexity of $\mathcal{O}(|S| \cdot log|S|)$, where S is the given set of workers. Zheng et al. [138] prove that Bayesian Voting is the optimal strategy under the definition of JQ. That is, given any fixed S, the JQ of S w.r.t. the Bayesian Voting strategy is not lower than the JQ of S w.r.t. any other strategy. So given a set of workers, its collective quality (or JQ) w.r.t. Bayesian Voting strategy is the highest among all voting strategies. [138] further proves that the computation of JQ w.r.t. the Bayesian Voting strategy is NP-hard. To reduce the computational complexity, they propose an $\mathcal{O}(|S|^3)$ approximation algorithm, within 1% error bound. Based on JQ computation, both of the two works [22], [138] give the solution to the Jury Selection Problem.

4.4.2 Task-based

In the task-based scenario, when a worker comes, existing works [74], [18], [139], [35] study how to select the most informative tasks and assign them to the coming worker, to achieve high overall quality. The problem is defined as:

Given n tasks, when a worker comes, which k tasks (k is predefined) should be assigned to the coming worker?

When a worker comes, [74], [18] compute an uncertainty score for each task based on its collected answers, select the k most uncertain tasks, and assign them to the worker. There are multiple ways to define the uncertainty. Liu et al. [74] use a quality-sensitive answering model to define each task's uncertainty, and Boim et al. [18] leverage an entropy-like method to compute the uncertainty of each task.

Recently, Zheng et al. [139] find that different crowdsourcing applications may have different ways to define quality. In their approach, a crowdsourcing application first specifies a quality metric (e.g., Accuracy, F-score) that it would like to optimize on its data. To meet the requirement, the assignment algorithm will decide which k tasks should be assigned according to the specified metric. Specifically, for each combination of k tasks, it computes how much quality will be improved if they are assigned to a coming worker, and selects the combination that can lead to the maximum improvement in quality. There are some other works [35], [137], [136] that model workers to have diverse skills among different domains, and choose the tasks from the domains that a coming worker is good at to assign.

There are some works that study the task assignment problem in slightly different settings. Many machine learning [64], [54], [55] and active learning techniques [130], [82], [37], [140] aim to assign a set of tasks to workers that are most beneficial to their trained models. Budget allocation, which assumes that there is a fixed cost budget, aims to optimally allocate the budget to different tasks. Intuitively, difficult tasks should be allocated with higher budgets. Chen et al. [24] leverage the Markov Decision Process (MDP) [107] to address the budget allocation problem. Gao et al. [43] propose a cost-sensitive model to decide whether a task can be better solved by humans or machines. Mo et al. [81] study how to set the *plurality* (i.e., the number of workers to answer each task) under a fixed budget.

5 Cost Control

Despite the availability of crowdsourcing platforms, which provide a much cheaper way to ask humans to do some work, it is still quite expensive when there is a lot of work to do. Therefore, a big challenge for crowdsourced data management is cost control. That is, how to reduce human cost while still keeping good result quality. In this section, we abstract five classes of techniques from existing works. For each class of techniques, we will show how it can be applied to reduce human cost, and discuss its pros and cons.

5.1 Pruning

The first class of techniques is to use computer algorithms to pre-process all the tasks, and prune the tasks that do not need to be checked by humans. The underlying idea of this technique is that in many situations, there are a lot of tasks that can be easily finished by computers, thus humans only need to do most challenging ones. This idea has been widely applied to crowdsourced join [118], [121], [28], [122], [125], [116] and search [129]. For example, many crowdsourced join algorithms often utilize a computer-based technique, called similarity join [120], to remove the record pairs whose similarity values are smaller than a threshold.

Pros & Cons. The pruning-based technique is a very effective idea for cost control. It can save human cost by orders of magnitude with only a little loss in quality [118]. Furthermore, it is a very general idea because most crowdsourced operators have already had a lot of computer-only implementations. We can easily design a pruning strategy based on them. However, on the other hand, this process can be a risky step because if an improper pruning strategy is chosen, then for the tasks that are falsely pruned by computers, they will never be checked by humans again.

5.2 Task Selection

Task selection has been introduced in Section 4.4 as an idea to improve quality. From another point of view, this can also be seen as a class of techniques to reduce cost. That is, given a quality constraint, the task selection technique can minimize human cost to meet the quality requirement, by selecting most beneficial tasks for humans to do. Different crowdsourced operators need different selection strategies. Due to its effectiveness on cost control, it is a widely studied topic for a large variety of crowdsourced operators, such as join [59], [125], [114], [45], [82], top-k/sort [49], [23], [95], [33], [131], [67], categorize [91], etc. Note that these works are complimentary to the assignment strategies introduced in Section 4.4 because for a given crowdsourced operator, a task selection strategy is first used to decide a set of most beneficial tasks; once these tasks are selected and sent to a crowdsourcing platform, a task assignment strategy is then used to collect high-quality answers from the crowd.

Pros & Cons. The task selection technique provides a flexible way to tune the trade-off between cost and quality. One

can easily improve quality by using the technique to select more tasks or reduce the selected tasks to save cost. A side effect is that it might increase latency. This is because many task selection techniques need to iteratively query the crowd to decide which tasks can be selected next. In this iterative process, the crowd can only do a small number of tasks per iteration, which ignores the fact that there is a large pool of crowd workers available.

5.3 Answer Deduction

In some cases, the tasks generated by crowdsourced operators have some inherent relationships, which can be leveraged for cost control. Specifically, given a set of tasks, after getting some tasks' results from the crowd, we can use this information to deduce some other tasks' results, saving the cost of asking the crowd to do these tasks. Many crowdsourced operators have such property, e.g., join [121], [116], [122], [48], planning [63], [134], mining [10]. For example, suppose a crowdsourced join operator generates three tasks: (A, B), (B, C), and (A, C). If we have already known that A is equal to B, and B is equal to C, then we can deduce that A is equal to C based on transitivity, thereby avoiding the crowd cost for checking (A, C).

Pros & Cons. The deduction technique avoids the crowd to do a lot of redundant work. This even works when computers have already filtered a large number of easy tasks and left some hard tasks which only humans can finish. The deduction technique can further reduce human cost in this situation. However, the downside is that human errors can be amplified compared to not using the deduction. Consider the above example. If the crowd made a mistake in (B, C), the error would also be propagated to (A, C).

5.4 Sampling

A sampling-based technique only uses the crowd to process a sample of data and then utilizes the crowd's answers on the sample to extrapolate their answers on the full data. This class of techniques has been shown to be very effective in crowdsourced aggregation [78], [52], and data cleaning [119]. For example, Wang et al. [119] propose a sample-and-clean framework that allows the crowd to only clean a small sample of data and uses the cleaned sample to obtain high-quality results from the full data.

Pros & Cons. Sampling is a very powerful tool for cost control. Decades of research on sample estimates has built good theories that can effectively bound the statistical error of the estimates. However, the sampling idea does not work for all crowdsourced operators. For example, when applying sampling to a crowdsourced max operator, the estimated max from the sample can deviate a lot from the true max value.

5.5 Miscellaneous

There are some specialized cost-control techniques that are mainly designed to optimize cost for a particular operator. For example, Marcus et al. [78] propose a count-based user interface to reduce the number of tasks required for crowdsourced count. The task interface is to ask the crowd to provide an approximate count for the number of items that satisfy a given constraint, instead of asking them to exactly label whether each item satisfies the constraint or not. Trushkowsky et al. [110] propose a pay-as-you-go approach for crowdsourced enumeration. They estimate how much result quality can be improved by asking the crowd to do more tasks. If the improved quality is not satisfactory, it will stop sending tasks to the crowd, thus saving the cost. More details about these techniques will be covered in Section 7.

Pros & Cons. Compared to the other more general cost-control techniques in this section, this class of techniques makes better use of the characteristics of individual operator to optimize crowd cost. But, the limitation is that this technique is restricted to a particular operator and does not work for a large variety of crowdsourced operators.

5.6 Cost vs. Quality

Some cost-control techniques are used before a requester publishes tasks, e.g., pruning and sampling; while some techniques can be used iteratively, e.g., answer deduction and task selection. For example, answer deduction often employs an iterative way to publish tasks, which first publishes some tasks to a crowdsourcing platform, and then based on the results of the tasks from the crowd, decides which of the remaining tasks should be published in the next round. This iterative process will continue until all the tasks are finished.

In fact, these cost-control techniques can be used together. For example, we can first use the pruning idea to prune a lot of easy tasks. Then, for the remaining tasks, we can utilize the task-selection idea to decide which tasks should be selected for the crowd.

In addition, there is a tradeoff between quality and cost. The cost-control techniques may sacrifice the quality. For example, the answer deduction may reduce the quality if the crowd makes some mistakes in their answers, and the pruning can decrease the quality if some important tasks are pruned as discussed above. Thus, when using a cost-control technique, it is important to consider how to balance the trade-off between quality and cost [118], [121], [23].

6 Latency Control

To control latency, a direct way is to set tasks at a higher price, as a higher price can attract more workers, reducing the time of accepting and accomplishing tasks [39]. Based on this idea, existing works [93], [44] dynamically update the price for each task. Intuitively, as the tasks are gradually answered, the more urgent tasks are set at a higher price while the not-so-urgent tasks are set at a lower price.

Recently, Daniel et al. [51] systematically survey the dominant sources of latency in a data-labeling system, and propose three novel techniques to address the latency issue of each source: (1) *Straggler mitigation* uses redundant tasks to mitigate the impact of slow workers on latency; (2) *Pool maintenance* dynamically maintains a pool of fast workers; (3) *Hybrid learning* combines active and passive learning to avoid idle workers in the pool.

In addition, there are some latency models (round model and statistical model).

6.1 Round Model

Sometime, tasks are processed in multiple rounds, where in each round, a number of selected tasks are published to the crowd, and after their answers are collected, another set of selected tasks can be published in the next round. Suppose there are enough active workers, Sarma et al. [103] simplify the definition of latency by assuming that each round spends 1 unit time, and then the latency (or the total required time) is modeled as the number of rounds.

Some existing works [103], [115] use the round model to do latency control. Suppose there are n tasks and each round selects k tasks, then it requires n/k rounds in total to finish all tasks. To improve latency, they [103], [115] can use the answer-deduction idea in Section 5.3 to reduce the number of published tasks, that is, tasks may have relationships and the answers of tasks received in previous rounds can be leveraged to decide the chosen tasks in the next round. Thus the tasks can be saved for asking, for those whose results can be deduced based on the answers of other tasks collected from previous rounds. This in fact decreases the total number of tasks to be asked (i.e., $< n$), and if each round selects k tasks that cannot be deduced, the total number of rounds is $< n/k$, thus reducing the latency.

6.2 Statistical Model

Some existing works [129], [39] collect statistics from real-world crowdsourcing platforms and use such information to model workers' behaviors. Yan et al. [129] build statistical models to predict the time of answering a task. They consider two delays: delay for the arrival of the first response, and the inter-arrival times between two responses. (1) The first delay captures the interval starting from the time of posting a task to the time of receiving the first answer. (2) The second delay, i.e., inter-arrival time, models the interval from receiving adjacent answers of a task. Faradani et al. [39] use statistical models to predict worker's arrival rate in a crowdsourcing platform and characterize how workers select tasks from the platform. With the assistance of the statistical models, some works [129], [44] study latency control by considering whether or not a task can be accomplished within a time constraint, and evaluating the benefit of publishing a new task w.r.t. particular goals.

6.3 Latency vs. Quality vs. Cost

There is a trade-off between latency and cost. For example, in order to reduce cost, some cost-control techniques (e.g., answer detection) have to publish tasks in multiple rounds. However, increasing the number of rounds will lead to long latency. The similar trade-off also exists between latency and quality. For example, to increase quality, task assignment, a quality-control technique, assigns hard tasks to more workers and easy tasks to fewer workers. To do so, it needs to select tasks in multiple rounds to better understand the tasks. Thus, a large number of rounds can improve the quality but reduce the latency.

In order to balance the trade-off among quality, cost, and latency, existing studies focus on different problem settings, including optimizing the quality given a fixed cost, minimizing the cost with a little sacrifice of quality, reducing the latency given a fixed cost, minimizing the cost within latency and quality constraints, optimizing the quality without taking too much latency and cost, etc.

7 Crowdsourced Operators

There are many crowdsourced operators proposed for improving real-world applications. Various techniques are adopted to optimize the operator's trade-off among three factors: cost, quality and latency. In this section, we review how these operators can be implemented. Specifically, for each operator, we list its task type (Section 3.1), the optimization goal (cost, quality, or latency), and the techniques used to achieve the goal. Table 1 summarizes existing studies on crowdsourced operators.

7.1 Selection

Given a set of items, crowdsourced selection selects the items that satisfy some constraints. One example is to select the images that have both mountains and humans. Existing works on crowdsourced selection include three operators: Crowdsourced Filtering [89], [88], Crowdsourced Find [103], and Crowdsourced Search [129]. The difference is that their *targeted sizes* of returned results are different:

Crowdsourced Filtering [89], [88] (or All-Selection) returns *all* items that satisfy the given constraints;

Crowdsourced Find [103] (or k-Selection) returns *a predetermined number of (denoted as k)* items that satisfy the given constraints;

Crowdsourced Search [129] (or 1-Selection) returns *only one* item that satisfies the given constraints.

For example, given 100 images, suppose there are 20 images that have both mountains and humans. Crowdsourced Filtering selects all 20 targeted images, Crowdsourced Find requires to select a bounded number of (say, 5) targeted images, and Crowdsourced Search only selects one targeted image. Next we respectively introduce how existing works deal with these three crowdsourced operators.

7.1.1 Crowdsourced Filtering

Parameswaran et al. [89] first focus on a simplified crowdsourced filtering problem with only a single constraint (e.g., selecting the images that have mountains), and then extend their solution to multiple constraints. The task type used by their work is to ask the crowd whether or not an item satisfies a given constraint (e.g., "Does the following image contain mountains?"), and the possible answer is Yes or No. Their target is to study the trade-off between quality and cost. To achieve this goal, a *strategy* function is defined on each task. The strategy function takes the answers collected for each task (the number of Yes/No) as input, and outputs one of the following decisions: (1) the item satisfies the given constraint, (2) the item does not satisfy the given constraint, (3) the item should be asked again. For an item, the cost is defined as the number of workers answering its corresponding task and the quality is defined as the probability of correctly answering the item. Based on the strategy's definition, Parameswaran et al. [89] study how to find an optimal strategy that satisfy cost and error constraints:

Given an error threshold τ, find an optimal strategy that minimizes the expected cost w.r.t. the error constraint (expected error $< \tau$).

To solve the problem, Parameswaran et al. propose a brute-force algorithm for the problem, and then develop an efficient approximate algorithm that performs well in

TABLE 1
Crowdsourced Operators

Operators		Task Type	Goal	Techniques
Selection	Filtering [89], [88]	Single Choice	Quality	Answer Aggregation, Task Assignment
			Cost	Task Selection
	Find [103]	Single Choice	Quality	Answer Aggregation, Task Assignment
			Cost	Task Selection
			Latency	Round Model
	Search [129]	Single Choice	Quality	Answer Aggregation, Task Assignment
			Cost	Task Selection
			Latency	Statistical model
Collection	Enumeration [110]	Labeling	Quality	Answer Aggregation
			Cost	Miscellaneous (Pay-as-you-go Approach)
	Fill [93]	Labeling	Quality	Answer Aggregation
			Cost	Miscellaneous (Compensation Scheme)
			Latency	Pricing
Join	CrowdER [118]	Single Choice & Clustering	Quality	Worker Elimination, Answer Aggregation
			Cost	Pruning, Miscellaneous (Task Design)
	Transitivity [121], [116], [122]	Single Choice	Quality	Answer Aggregation, Task Assignment
			Cost	Pruning, Answer Deduction
	[45], [125]	Single Choice	Quality	Answer Aggregation, Task Assignment
			Cost	Task Selection
Topk/Sort	Heuristics-Based [49]	Single Choice	Quality	Answer Aggregation, Task Assignment
			Cost	Task Selection
	Machine Learning [23], [95]	Single Choice	Quality	Answer Aggregation, Task Assignment
			Cost	Task Selection
	Iterative Reduce [33], [49]	Single Choice	Quality	Answer Aggregation, Task Assignment
			Cost	Task Selection, Answer Deduction
	Heap-Based [27]	Single Choice	Quality	Answer Aggregation, Task Assignment
			Cost	Task Selection, Answer Deduction
	Hybrid [131], [67]	Single Choice & Rating	Quality	Answer Aggregation, Task Assignment
			Cost	Task Selection
Categorize	[91]	Single Choice	Quality	Answer Aggregation, Task Assignment
			Cost	Task Selection
Aggregation	Max [49], [113]	Single Choice	Quality	Answer Aggregation, Task Assignment
			Cost	Task Selection, Answer Deduction
	Count [78]	Single Choice, Labeling	Quality	Worker Elimination, Answer Aggregation
			Cost	Sampling, Miscellaneous (Task Design)
	Median [52]	Single Choice	Quality	Answer Aggregation
			Cost	Sampling
	Group By [27]	Single Choice	Quality	Answer Aggregation, Task Assignment
			Cost	Task Selection, Answer Deduction
Skyline	[75], [76]	Labeling	Quality	Answer Aggregation, Task Assignment
			Cost	Task Selection
	[47]	Single Choice	Quality	Answer Aggregation, Task Assignment
			Cost	Task Selection
Planning	CrowdPlanr [63], [77]	Labeling	Quality	Answer Aggregation, Task Assignment
			Cost	Task Selection, Answer Deduction
	Route Planning [134]	Single Choice	Quality	Answer Aggregation, Task Assignment
			Cost	Task Selection, Answer Deduction
	CrowdPlanner [105], [106]	Single Choice	Quality	Answer Aggregation, Task Assignment
			Cost	Task Selection
Schema Matching	[133], [86], [36]	Single Choice	Quality	Answer Aggregation, Task Assignment
			Cost	Task Selection, Answer Deduction
Mining	CrowdMiner [12], [13]	Labeling	Quality	Answer Aggregation, Task Assignment
			Cost	Task Selection
	OASSIS [10], [11]	Labeling	Quality	Answer Aggregation, Task Assignment
			Cost	Task Selection, Answer Deduction
Spatial	[109]	Labeling	Quality	Answer Aggregation, Task Assignment
			Cost	Task Selection

practice. They also extend the output of a strategy to a probability distribution (rather than a single choice), and study how to find an optimal probabilistic strategy. In this work [89], they make two assumptions: (1) all workers are of equal quality to answer a given task, and (2) all items are of equal difficulty to satisfy a given constraint. They relax these assumptions in a follow-up paper [88].

7.1.2 Crowdsourced Find

As Crowdsourced Find cares more about how fast k qualified items can be selected, Sarma et al. [103] mainly focus on the trade-off between cost and latency. Specifically, they define latency as the number of rounds (See Section 6.1) and cost as the number of tasks asked. The task type used is the same as that in Crowdsourced Filtering, i.e., to ask if an item satisfies a given constraint. Assuming that workers always give true answers, Sarma et al. study a sequential algorithm that asks one task at each round, and stops if k qualified items are observed. It is easy to see that the sequential algorithm requires the least cost. Using the sequential algorithm as a baseline, they further study how to improve its latency without increasing the cost:

For any given problem instance J, find an algorithm T such that (1) T correctly returns a solution and the cost is optimal (i.e., the cost is the same as the sequential algorithm on J); (2) no other algorithm has a lower latency than T that can correctly return a solution with optimal cost.

To address the problem, Sarma et al. develop an efficient

algorithm that selects tasks for the next round based on the answers collected from previous rounds. By releasing the requirement that an algorithm should have the optimal cost, they study other problem formulations with additive and approximation error bounds on the cost constraint.

7.1.3 Crowdsourced Search

Crowdsourced Search, which requires only one qualified item to be returned, is specifically studied in [129]. It focuses on a real-world application: search a target image in real time on mobile phones. For example, given a target image with a building, and a set of candidate images, the query is to search an image in the candidate images that contains the same building. Each task is to put a candidate image and the target image together, and ask workers whether the two images contain the same building. Workers will select Yes or No as the answer. Each search query has a deadline, and the objective of the query is defined as below:

Given a target image and a deadline, select at least one correct image from a set of candidate images before the deadline while minimizing the cost.

The above objective considers the trade-off among three factors: cost, quality and latency. The cost is defined as the number of answers collected. The quality is defined as the probability of correctly selecting one image out, where a majority-5 rule[3] is used to decide the result. The latency is predicted using a statistical model (see Section 6.2). Yan et al. [129] consider all these three factors, and design an efficient and effective algorithm to meet the objective.

7.2 Collection

Different from the above-mentioned selection operator which select items from a given set of *known* items (closed-world assumption), Crowdsourced Collection tries to collect *unknown* items from the crowd (open-world assumption). Typical Crowdsourced Collection operators contain Crowdsourced Enumeration [110] and Crowdsourced Fill [93]. The former asks the crowd to enumerate a list of objects that satisfy a constraint and the latter asks the crowd to fill missing values in a table.

7.2.1 Crowdsourced Enumeration

An example of crowdsourced enumeration is to find all states in the US, where each task with the description "Please name one or more states in the US." is assigned to workers. If two workers' answers are (Texas, Utah) and (Utah, Arizona) respectively, then the returned result will be (Texas, Utah, Arizona). As workers' answers are collected, an important problem is: *When is the result set complete?*, that is, to estimate when the result size is equal to the total number of states, which is unknown in advance. To solve the problem, Trushkowsky et al. [110] consider a similar problem studied in biology and statistics, called *Species Estimation* problem, which counts the number of unique species of animals on an island by putting out traps each night, and in the next morning, the collected animals are counted and released, and then the process is repeated daily. However, directly using the estimator (which estimates the total number of species in the species estimation) may not well capture or even contradict human behavior. One typical difference is that species estimation problem samples animals *with replacement*, while in crowdsourcing, each worker "samples" their answers *without replacement* from some underlying distribution. Moreover, the underlying distributions of different workers may vary a lot, and the estimator in the species estimation cannot capture the worker skew and arrival rates. That is, if a worker (called streaker) arrives and suddenly dominates the number of answers, then the result size will be overestimated.

Trushkowsky et al. [110] propose a new estimator to overcome the limitations. The estimator especially solves the overestimation issues caused by the existence of streakers. The way to find streakers is based on an observation that streakers give many unique answers (i.e., answers not provided by others). Trushkowsky et al. consider the effect of streakers in developing the new estimator. Furthermore, they propose a pay-as-you-go approach to balance the trade-off between cost and quality (See Section 5.5).

7.2.2 Crowdsourced Fill

Crowdsourced fill [93] provides workers with a partially-filled table and asks the workers to fill the missing values or update the existing wrong answers in the table. To balance the trade-off between cost, latency and quality, it targets at

Exploiting workers' desire to earn money in order to obtain a table with high quality, while without taking too much cost and latency.

To achieve the goal, Park and Widom [93] develop a system that works as follows: (1) each worker cannot only fill the table, but also upvote or downvote the values in a row; (2) requesters can set specific constraints in the values of the table (e.g., collecting any football player with position ="FW" and height$\geq$190cm); (3) the system can support real-time collaboration among different workers and handle concurrency well; (4) when a table is finalized, the system will allocate the total monetary award to all workers, and the award to each worker is proportional to the contributions made by the worker.

7.3 Join/Entity Resolution

Join is a very important operator in relational database systems. There are many different types of join operators, such as Cross join, Theta-join, and Outer join. Existing crowdsourcing works mainly focus on Equi-join. Given a table (or two tables), an equi-join is to find all *equal* record pairs in the table (or between the two tables). Note that when we say two records are *equal*, it does not only mean that they are identical records, but also means that they are different but refer to the same real-world entity (e.g., "iPhone Four" and "iPhone 4th Gen", or two different pictures of the same person). This problem is also known as entity resolution (ER), which has been studied for decades, but machine-based methods are still far from perfect. Therefore, applying crowdsourcing to solving this problem has recently attracted significant attention.

7.3.1 Background

In fact, incorporating humans into the ER process is not a new idea. As early as 2002, a user interactive ER approach

3. Each task is assiged to 5 workers at most, and Majority Voting is used to aggregate answers, i.e., the result is returned upon getting three consistent answers.

has been proposed [101]. However, what's new for crowdsourced ER is that *humans* can not be simply modeled as a few domain experts, who never make mistakes. Instead, there are hundreds of thousands of ordinary workers (i.e., the crowd) available. They may even provide worse results than machines (without a careful system design). In other words, both humans and machines can make mistakes, thus it is unclear if we can still combine them together such that the combined one is able to achieve better performance than human-only or machine-only alternatives. In 2012, two research groups [118], [28] answered this question independently by building entity-resolution systems on a real-world crowdsourcing platform (i.e., AMT [1]). Both of their systems validated that such goal could be achieved through a good workflow design and quality-control mechanism.

In the following, we will review the recent work on crowdsourced ER. A human-only solution is to ask the crowd to check all n^2 pairs of records. Even with a modest table size (e.g., $n = 100,000$), this method will be very expensive. Hence, most existing works adopt a two-phase framework, where in the first phase, a pruning method is used to generate a candidate set of matching pairs using an inexpensive machine-based technique, and in the second phase, the crowd are leveraged to check which candidate pairs are really matching.

7.3.2 Candidate Set Generation

For the candidate set generation phase, a similarity-based method is often adopted [118], [121], [125], [116], [71], [38], which computes a similarity value for each pair of records using a similarity function and takes the pairs whose similarity values are above a threshold as candidate pairs. The method has many advantages when being applied in practice. First, it only needs two input parameters (i.e., a similarity function and a threshold), which does not require a lot of human work for parameter tuning. Second, there are many similarity-join algorithms [120] proposed to efficiently find similar pairs that are above a threshold without enumerating all n^2 pairs.

If ER tasks become very complex, where a simple similarity-based method cannot generate a very good candidate set, people tend to manually specify domain-specific rules, called blocking rules. For example, the following blocking rule states that if the product names match but the color does not match, then the two products do not match:

$$\big(\text{name\_match} = \text{Y}\big) \wedge \big(\text{color\_match} = \text{N}\big) \longrightarrow \textsf{NO}.$$

Although the use of blocking rules can result in a more satisfactory candidate set, the design of high-quality rules often takes a lot of time. To overcome this limitation, Gokhale et al. [45] propose a crowd-based rule generation approach. Their basic idea is to think a blocking rule as a path in a decision tree, from root to one of the leaf nodes whose label is "NO". Based on this idea, they first apply an active-learning technique to learn a collection of decision trees from the crowd, and after that, they use the crowd again to check which paths in the decision trees (i.e., candidate blocking rules) make sense.

7.3.3 Candidate Set Verification

In the candidate set verification phase, the goal is to decide which pairs in the candidate set are really matching. Because we need to pay the crowd for doing tasks, existing works explore different perspectives to reduce crowd cost while still keeping good result quality, e.g., task design [79], [118], [126], leveraging transitive relations [121], [116], [48], [122], and task selection [45], [82], [125], [114], [59].

Task Design. Task design mainly involves two problems: user interface (UI) design and task generation.

The single-choice task type is one of the most widely used UIs for crowdsourced ER. It contains a pair of records and asks the crowd to choose "matching" or "non-matching" for the pair. Even for this simple UI, there are many different design choices [126], [79]. For example, we may want to provide a "maybe" option that the crowd can select when they are not sure about their answer, or we can put multiple pairs instead of just a single pair into a task, etc. In addition, the clustering task type has also been used by some works to reduce the cost [79], [118].

Once a particular UI is chosen, the next important question is how to generate tasks for the UI such that all the candidate pairs can be checked. This is a trivial problem for some types of UIs (e.g., single choice), but can be very tricky for others. For example, it has been shown that the cluster-based task generation problem, which aims to generate the minimum number of cluster-based tasks to check a set of candidate pairs, is NP-Hard [118].

Leveraging Transitive Relations. Leveraging transitive relations is another hot topic for candidate set verification. It is an answer-deduction technique discussed in the previous cost-control section. For entity resolution, there are two types of transitive relationships: (1) if $A = B$ and $B = C$, then $A = C$; (2) if $A = B$ and $B \neq C$, then $A \neq C$. After the crowd labels some pairs (e.g., $A = B$, $B \neq C$), we may be able to use the transitive relationships to deduce the labels of some other pairs (e.g., $A \neq C$), thereby saving crowd cost. However, this may not always be true. Consider the same three record pairs. If we label them in a different order (e.g., $A \neq C$ and $B \neq C$), we cannot use transitivity to deduce the label of (A, B), thus still requiring the crowd to label it. Therefore, one natural question is what is the optimal labeling order that can maximize the benefit of using transitivity.

In an ideal case, it is optimal to first present matching pairs to the crowd, and then present non-matching pairs [121]. But, this cannot be achieved in practice because it is unknown which pairs are matching or non-matching upfront. Hence, a heuristic approach [121] is proposed to approximate this ideal case, which first computes a similarity value for each candidate pair and then presents the candidate pairs to the crowd in a decreasing order of similarity values. This approach works very well when there exists a good similarity function to compute the similarity values. To handle the case that such similarity function does not exist, some other heuristic approaches have been proposed [116], which can provide a better worst-case performance guarantee than the similarity-based one.

Although transitivity is good for reducing cost, it may have some negative effects on quality and latency. In terms of quality, transitivity will amplify the impact of crowd errors on ER results. For example, suppose $A = B$ and $B = C$. If the crowd falsely label them as $A = B$ and $B \neq C$, such error will be propagated through transitivity, resulting in an erroneous label of $A \neq C$. Some ideas such as

using correlation clustering [122] or designing new decision functions [48] have been explored to tackle this issue. In terms of latency, in order to leverage transitivity, we cannot present all candidate pairs to the crowd at the same time. Instead, it should be an iterative process, where only a single pair or a few pairs can be presented to the crowd at each iteration. As a result, the iterative process may take much longer time to complete because it does not fully utilize all available crowd workers. This is considered as a main challenge in the use of transitivity for crowdsourced ER and various parallel algorithms are proposed to accelerate the process [121], [122].

Task Selection. Because the crowd is costly, sometimes it may be infeasible to ask them to check all candidate pairs. When there is only a limited amount of monetary budget for crowdsourcing, a natural idea is to explore how to select most valuable candidate pairs for the crowd to check. Existing works on this topic have different selection objectives: query-workload driven [59], ER-result driven [125], [114], and classifier driven [45], [82].

A query-workload driven approach assumes the existence of a query workload, and it aims to select those pairs such that when they are checked by the crowd, it will have the most benefit for the query workload. Inspired by the concept of the value of perfect information (VPI), Jeffrey et al. [59] develop a decision-theoretic framework in dataspace systems to achieve this goal.

Unlike the workload driven approach, an ER-result driven approach has a different optimization goal. It aims to select those pairs that are most valuable to the ER result quality. The approach often models candidate pairs as a probabilistic graph, where each edge in the graph represents a candidate pair and the probability of each edge represents the probability that the corresponding candidate pair is a matching pair. The probabilistic graph is used to measure the quality of the current ER results and then predicate which pairs should be selected that can maximize the improvement of the quality [125], [114].

A classifier-driven approach uses the crowd to train a classifier. Its goal is to select the candidate pairs that are most beneficial to the training process of the classifier. According to different characteristics of the classifiers, they have different selection strategies. The random forest classifier selects the pairs based on the percentage of decision trees that have contradictory classification results [45]; the SVM classifier uses the distance of each pair to the decision hyperplane to decide which pairs to choose [82].

7.4 Top-k and Sort

Given a set of items $\mathcal{O} = \{o_1, o_2, \cdots, o_n\}$, where the items are comparable but hard to compare by machines, crowdsourced top-k aims to find a k-size item set $\mathcal{R} = \{o_1, o_2, \cdots, o_k\}$ where o_i is preferred to o_j (denoted by $o_i \succ o_j$) for $o_i \in \mathcal{R}$ and $o_j \in \mathcal{O} - \mathcal{R}$, and crowdsourced sort aims to sort the items and gets a sorted list $o_1 \succ o_2 \succ \cdots \succ o_n$. Zhang et al. [135] give an experimental survey on this problem.

7.4.1 Workflow

To utilize the crowd to find top-k items, we need to generate crowdsourced tasks. There are two widely-used ways to generate crowdsourced tasks. The first is single choice, which selects two items and asks the crowd to select the preferred one (a.k.a., pairwise comparison). The second is rating, which selects multiple items and asks the crowd to assign a rate to each item. The rating-based method has some weaknesses. First, the crowd prefers pairwise comparisons than ratings as the former is much easier. Second, it is rather hard for the crowd to assign an accurate rate, and items in different rating groups may not be correctly compared. Thus the rating-based method usually has a lower accuracy than pairwise comparison [67], [79]. Most of existing works use pairwise comparisons. Next we introduce the pairwise comparison based framework.

7.4.2 Pairwise Comparisons

Pairwise comparison methods use the single-choice task type, where each task is to compare two items. To reduce the monetary cost, existing methods employ a task selection strategy, where b pairs are crowdsourced in each round. Based on the comparison answers of the crowdsourced pairs, it decides how to select b pairs in next round. To reduce crowd errors, each task is assigned to multiple workers and the final result is aggregated based on the answers of these workers, e.g., weighted majority vote. To model the results, a directed graph is constructed where nodes are items and edges are aggregated comparison answers. For each pair (o_i, o_j), if the aggregated result is $o_i \succ o_j$, there is a directed edge from o_i to o_j where the weight is the aggregated preference.

7.4.3 Result Inference

Given the answers for the generated tasks, the Result Inference tries to infer the query results, i.e., top-k items or a sorted list. There are five types of methods: (1) score-based methods, (2) iterative reduce methods, (3) machine learning methods, (4) heap-based methods, and (5) hybrid methods.

(1) Score-Based Methods

Guo et al. [49] prove that finding the top-1 item with the largest probability is NP-Hard by a reduction from Kemeny rankings. Thus inferring the top-k items is also NP-Hard and some score-based algorithms are proposed, which assign each item o_i with a score s_i and select the k items with the largest scores as top-k results (or sort them based on the scores). Next we discuss how to assign scores.

BordaCount [6]. The score of item o_i is its out-degree (i.e., the number of wins compared with its out-neighbors).

Copeland [96]. The score of item o_i is its out-degree minus its in-degree (i.e., the number of wins minus the number of losses).

Local [49]. The above two methods only consider the neighbors of each item and cannot capture more information. Local top-k algorithm [49] is proposed to address this problem by considering 2-hop neighbors. Obviously if an item has more 2-hop out-neighbors (i.e., its out-neighbors' out-neighbors), the item will beat more items (based on transitivity), and thus the item has a larger score. Similarly, if an item has more 2-hop in-neighbors (i.e., its in-neighbors' in-neighbors), the item will be beaten by more items, and thus the item has a lower score.

Indegree [49]. It computes the score based on the Bayesian model. It first computes the probability of $o_i \succ o_j$ given

the aggregated preference, and then uses the probability to compute the score.

Modified PageRank(MPageRank) [49]. It extends the original PageRank by considering the crowdsourced comparisons and computes the score of each item.

RankCentrality [85]. It uses the random walk to compute the score of each item.

ELO [32]. It is a chess ranking system and can be used to compute the score s_i. The basic idea is that, if item o_i with higher ranking beats another lower one o_j, only a few scores will be added to s_i; on the contrary, if o_j wins, a lot of scores will be added to s_j.

Balanced Rank Estimation(BRE)/Unbalanced Rank Estimation(URE) [123]. The score is computed based on the probability theory. The balanced rank estimation (BRE) considers both incoming and outgoing edges to compute the score, which computes the relative difference of the number of items proceeding and succeeding it. The unbalanced rank estimation (URE) only considers the number of items proceeding o_i to compute the score of o_i.

(2) Iterative Reduce Methods

The iterative-reduce methods adaptively eliminate the low rank items that have small possibilities in the top-k results, until k items left.

Iterative [49]. It first utilizes the score-based methods to compute the scores of each item and then removes a half of items with the smallest scores. Next it re-computes the scores on the survived items and repeats the iterations until k items left.

PathRank [33]. The main idea of PathRank is to perform a "reverse" depth first search (DFS) for each node, which traverses the graph by visiting the in-neighbors of each node. If it finds a path with length larger than k, it eliminates the item as k items have already been better than the item.

AdaptiveReduce [33]. Initially there are n items. Then it selects an informative set and utilizes the set to eliminate items with small possibilities in the top-k answers. It repeats this step using the survived items until finding top-k items.

Bound-Based Methods [21]. The bound-based methods build a stochastic matrix, and use several algorithms to identify the top-k results based on the matrix: i) *Sampling Strategy with Copeland's Ranking (SSCO).* It selects the top-k rows with most entries above 0.5 as the top-k results. ii) *Sampling Strategy with Sum of Expectations (SSSE).* It selects the top-k rows with the largest average value as the top-k results. iii) *Sampling Strategy based on Random Walk (SSRW).* It first computes a stochastic matrix and then derives its principal eigenvectors (that belong to the eigenvalue 1). Then it identifies the top-k rows with the largest eigenvalues as the top-k answers.

(3) Machine Learning Methods

These methods assume that each item has a latent score which follows a certain distribution. Then they utilize machine learning techniques to estimate the score. Finally, they use the latent scores to sort the items or get top-k items.

CrowdBT with Bradley-Terry Model [20]. The Bradley-Terry (BT) model can be used to estimate the latent score [20]. In the BT model, the probability of $o_i \succ o_j$ is assumed as $\frac{e^{s_i}}{e^{s_i}+e^{s_j}}$. Then based on the crowdsourced comparisons, it computes the latent scores by maximizing $\sum_{o_i \succ o_j \in \mathcal{L}} \log(\frac{e^{s_i}}{e^{s_i}+e^{s_j}})$, where $\mathcal{L}$ is a set of crowdsourced comparison answers. But the BT model does not consider the workers' qualities. To address it, Chen et al. [23] propose the CrowdBT model, assuming that each worker has a quality η_w as discussed in Worker Probability (Section 4).

CrowdGauss with Gaussian Model [95]. It assumes that the score follows the Gaussian distribution, where the score is the mean of the distribution. The probability of $o_i \succ o_j$, i.e., $P(o_i \succ o_j)$, can be computed by the cumulative distribution function (Φ) of the two standard Gaussian distributions, i.e., $P(o_i \succ o_j) = \Phi(s_i - s_j)$. Then CrowdGauss computes the scores by maximizing $\sum_{o_i \succ o_j \in \mathcal{L}} M_{ij} \cdot \log(\Phi(s_i - s_j))$. where M_{ij} is the number of workers reporting $o_i \succ o_j$.

HodgeRank [60]. In order to estimate a global order for n items, HodgeRank [60] utilizes a matrix decomposition based techniques to compute the score.

TrueSkill [53]. TrueSkill improves ELO by reducing the repeated times as ELO needs to repeat many times to convergence. Different from ELO, the score of each item o_i is represented by a Gaussian distribution $N(s_i, \delta_i)$, where s_i is the estimated score for o_i and δ_i is the deviation of s_i. For each crowdsourced answer $o_i \succ o_j$, it updates the scores and deviations.

(4) Heap-Based Methods

TwoStageHeap [27]. In the first phase, the items are divided into $\frac{n}{X}$ buckets (where $X = \frac{xn}{k^2}$) such that the probability of two top-k items appearing in the same bucket is at most x. In each bucket, a tournament based max algorithm [27] is conducted to select the best item in the bucket. Each pair on top levels of the tournament is compared multiple times and each pair on low levels of the tournament is compared only once. The second phase utilizes a heap-based method [40] to identify the top-k results from these best items. To tolerate errors, when constructing and re-heapifying the heap, each pair is compared by multiple workers and the algorithm uses the majority voting to obtain a combined preference. After popping an item from the heap, the algorithm asks next pairs following the re-heapifying order.

(5) Hybrid Methods

There are two algorithms [131], [67] that combine rating and comparison tasks. They utilize the rating and comparison answers to learn the score. For rating, they pre-define τ categories and each category χ_c has a range $(\gamma_{c-1}, \gamma_c]$, where $\gamma_0 < \gamma_1 < \cdots < \gamma_\tau$. If the score of item o_i falls in $(\gamma_{c-1}, \gamma_c]$, o_i is in category χ_c.

Combine [131]. It first selects some rating and comparison tasks, and then infers the scores based on these results. The score for each item o_i is modeled by $s_i + \varepsilon_i$, where $\varepsilon_i \sim N(0, \delta^2)$, which is utilized to tolerate crowd errors. For rating, it computes the probability of o_i being in the category χ_c by standard Gaussian distribution. For comparison, it constructs the comparison matrix M and computes the probability of observing M based on the Bayesian theory. Then it combines rating and comparison to infer the results.

Hybrid [67]. It first crowdsources all rating tasks and then selects some candidate items with higher ratings. Next it chooses some pairs from the candidates as comparison tasks. The score of each item o_i is modeled as a normal variable $N(s_i, \delta^2)$. Given the rating results E_R and comparison answers E_C, it assumes that these results are gotten independently, and computes the scores based on the results.

However, the maximum likelihood estimation is rather expensive, thus a modified PageRank approximation [67] is proposed to estimate the score for each item.

7.4.4 Task Selection

The Task Selection decides which candidate pair of items should be asked next. There are three types of methods: (1) heuristic-based methods, (2) bound-based methods, and (3) active learning methods.

(1) Heuristic-Based Methods

Guo et al. [49] prove that selecting the pairs to maximize the probability of obtaining the top-k results given the comparison answers of arbitrary crowdsourced pairs is NP-Hard and propose four heuristics, which are designed for selecting max (top-1) result. The algorithms first compute a score s_i for item o_i as discussed above. Suppose the sorted items based on the scores are $o_1, o_2, \cdots, o_n$. Then, the algorithms select the next b pairs as follows.

Max. It selects b pairs: (o_1, o_2), (o_1, o_3), $\cdots$, (o_1, o_{b+1}).

Group. It groups the i-th item with the $(i+1)$-th item and selects (o_1, o_2), (o_3, o_4), $\cdots$, (o_{2b-1}, o_{2b}).

Greedy. It selects the pairs based on $s_i \times s_j$ in descending order, and selects b pairs with the largest value.

Complete. It first selects x items with the highest scores, where x is the largest number satisfying $\frac{x*(x+1)}{2} \leq b$. The selected pairs include two parts. The first part includes all $\binom{x}{2}$ pairs among these x items. The second part contains $(o_1, o_{x+1}), (o_2, o_{x+1}), \cdots, (o_{b-\frac{x*(x+1)}{2}}, o_{x+1})$.

(2) Bound-Based Methods

SSCO and SSSE [21] estimate a bound for each pair and utilize the bound to select next pairs. They first compute a confidence interval $[l_{ij}, u_{ij}]$, where l_{ij} (u_{ij}) is the lower (upper) bound of the probability of $o_i \succ o_j$. Based on the confidence interval, they select a set S and discard a set D. Since the relationships between pairs in $S \cup D$ have been effectively captured, these pairs do not need to be compared. Thus they select pairs that are not in $S \cup D$. In comparison, SSRW [21] selects the next pairs by random walk on the stochastic matrix.

(3) Active Learning Methods

CrowdGauss [95]. The scores can be modeled by a multivariate Gaussian distribution $N(\hat{s}, \mathcal{C})$, where $\hat{s}$ is a $1 \times n$ matrix indicating the score for all the items (initialized by random values), and $\mathcal{C}$ is the covariance matrix of $\hat{s}$ ($\mathcal{C}_{ij}$ is the value at i-th row and j-th column). In each round of pair selection, the expected information gain for each pair (o_i, o_j) is computed, and it selects the pair with the largest expected information gain and updates $\hat{s}$ and $\mathcal{C}$.

CrowdBT [23]. The above method does not consider the worker quality. Chen et al. [23] propose an active learning method by taking into account the worker quality. The score of item o_i is modeled by a Gaussian distribution $N(s_i, \delta_i)$ and the quality of worker w is modeled by Worker Probability (see Section 4).

Combine [131]. Instead of just utilizing pairwise comparisons, Ye et al. [131] propose an active-learning strategy by combining rating and comparison together. Given a budget, it selects some rating-based questions and comparison-based questions to maximize the expected information gain.

7.5 Aggregation

Aggregation queries are widely used in data analysis. We will first describe how to use the crowd to deal with various aggregation functions, and then discuss how to support the Group By clause.

7.5.1 Max

Crowdsourced max is a problem that finds the *max* item in a dataset. The item comparison is often based on human subjectivity. For example, finding the most beautiful picture about the Golden Gate Bridge; returning the best Chinese restaurant in San Francisco. There are two types of solutions to crowdsourced max: structured [113] and unstructured [49].

The structured solution generates tasks in a predefined structure. It is guaranteed that after tasks are finished using the structure, the max item can be directly deduced based on transitivity (i.e., if A is better than B, and B is better than C, then A is better than C). Venetis et al. [113] propose two max algorithms based on this idea: (1) The bubble max algorithm is derived from the well-known bubble sort algorithm. It first asks the crowd to compare (A, B). Suppose A is better than B. Then, it asks the crowd to compare the better one (i.e. A) with C. Similarly, after the crowd compare all $n-1$ pairs, the final item will be considered as the best one. (2) The tournament max algorithm is a tournament-like algorithm that is widely used in sports for choosing the best team. It first generates $\frac{n}{2}$ pairs, i.e., (A, B), (C, D), (E, F), etc, and asks the crowd to compare them. Then, it chooses the better one of each pair and generates $\frac{n}{4}$ new pairs. Iteratively, when there is only one item left, it will be considered as the best one. It has been shown that the tournament max algorithm performs better than the bubble max algorithm. To further improve the performance, Venetis et al. also develop a hill climbing based approach to adaptively tune two parameters used in the max algorithms.

Unlike the structured solution, the unstructured solution generates tasks in a dynamic way. It utilizes the results of finished tasks to dynamically decide which tasks should be generated next. Guo et al. [49] identify two main problems (Result Inference and Task Selection) in this setting and formalize them based on Maximum Likelihood. See the detailed solution to these problems in Section 7.4.

7.5.2 Count

Crowdsourced count is a problem that counts the number of items in a dataset that satisfy some given constraints. At first glance, this is quite similar to crowdsourced filtering. But, the difference is that the crowdsourced count does not require returning the satisfied items but only a number, thus there are some interesting new ideas.

The first idea is to use sampling [78]. Instead of checking all the items in the entire dataset, this idea only checks the items in a random sample. Sample estimation is a well studied topic in statistics. Let $|D|$ be the data size, $|S|$ be the sample size, and k be the number of the items in the sample that satisfy the constraints. The count w.r.t the full data can be estimated from the sample as $k \cdot \frac{|D|}{|S|}$. This is an unbiased estimate, and the error of the estimate decreases proportional to $\frac{1}{\sqrt{|S|}}$.

In addition to sampling, another interesting idea is a new user interface, called *count-based interface* [78]. Recall that crowdsourced filtering mainly uses a *single-choice interface*, which asks the crowd to choose "Yes" or "No" for an item, indicating whether the item satisfies the constraints. In comparison, a count-based interface shows a small batch of items (e.g., ten photos) to the crowd and asks them to approximately estimate the number of items that satisfy the constraints, e.g., counting how many of the ten photos contain mountains. Interestingly, the two interfaces are suitable for different situations. The count-based interface is more suitable for images, but the single-choice interface performs better for text. This might be because that humans are better at processing batches of images than strings of text.

A slight variant of the counting problem is to count the number of objects in a *single* photo (e.g., how many persons are there in one photo?). In this problem, a new challenge is how to split the photo into small segments such that each segment only contains a small number of objects. The reason for doing that is that humans are often good at batch counting, but the batch size cannot be very large. For example, showing a photo with hundreds of people to a worker to count will lead to long waiting time and low-quality answers. To address this problem, [102] explores two scenarios. One assumes that there is no good computer vision approach available that can help to identify the objects in the photo and a top-down algorithm is proposed to generate image segments; the other is a bottom-up approach which first uses a computer vision approach to identify the objects in the photo, and merges the image segments containing an identified object into bigger ones.

7.5.3 Median

A median operator aims to find the centroid in a set of items. It is a key operation in various clustering algorithms (e.g., k-means). One simple crowdsourced implementation is to apply a crowdsourced sort operator to the set of the items and then return the middle item in the sorted list. Because the median operator only needs to return the centroid, the expensive sorting process can actually be avoided using the crowd-median algorithm [52]. The algorithm presents a task with three items to the crowd and asks them to pick the item (called outlier) that is different from the other two. If the underlying data follows a univariate normal distribution, the centroid often has the highest probability of being an outlier. Thus, the problem is reduced to finding the item with the highest probability. Note that the algorithm does not enumerate all $\binom{n}{3}$ possible tasks to compute the probabilities, and instead it uses sampling to estimate the probability for each item.

7.5.4 Group By

A crowdsourced group-by operator can group a set of items based on their unknown type. It is often used with aggregation functions (e.g., COUNT, MAX) to enable more advanced data analysis. For example, the following group-by aggregation query

```
SELECT BEST-LOOKING(photo) FROM table
GROUP BY PERSON(photo)
```

will find the best looking photo of each individual person. In the database community, prior work [27] assumes that there is an underlying ground truth for the type of each item. For example, the ground truth of the above query is a person's name. Based on this assumption, they can derive good theoretical bounds on the minimum number of tasks that are required to ask for crowds. In the machine learning community, there are some works that do not need to make this assumption. Their basic idea is to treat group-by as a clustering problem and to study how to incorporate crowdsourcing into existing clustering algorithms [132], [46], [52].

7.6 Categorize

Given a set of categories and a set of uncategorized (i.e., unlabeled) objects, the object categorization problem aims to ask the crowd to find the most suitable category (i.e., label) for each object [91]. Each task is a single choice question. Given an object and a category, it asks if the object belongs to the category.

There are three dimensions that characterize the different instances of the category problem. Dimension 1: Single or Multiple target categories. In the single case, each object has a single target category; while in the multiple case, each object has multiple target categories. Dimension 2: Bounded or Unlimited number of tasks. In the bounded case, a budget is given and only a number of tasks can be asked. In this case, it aims to find the category as accurately as possible within the budget. In the unlimited case, it aims to ask the minimal number of questions to precisely identify the target categories. Dimension 3: Tree (or forest, which is a set of trees) or directed acyclic graph (DAG) . The categories are not independent and can be organized by a tree or DAG.

Parameswaran et al. [91] give the complexities of the category problems for all combinations. They further prove that the problems with the DAG model are NP-hard and the problems with the tree model can be solved in polynomial time. For the DAG model, they design brute-force algorithms to find the optimal solutions; and for the tree model, they propose dynamic programming algorithms.

7.7 Skyline

Given a collection of items, a skyline operator aims to find all the items in the collection that are not *dominated* by others. Here, each item has n attributes, and we say an item o_1 is dominated by another item o_2 if and only if o_1 is not as good as o_2 for any attribute. For example, suppose we want to find the best hotel in a city by considering three attributes: price, distance, and rating. There might be a lot of hotels in the city. Using a skyline operator, we can remove those hotels that cannot be our optimal choice.

Machine-based skyline algorithms have been extensively studied for more than a decade [19]. However, there are still some limitations that are not easy to overcome without human involvement. For example, when data has missing values, it is difficult to obtain high-quality skyline results without asking humans to fill those missing values; when data contains some attributes that are hard to compare (e.g., deciding which photo looks more beautiful), we need to ask humans to make such comparisons. These limitations motivate two research directions for crowdsourced skyline.

Skyline on Incomplete Data. One direction is to study how to leverage crowdsourcing to obtain high-quality skyline answers from incomplete data. It is expensive to ask the

crowd to fill all missing values. In fact, some missing values are not necessary to be filled. For example, suppose we want to know whether an item A dominates another item B. If we can derive from incomplete data that B has one attribute that is better than A, then we will know that A does not dominate B, without filling any missing value. Existing works [75], [76] adopt a hybrid human-machine workflow, which first uses machines to fill all missing values in a heuristic way, and then asks the crowd to examine those items that are most beneficial to improve the skyline quality.

Skyline with Crowdsourced Comparisons. Another idea that incorporates crowdsourcing into the skyline computation is to use the crowd to compare those challenging attributes. Because the crowd are very likely to make mistakes, the skyline algorithm has to handle noisy comparison answers. [47] studies how many comparisons are required to return the correct skyline with a high probability.

7.8 Planning

Crowdsourced planning query [63] aims to find an ordered sequence of items. For example, when a tourist visits a new city, s/he wants to travel all the scenic spots in that city. A better plan should consider the distance between scenic spots, and the best visiting time and duration of each scenic spot. Different from traditional planning query which is automatically solved by computers [83], crowdsourced planning query is more difficult in terms of involved computational complexity (e.g., finding the best plan by considering many factors) and hard-to-understand goals (e.g., satisfying the personalized requirement).

The crowdsourced planning query is formally defined [63] as below.

Given a large set of items, the target is to choose a subset of items, and then order the chosen items in a sequence (called "plan") with the best quality.

For example, considering visiting four scenic spots (Forbidden City, Summer Palace, Great Wall, Olympic Park) in Beijing, a good plan is ⟨Great Wall, Summer Palace, Olympic Park, Forbidden City⟩. The set of all possible plans can be modeled as a tree, and a plan is a root-to-leaf path. The tree can be incrementally built by asking workers to do some tasks, e.g., "given the sequence ⟨Great Wall, Summer Palace⟩, what is the next scenic spot that I should visit?". Each edge in the tree has a score, which indicates the confidence of the edge derived from workers' answers. The quality of a plan is defined as the product of the scores of all the edges in the plan. The target of the problem is to derive the plan with the best quality. To balance cost and quality, Kaplan et al. [63] formally define that a plan is a *correct answer* to the query if the quality of the plan is close to (within ϵ) the best quality. Then, they develop an algorithm to address the problem of which task to ask next and when to stop asking tasks. Finally, they prove that the developed algorithm is instance-optimal [34] w.r.t. the cost.

How crowdsourcing can assist artificial intelligence in addressing planning query is studied in [141], [108]. A specific planning application: route planning is studied in [134], [105], and they both focus on the cost and quality trade-off. Given the candidate routes generated from existing route recommendation systems, Zhang et al. [134] study how to select tasks to reduce uncertainty in the routes. Each route has a probability to be the best route. They use the entropy of all the routes to characterize the uncertainty. By carefully selecting tasks, where a task can be "Starting from A, following which direction in (C, D, E) that I can get to B?", the uncertainty can be reduced and the best route is easier to be selected. Su et al. [105] take candidate routes as input, and select significant locations (called "landmarks") to distinguish different routes. An example task is "Do you prefer the route passing landmark A at 2:00 pm?". They study two main problems in this setting: task generation (how to automatically generate user-friendly tasks) and worker selection (how to choose a set of suitable works for a given task), to get a good trade-off between cost and quality.

7.9 Schema Matching

Schema matching aims to find the attribute mappings between different database schemas. For example, given two relational tables: Table *Professor* with attributes *(Name, Phone)* and Table *ProfInfo* with attributes *(Profname, Fax, Tel)*, the target is to find a correct matching between the attributes in the two tables. A possible matching is {(*Professor.Name, ProfInfo.Profname*), (*Professor.Phone, ProfInfo.Tel*)}. Existing works [98] develop automatic algorithms to generate a set of possible matchings, by considering linguistic and structural information. However, it is rather hard to remove the ambiguity using machine-only approaches. Thus recent works [133], [86], [36] leverage human insights (or crowdsourcing) to reduce the uncertainty in schema matching.

Given two schemas, Zhang et al. [133] propose a hybrid (human-machine) approach to address the schema matching problem. Specifically, they first use machine-based approaches to generate a set of possible matchings, where each matching (containing a set of correspondences) has a probability to be the correct matching. Then they define the uncertainty in schema matching as the entropy of all possible matchings. To reduce the uncertainty, they generate some tasks for the crowd, where each task is to ask whether or not a given correspondence is a correct matching. For example, "Is the correspondence (*Professor.Name, ProfInfo.Profname*) correct?". In order to balance cost and quality, they develop effective task selection techniques, with the target of reducing the most uncertainty of schema matching with the lowest cost.

Rather than matching over two schemas, Hung et al. [86] consider a setting where the matching is conducted in a network of multiple schemas. To reduce the uncertainty of the whole schema network, they first build a probabilistic matching network, where each attribute correspondence is associated with a probability. The built network conforms to some specified integrity constraints, e.g., the 1-to-1 constraint, which regulates that each attribute of one schema can be matched to at most one attribute of another schema. The uncertainty of the schema network is defined as the entropy of a set of random variables, where each one denotes whether a given correspondence exists in the correct matching. Then the target is to select correspondences to ask the crowd, such that the uncertainty of the schema network is reduced the most. They also study how to approximate a correct matching based on workers' answers, i.e., deciding whether or not each correspondence is correct by considering the specified integrity constraints.

Different from relational tables which give relatively complete schema information, Ju et al. [36] study how to match web tables, which are rather dirty and incomplete. They leverage the power of knowledge bases, which can offer concepts with wide coverage in a high accuracy. Their approach first maps the values in each table column to one or more concepts, and the columns (in different tables) that represent the same concept are matched with each other. Then, they adopt a hybrid approach: machines do the easy work of matching pairs of columns based on the concepts, while the crowd will discern the concepts for the columns that are regarded as difficult by machines. Each task presents workers with values in a column as well as a set of candidate concepts, then workers will decide which is the best concept for the column. In terms of task selection, a utility function is defined over columns, considering the factors such as the columns' difficulties, and effective solutions are proposed next, targeting at selecting which columns to ask so that the expected utility is maximized.

7.10 Mining

Crowd Mining tries to mine significant patterns from workers' behaviors. In this section, we first present crowd mining formulation [12] and complexity [8]. Then we discuss two studies on leveraging ontology to assist crowd mining [10], [14]. Finally, a generic mining architecture is described [9].

7.10.1 Formulation and Complexity

To capture significant patterns, association rule [7] in data mining is widely used, where an association rule, denoted as $r : A \rightarrow B$ indicates that A implies B. An example rule r : flu→garlic means that if having flu (a disease), then taking garlic (a treatment). Traditional approaches in association rule mining, however, cannot be directly applied to crowdsourcing settings, as the traditional approaches depend on the fact that the rules are mined from a database with transactions (e.g., list of treatments for each disease). But, in a crowdsourcing environment, it is hard to ask a worker to provide extensive transactions (e.g., telling all of her/his past treatments for each disease). To address the issue, Amsterdamer et al. [12] point out that social studies have shown that although people cannot recall all treatments in their transactions, they can provide simple summaries, e.g., they may know "When I have flu, most of the times I will take garlic.". So the problem is defined as

How to aggregate simple summaries from crowd workers and find the overall significant rules?

There are two parameters defined on a rule $r : A \rightarrow B$, support and confidence, where the support indicates the probability that A and B occur together, and the confidence indicates that given A occurs the conditional probability that B occurs. Given a rule r, its support and confidence can be answered by workers. For example, a task asking the support of r is "How often do you have flu and use garlic as the treatment?" and a task asking its confidence is "When you have the flu, how often do you take garlic?" Furthermore, there is another task type: open task, which asks the crowd to provide a possible rule with support and confidence. For example, an example task is "Tell about an illness, the way you treat it and how often they both occur." Amsterdamer et al. study three problems in order to balance quality and cost: (1) the aggregation problem: how to compute the significant rules based on workers' answers, (2) the assignment problem: which rule should be chosen as the next task, and (3) the trade-off between asking open tasks to obtain possibly new information, and asking tasks of existing rules to improve estimation.

Following the crowd mining definition, Amarilli et al. [8] study the complexity of frequent itemset mining. The frequent itemset mining is to mine the combination of items that are frequent enough in crowd's behaviors. For example, a task trying to find whether or not sport and tennis are frequent items for a worker will ask "Do you usually play tennis as a sport?", and a possible answer of Yes or No will be reported by the worker. Amarilli et al. consider two complexities: (1) Crowd Complexity, which measures the number of tasks that need to be asked in order to compute all frequent itemsets; (2) Computational Complexity, which measures the computational effort required to choose the tasks. They theoretically study the two complexities under the defined model.

7.10.2 Leveraging Ontology

Based on [12], a system called OASSIS is then developed [10], which targets at

Enabling requesters to pose general queries, such that the relevant answers representing frequent patterns can be returned.

An example query to the system is "Finding popular combinations of an activity in a child-friendly attraction in NYC and a restaurant nearby." OASSIS combines the ontology (general knowledge) and the crowd (personal knowledge) together. The ontology contains a number of triples (called facts), and an example fact (Central Park, inside, NYC) indicates that Central Park is inside NYC. The crowd is asked with two task types: (1) concrete task asks the frequency of a combination of facts from a worker, e.g., "How often do you go biking in Central Park and eat Falafel at Maoz Veg?" (2) specification task asks the crowd to give more facts, e.g., "Other than biking, what type of sport do you also do in Central Park and how often do you do that?" OASSIS uses the ontology to reduce the number of tasks. For example, if doing sports in Central Park is regarded as infrequent, then it is not worth asking the crowd the frequency of riding bicycle in Central Park. OASSIS contains a query language called OASSIS-QL (based on the SPARQL), and it asks requesters to formulate the query using the provided tools.

However, OASSIS-QL is too hard for most requesters without technical backgrounds. Based on this observation, Amsterdamer et al. [14] further consider the problem of how to translate the Natural Language (NL) query to a formal, declarative OASSIS-QL query. Given an NL query, e.g., "What are the most interesting places near the Forest Hotel, Buffalo, we should visit in the fall?" Their approach decomposes the NL query into two parts: the general knowledge part, which can be matched to ontology (in the example, it corresponds to "What places are near Forest Hotel, Buffalo?") and the individual knowledge part, which can express personal knowledge (in the example, it corresponds to "interesting places" and "we should visit in the fall"). Then, it translates the general knowledge part (using existing General Query Generators) and the individual knowledge part (using self-defined Individual Expressions

tools) to corresponding sub-queries. Finally a combination tool is designed to construct a formal OASSIS-QL query by combining the two generated sub-queries.

7.10.3 A Generic Mining Architecture

A generic crowd mining architecture is described in [9], which summarizes six important components: (1) *Data Repositories*, which store external knowledge (e.g., ontology) and the crowd's answers; (2) *Query Engine*, which aims at computing a query plan that can minimize the crowd's effort and machine's execution time; (3) *Crowd Task Manager*, which processes the crowd's answers and decides the tasks that will be assigned to coming workers; (4) *Inference and Summarization*, which aggregates the crowd's answers with the assistance of external knowledge (e.g., ontology); (5) *Crowd Selection*, which selects appropriate workers to assign tasks; (6) *NL Parser/Generator*, which translates the natural language (NL) given by a requester to a formal declarative query (e.g., SPARQL query).

7.11 Spatial Crowdsourcing

Many crowdsourced tasks contain spatial information, e.g., labeling a restaurant and taking a photo in a specified location. Spatial crowdsourcing has a significant difference from other operators: the workers can answer most of the tasks in other operators while workers can only answer some spatial tasks whose locations are close to the workers. Thus most of studies focus on how to effectively assign the tasks to appropriate workers based on spatial proximity [65], [66], [30], [111], [97], [109], [106], [25], [16], [134], [56], [57].

7.11.1 Spatial Crowdsourcing with Euclidean Space

In euclidean space, the locations of both spatial tasks and workers are represented by geo-coordinates with longitudes and latitudes. Spatial tasks can be assigned to workers with two different modes [65]: (1) Worker Selection Model, where the spatial crowdsourcing server (SC-server) publicly publishes the tasks on a crowdsourcing platform, and online workers can voluntarily choose the nearby spatial tasks; (2) Server Assignment Model, where all online workers send their locations to the SC-server, and the SC-server assigns the tasks to workers.

(1) Worker Selection Model

The advantage of the worker selection task mode is its simplicity and that it is easy to implement. Deng et al. [30] study the *maximum task scheduling* problem, which aims to recommend a longest valid sequence of tasks for a worker. Given a worker and a set of tasks where each task has an expiration time, a worker is asked to do a sequence of tasks. To be specific, after finishing one task, the worker is required to travel to the location of the next task, which may incur some time cost. The goal is to find the longest valid sequence for the worker that all the tasks in the sequence are completed within the expiration time. The problem is proved to be NP-hard, and a dynamic-programming algorithm is proposed to find the exact optimal answer with time complexity of $\mathcal{O}(2^n \cdot n^2)$, where n is the number of available tasks. Since the exact algorithms cannot scale to large number of tasks, effective approximation algorithms are proposed by greedily picking the tasks with the least expiration time or the smallest distance.

However, this model has some limitations. First, the SC-server does not have any control over tasks and workers, which may result in the case that some spatial tasks will never be assigned, while others are assigned redundantly. Second, a single worker does not have a global knowledge of all the tasks, and thus tasks are blindly selected without considering the overall travel cost and task expiration time.

(2) Server Assignment Model

The server assignment model overcomes the limitations of worker selection model, which assigns tasks to nearby workers while optimizing overall objectives.

Maximize the Number of Assigned Tasks [65]. Kazemi et al. [65] assume that every worker has a spatial region and can accept at most T tasks within the region. For each time instance t, it runs an assignment algorithm to assign available tasks to workers under above constraints. Given a time interval consisted of several continuous time instances $t_1, t_2, \cdots, t_n$, it aims to maximize the total number of assigned tasks during the time interval. The problem is proved to be NP-hard. To solve the problem, it first proposes a greedy strategy to do the maximum assignment at every time instance by reducing the problem to the maximum flow problem on a transformed graph. Then it improves the performance by giving high priorities to the tasks with high location entropy during the assignment, where a high location entropy indicates that there are many workers around the area of the task which can be more easily assigned.

Maximize the Number of Correctly Assigned Tasks [66]. Kazemi et al. [66] assume that (a) each task has a confidence score; (b) each worker has a reputation score and a group of workers have an aggregated reputation score, which can be computed based on the reputation scores of individual workers in the group. The problem aims to maximize the number of tasks assigned to workers or groups, satisfying that (a) the task is in the spatial region of the worker; (b) for a task, the reputation score of the assigned worker or the aggregation score of the assigned group must be larger than the confidence score; (c) a worker can only accept at most T tasks. The problem is proved to be NP-hard, and a greedy algorithm is proposed by iteratively assigning tasks to workers or groups until no more valid assignments exist. It also proposes an incremental search strategy which removes a single task-worker (task-group) assignment generated by the greedy algorithm and replaces it with assignments of more tasks. The incremental search process is computationally expensive, and heuristics are proposed to improve its efficiency.

Maximize Successful Rate [111]. Note that some tasks may not be able to be successfully completed. For example, a worker may give up a task, and the task can only be completed successfully with a certain probability. Hassan et al. [111] aim to maximize the successful rate under limited budget. The problem is analogous to the well-known multi-armed bandit problem where each worker is considered as an arm and each assignment is equivalent to paying an arm. The probability of being successful is the same as the resulting reward. It extends the traditional multi-armed bandit algorithms to the spatial assignment problem.

A drawback of the server assignment model is that workers should report their locations to the SC-server, which can pose privacy issues [109], [97].

7.11.2 Spatial Crowdsourcing with Road Network

There are several works on road network. We have shown that [106] and [134] study the route planning problem in Section 7.8. Artikis et al. [16] utilize spatial crowdsourcing to manage urban traffic on road network, where the crowdsourcing component is used to supplement the data sources by querying human volunteers. If two sensors disagree on a traffic situation, e.g., the same bus within a small time period reports two different congestion degrees, they will search workers close to the location with disagreements, and ask them about the real traffic situation.

8 Crowdsourced Optimization and Systems

There are several crowdsourcing systems that integrate crowdsourcing into relational database management systems (RDBMS) and enable RDBMS to process computer-hard queries. The basic workflow of query processing consists of query parser, query plan generation, optimization, and execution. Given a query, a parser is first applied and multiple plans can be generated. Then the query optimization selects the best query plan, and finally, the plan is executed with both machines and the crowd. Existing crowdsourcing database systems focus on query model, query operators, and query optimization techniques. Next we discuss different existing crowdsourced systems.

8.1 CrowdDB

CrowdDB [42] extends SQL and defines a new query language, called CrowdSQL, that is used to define which table or attribute should be crowdsourced. In query processing, CrowdDB introduces three crowd operators. (1) CrowdProbe: collect missing information of attributes or new tuples from the crowd. The typical user interface of CrowdProbe is a form with several fields for collecting information from the crowd. (2) CrowdJoin: implement an index nested-loop join over two tables, where at least one of which is crowdsourced. In particular, the inner relation must be a Crowd table and the user interface is used to crowdsource new tuples of inner relation which can be joined with the tuples in outer relation. (3) CrowdCompare: This operator is designed to implement two functions, Crowd-Equal and Crowd-Order, defined in the CrowdDB's query model. The interface of the operator crowdsources two tuples and leverages the crowd to compare these tuples. Crowd-Equal compares two values and asks the crowd to decide whether they have the same value. Crowd-Order asks the crowd to give an order according to a predefined attribute. CrowdDB also proposes rule-based optimization techniques for processing queries with multiple operators.

8.2 Qurk

Qurk [80] uses a SQL-based query language with user-defined functions (UDFs) to enable crowdsourced data management. To facilitate users to implement the UDFs, Qurk has several pre-defined task templates that can generate the UIs for posting different kinds of tasks to the crowd. Typical crowdsourcing tasks include: 1) Filter: produce tuples that satisfy the conditions specified in the UDF. 2) Sort: rank the input tuples according to the UDFs specified in the order-by clause. 3) Join: compare input tuples and perform join according to the UDF. 4) Generative: allow workers to generate data for multiple fields.

In query processing, Qurk focuses on implementing join and sort. (1) CrowdJoin: Similar to CrowdDB, Qurk also implements a block nested loop join and crowdsources the tuples from two tables for evaluating if they satisfy join conditions. In particular, Qurk studies the techniques for batching multiple comparisons to reduce the cost. (2) CrowdSort: Qurk implements two basic approaches to execute sort. The comparison-based approach solicits the crowd to directly specify the ordering of items. This approach may be expensive for large datasets due to the quadratic comparison. Another task type, rating, is used to reduce the cost using a well-defined interface (see Section 7.4).

Qurk has two important components for cost optimization: task cache and task model. Task cache maintains the crowdsourced answers from previous tasks, and task model trains a model to predict the results for the tasks based on the data that are already collected from the crowd. So if one task can get necessary information from task cache or task model, it will not be published to the crowdsourcing platform. Once the task cannot get useful information from task cache and task model, it will be pushed to the task complier. The complier generates and publishes the tasks to the crowdsourcing platform. Statistic manager determines the number of tasks, assignment and the cost for each task.

8.3 Deco

Deco [90] separates the user view and system view. The logical relations are specified by a schema designer and queried by an end-user. Raw schema is stored in the RDBMS and it is invisible to the schema designer and users. Deco focuses on crowdsourcing missing values or new tuples based on the defined fetch rules. Deco designs fetch rules that allow the schema designers to specify how data are collected from the crowd. Given a fetch rule: get the value of attribute A2 given the value of attribute A1, Deco presents the values of attributes in A1 and asks the crowd to give the values of attributes in A2. For instance, given "China $\rightarrow$ Capital", Deco collects the capital of China from the crowd. In particular, if A1 is empty, the system fetchs new values of attributes in A2. As inconsistency may exist in the collected data, Deco can also specify resolution rules such as de-duplication and majority voting to resolve inconsistencies in the collected data. Deco also supports other operators, such as Dependent Left Outer Join, Filter and Scan.

Based on the defined operators, given a complicated query, a fundamental query optimization problem is [92]

How to find the best query plan to the query, which has the least estimated monetary cost across all possible query plans.

To solve the problem, [92] first defines the monetary cost. Considering the fact that the existing data in the database can be leveraged, the cost is formally defined as the new data that needs to be obtained from the crowd. In order to find the best query plan with the minimum cost, there are two problems addressed in [92]. (1) Cost Estimation: how to estimate the cost of a query plan. As a query plan is executed, the database may collect new data from the crowd, which may affect the cost estimation of subsequent processes. By considering this effect, [92] proposes an iterative approach to estimate the cost for a query plan. (2) Optimal query plan generation. Simply enumerating all possible query plans is computationally expensive, and [92] considers to reuse the common sub-plans in order to reduce

the redundant computation. Then the best query plan, i.e., with least estimated cost, can be returned.

9 Crowdsourcing Platforms

We introduce crowdsourcing platforms that can be used to evaluate crowdsourced data management techniques.

9.1 Amazon Mechanical Turk (AMT)

AMT [1] is a widely used crowdsourcing platform. AMT focuses on micro-tasks, e.g., labeling an image. A requester can group multiple micro-tasks as a Human Intelligence Task (called HIT). The requester can also set some requirements, e.g., the price of a HIT, the time constraint for answering a HIT, the expiration time for a job to be available on AMT, and the qualification test.

A requester can build HITs from several different ways. (1) The requester user interface. AMT provides many templates, such as categorization, data collection, sentiment analysis and image tagging. After designing the worker interface, requesters need to upload the task files. The user interface is very easy to use, even for untrained requesters. (2) AMT Command Line Tools (CLT). AMT predefines a set of commands in CLT, such as loadHITs, getResults, grantBonus, blockWorker and so on. Requesters can utilize CLT to easily build HITs by specifying three files: the data file, the task file to specify user interface for the tasks, and the property file to add the title, description, keywords, reward and assignments for the HIT. The CLT is suitable when a requester has a relatively small number of assignments. (3) AMT APIs. There are many APIs in AMT, including the creation of HITs, block/unblock the workers, collection of the finished answers and statistics collection for the requester and workers. There are three steps to use the APIs to publish tasks: (i) download the SDK for a specified language, e.g., python and java; (ii) specify the title, description, reward, the content of tasks and the detailed properties for the HIT; (iii) publish the HITs to the platform. (4) Requesters can build their own server to manage the tasks and embed their tasks into AMT using innerHTML. When a worker requires a task, AMT transforms the requirement to the requester's server and then the requester can decide how to assign tasks to the server. When a worker submits an answer to AMT, AMT also transforms the result to the requester.

A worker can browse HITs on AMT. Each HIT has some information, e.g., the description of the task, the price, the keywords, the qualification test if required, and the requester's id. After a worker submits the answers of HITs to the platform, s/he can find the total earnings and the status of the submitted HITs on the platform.

9.2 CrowdFlower

CrowdFlower [2] has similar functionalities with AMT, but they still have some differences. First, CrowdFlower has a quality-control component, and it leverages the Gold-Injected method (Section 4) to block low-quality workers. Second, besides publishing the tasks on its own platform, CrowdFlower also publish the tasks on other platforms.

9.3 Other Platforms

There are other crowdsourcing platforms. ChinaCrowd [4] is a multilingual crowdsourcing platform, which supports Chinese and English. Upwork [3] can support macro-tasks, e.g., developing a mobile application. gMission [25] is a spatial crowdsourcing platform that supports spatial tasks.

10 Research Challenges

In this section, we discuss some research challenges and opportunities in crowdsourced data management.

Query Plan. Because SQL is a declarative query language, a single query often corresponds to multiple query plans; it relies on a query optimizer to select the best plan. Traditionally, the way a query optimizer works is to estimate the computation cost of each query plan and choose the one with the minimum estimated cost. However, this process turns to be quite challenging in a crowdsourcing environment because (1) there are three optimization objectives (result quality, monetary cost, and latency) that need to be considered and (2) humans are much more unpredictable than machines.

Benchmark. A large variety of TPC benchmarks (e.g., TPC-H for analytic workloads, TPC-DI for data integration) standardize performance comparisons for database systems and promote the development of database research. Although there are some open datasets (http://dbgroup.cs.tsinghua.edu.cn/ligl/crowddata), there is still lack of standardized benchmarks available. In order to better explore the research topic, it is important to study how to develop evaluation methodologies and benchmarks for crowdsourced data management systems.

Big Data. In the big data era, data volumes are increasing very fast. Compared to machines, humans are a lot more expensive, thus it would be increasingly more costly to apply crowdsourcing to emerging big data scenarios. There are some existing works that aim to address this problem, but they only work for some certain data processing tasks, such as data cleaning [119], data labeling [82]. Therefore, it is important to continue this study and to develop new techniques that work for all kinds of data processing tasks.

Macro-Tasks. Most of existing studies focus on micro-tasks, which can be easily assigned to workers and instantly answered by workers. However many real applications need to use macro-tasks, such as writing a paper. Macro-tasks are hard to be split and accomplished by multiple workers, because they will loose the context information if they are split [50]. Workers are not interested in answering a whole macro-task as each macro-task will take a long time. Thus it is rather challenging to support macro-tasks, including automatically splitting a macro-task, assigning tasks to crowd or machines, and automatically aggregating the answers.

Privacy. There are several types of privacy issues in crowdsourcing. First, the requester wants to protect the privacy of their tasks [128]. The tasks may contain sensitive attributes and could cause privacy leakage. Malicious workers could link them with other public datasets to reveal individual private information. Although the requester can publish anonymity data to the workers using existing privacy techniques, e.g., K-Anonymity, it may lower down the quality as the workers cannot get the precise data. Thus it is challenging to trade-off the accuracy and privacy for requesters. Second, the workers have privacy-preserving requirement. Personal information of workers can be inferred from the answers provided by the workers, such as their location, profession, hobby. On the other hand, the requester wants to assign their tasks to appropriate workers that are skilled at their tasks (or close to the tasks). And it is challenging to devise privacy-preserving task assignment techniques.

Mobile Crowdsourcing. With the growing popularity of smartphones, there are emerging more and more mobile crowdsourcing platforms, e.g., gMission [25], Waze [5], ChinaCrowd [4]. These mobile platforms pose many new challenges for crowdsourced data management. First, many more factors (e.g., spatial distance, mobile user interface) will affect workers' latency and quality. It is more challenging to control quality, latency and cost for mobile platforms. Second, traditional crowdsourcing platforms adopt worker selection model to assign tasks; however mobile crowdsourcing requires to support server assignment model (see Section 7.11). It calls for new task assignment techniques.

11 Conclusion

In this paper, we review extensive studies on crowdsourced data management. Most of existing algorithms focus on balancing quality, cost, and latency, and we summarize all of existing techniques to address these challenges. For quality control, we review the worker modeling, worker elimination, answer aggregation, and task assignment techniques; for cost control, we discuss the pruning, task selection, answer deduction, sampling, and miscellaneous techniques; for latency, we discuss the pricing, latency models including round model and statistical model. Next, we discuss the task types that are widely used to support various applications in crowdsourcing platforms, and review the techniques to support various crowdsourced operators. We also review existing crowdsourced data management systems and optimization techniques. Finally, we discuss crowdsourcing platforms and provide the challenges to improve crowdsourced data management.

Acknowledgement

This research is supported in part by the 973 Program of China (2015CB358700), the NSF of China (61422205, 61472198), Tencent, Huawei, Shenzhou, FDCT/116/2013/A3, MYRG105(Y1-L3)-FST13-GZ, the Chinese Special Project of Science and Technology (2013zx01039-002-002), NSF CISE Expeditions Award CCF-1139158, DOE Award SN10040 DE-SC0012463, DARPA XData Award FA8750-12-2-0331, and gifts from Amazon Web Services, Google, IBM, SAP, The Thomas and Stacey Siebel Foundation, Adatao, Adobe, Apple Inc., Blue Goji, Bosch, Cisco, Cray, Cloudera, Ericsson, Facebook, Fujitsu, Guavus, HP, Huawei, Intel, Microsoft, Pivotal, Samsung, Schlumberger, Splunk, State Farm, Virdata and VMware.

References

[1] https://www.mturk.com/.
[2] http://www.crowdflower.com.
[3] https://www.upwork.com.
[4] http://www.chinacrowds.com.
[5] https://www.waze.com.
[6] R. M. Adelsman and A. B. Whinston. Sophisticated voting with information for two voting functions. *Journal of Economic Theory*, 15(1):145–159, 1977.
[7] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *SIGMOD*, pages 207–216, 1993.
[8] A. Amarilli, Y. Amsterdamer, and T. Milo. On the complexity of mining itemsets from the crowd using taxonomies. In *ICDT*, 2015.
[9] Y. Amsterdamer, S. Davidson, A. Kukliansky, T. Milo, S. Novgorodov, and A. Somech. Managing general and individual knowledge in crowd mining applications. In *CIDR*, 2015.
[10] Y. Amsterdamer, S. B. Davidson, T. Milo, S. Novgorodov, and A. Somech. Oassis: query driven crowd mining. In *SIGMOD*, pages 589–600. ACM, 2014.
[11] Y. Amsterdamer, S. B. Davidson, T. Milo, S. Novgorodov, and A. Somech. Ontology assisted crowd mining. *PVLDB*, 7(13):1597–1600, 2014.
[12] Y. Amsterdamer, Y. Grossman, T. Milo, and P. Senellart. Crowd mining. In *SIGMOD*, pages 241–252. ACM, 2013.
[13] Y. Amsterdamer, Y. Grossman, T. Milo, and P. Senellart. Crowdminer: Mining association rules from the crowd. *PVLDB*, 6(12):1250–1253, 2013.
[14] Y. Amsterdamer, A. Kukliansky, and T. Milo. Nl2cm: A natural language interface to crowd mining. In *SIGMOD*, pages 1433–1438. ACM, 2015.
[15] A.P.Dawid and A.M.Skene. Maximum likelihood estimation of observer error-rates using em algorithm. *Appl.Statist.*, 28(1):20–28, 1979.
[16] A. Artikis, M. Weidlich, F. Schnitzler, I. Boutsis, T. Liebig, N. Piatkowski, C. Bockermann, K. Morik, V. Kalogeraki, J. Marecek, et al. Heterogeneous stream processing and crowdsourcing for urban traffic management. In *EDBT*, pages 712–723, 2014.
[17] B. I. Aydin, Y. S. Yilmaz, Y. Li, Q. Li, J. Gao, and M. Demirbas. Crowdsourcing for multiple-choice question answering. In *AAAI*, pages 2946–2953, 2014.
[18] R. Boim, O. Greenshpan, T. Milo, S. Novgorodov, N. Polyzotis, and W. C. Tan. Asking the right questions in crowd data sourcing. In *ICDE*, 2012.
[19] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.
[20] R. A. Bradley and M. E. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, pages 324–345, 1952.
[21] R. Busa-Fekete, B. Szorenyi, W. Cheng, P. Weng, and E. Hullermeier. Top-k selection based on adaptive sampling of noisy preferences. In *ICML*, pages 1094–1102, 2013.
[22] C. C. Cao, J. She, Y. Tong, and L. Chen. Whom to ask? jury selection for decision making tasks on micro-blog services. *PVLDB*, 5(11):1495–1506, 2012.
[23] X. Chen, P. N. Bennett, K. Collins-Thompson, and E. Horvitz. Pairwise ranking aggregation in a crowdsourced setting. In *WSDM*, pages 193–202, 2013.
[24] X. Chen, Q. Lin, and D. Zhou. Optimistic knowledge gradient policy for optimal budget allocation in crowdsourcing. In *ICML*, pages 64–72, 2013.
[25] Z. Chen, R. Fu, Z. Zhao, Z. Liu, L. Xia, L. Chen, P. Cheng, C. C. Cao, Y. Tong, and C. J. Zhang. gmission: a general spatial crowdsourcing platform. *PVLDB*, 7(13):1629–1632, 2014.
[26] P. Dai, C. H. Lin, Mausam, and D. S. Weld. Pomdp-based control of workflows for crowdsourcing. *Artif. Intell.*, 202:52–85, 2013.
[27] 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.
[28] 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.
[29] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *J.R.Statist.Soc.B*, 30(1):1–38, 1977.
[30] D. Deng, C. Shahabi, and U. Demiryurek. Maximizing the number of worker's self-selected tasks in spatial crowdsourcing. In *SIGSPATIAL*, pages 324–333. ACM, 2013.
[31] C. B. Eiben, J. B. Siegel, J. B. Bale, S. Cooper, F. Khatib, B. W. Shen, F. Players, B. L. Stoddard, Z. Popovic, and D. Baker. Increased diels-alderase activity through backbone remodeling guided by foldit players. *Nature biotechnology*, 30(2):190–192, 2012.
[32] A. E. Elo. *The rating of chessplayers, past and present*, volume 3. Batsford London, 1978.
[33] B. Eriksson. Learning to top-k search using pairwise comparisons. In *AISTATS*, pages 265–273, 2013.
[34] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66(4):614–656, 2003.
[35] J. Fan, G. Li, B. C. Ooi, K. Tan, and J. Feng. icrowd: An adaptive crowdsourcing framework. In *SIGMOD*, pages 1015–1030, 2015.
[36] J. Fan, M. Lu, B. C. Ooi, W.-C. Tan, and M. Zhang. A hybrid machine-crowdsourcing system for matching web tables. In *ICDE*, pages 976–987. IEEE, 2014.
[37] M. Fang, J. Yin, and D. Tao. Active learning for crowdsourcing using knowledge transfer. In *AAAI*, pages 1809–1815, 2014.

[38] Y. Fang, H. Sun, G. Li, R. Zhang, and J. Huai. Effective result inference for context-sensitive tasks in crowdsourcing. In *DASFAA*, 2016.

[39] S. Faradani, B. Hartmann, and P. G. Ipeirotis. What's the right price? pricing tasks for finishing on time. In *AAAI Workshop*, 2011.

[40] U. Feige, P. Raghavan, D. Peleg, and E. Upfal. Computing with noisy information. *SIAM J. Comput.*, pages 1001–1018, 1994.

[41] J. Feng, G. Li, H. Wang, and J. Feng. Incremental quality inference in crowdsourcing. In *DASFAA*, pages 453–467, 2014.

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

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

[44] Y. Gao and A. G. Parameswaran. Finish them!: Pricing algorithms for human computation. *PVLDB*, 7(14):1965–1976, 2014.

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

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

[47] B. Groz and T. Milo. Skyline queries with noisy comparisons. In *PODS*, pages 185–198, 2015.

[48] A. Gruenheid, D. Kossmann, S. Ramesh, and F. Widmer. Crowdsourcing entity resolution: When is A=B? Technical report, ETH Zürich.

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

[50] D. Haas, J. Ansel, L. Gu, and A. Marcus. Argonaut: Macrotask crowdsourcing for complex data processing. *PVLDB*, 8(12):1642–1653, 2015.

[51] D. Haas, J. Wang, E. Wu, and M. J. Franklin. Clamshell: Speeding up crowds for low-latency data labeling. *PVLDB*, 9(4):372–383, 2015.

[52] H. Heikinheimo and A. Ukkonen. The crowd-median algorithm. In *HCOMP*, 2013.

[53] R. Herbrich, T. Minka, and T. Graepel. Trueskill: A bayesian skill rating system. In *NIPS*, pages 569–576, 2006.

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

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

[56] H. Hu, G. Li, Z. Bao, and J. Feng. Crowdsourcing-based real-time urban traffic speed estimation: From speed to trend. In *ICDE*, 2016.

[57] H. Hu, Y. Zheng, Z. Bao, G. Li, and J. Feng. Crowdsourced poi labelling: Location-aware result inference and task assignment. In *ICDE*, 2016.

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

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

[60] X. Jiang, L.-H. Lim, Y. Yao, and Y. Ye. Statistical ranking and combinatorial hodge theory. *Math. Program.*, pages 203–244, 2011.

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

[62] M. Joglekar, H. Garcia-Molina, and A. G. Parameswaran. Comprehensive and reliable crowd assessment algorithms. In *ICDE*, pages 195–206, 2015.

[63] H. Kaplan, I. Lotosh, T. Milo, and S. Novgorodov. Answering planning queries with the crowd. *PVLDB*, 6(9):697–708, 2013.

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

[65] L. Kazemi and C. Shahabi. Geocrowd: enabling query answering with spatial crowdsourcing. In *SIGSPATIAL*, pages 189–198. ACM, 2012.

[66] L. Kazemi, C. Shahabi, and L. Chen. Geotrucrowd: trustworthy query answering with spatial crowdsourcing. In *SIGSPATIAL*, pages 304–313, 2013.

[67] A. R. Khan and H. Garcia-Molina. Hybrid strategies for finding the max with the crowd. Technical report, 2014.

[68] D. Koller and N. Friedman. *Probabilistic Graphical Models - Principles and Techniques*. MIT Press, 2009.

[69] L. I. Kuncheva, C. J. Whitaker, and C. A. Shipp. Limits on the majority vote accuracy in classifier fusion. *Pattern Anal. Appl.*, 6(1):22–31, 2003.

[70] A. Kurve, D. J. Miller, and G. Kesidis. Multicategory crowdsourcing accounting for variable task difficulty, worker skill, and worker intention. *TKDE*, 27(3):794–809, 2015.

[71] G. Li, D. Deng, J. Wang, and J. Feng. PASS-JOIN: A partition-based method for similarity joins. *PVLDB*, 5(3):253–264, 2011.

[72] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and computation*, 108(2):212–261, 1994.

[73] Q. Liu, J. Peng, and A. T. Ihler. Variational inference for crowdsourcing. In *NIPS*, pages 701–709, 2012.

[74] 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.

[75] C. Lofi, K. E. Maarry, and W. Balke. Skyline queries in crowd-enabled databases. In *EDBT*, pages 465–476, 2013.

[76] C. Lofi, K. E. Maarry, and W. Balke. Skyline queries over incomplete data - error models for focused crowd-sourcing. In *ER*, pages 298–312, 2013.

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

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

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

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

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

[82] B. Mozafari, P. Sarkar, M. Franklin, M. Jordan, and S. Madden. Scaling up crowd-sourcing to very large datasets: a case for active learning. *PVLDB*, 8(2):125–136, 2014.

[83] D. Nau, M. Ghallab, and P. Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[84] R. Neal and G. E. Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in Graphical Models*, pages 355–368. Kluwer Academic Publishers, 1998.

[85] S. Negahban, S. Oh, and D. Shah. Iterative ranking from pairwise comparisons. In *NIPS*, pages 2483–2491, 2012.

[86] Q. V. H. Nguyen, T. T. Nguyen, Z. Miklós, K. Aberer, A. Gal, and M. Weidlich. Pay-as-you-go reconciliation in schema matching networks. In *ICDE*, pages 220–231. IEEE, 2014.

[87] W. R. Ouyang, L. M. Kaplan, P. Martin, A. Toniolo, M. B. Srivastava, and T. J. Norman. Debiasing crowdsourced quantitative characteristics in local businesses and services. In *IPSN*, pages 190–201, 2015.

[88] A. G. Parameswaran, S. Boyd, H. Garcia-Molina, A. Gupta, N. Polyzotis, and J. Widom. Optimal crowd-powered rating and filtering algorithms. *PVLDB*, 7(9):685–696, 2014.

[89] 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.

[90] A. G. Parameswaran, H. Park, H. Garcia-Molina, N. Polyzotis, and J. Widom. Deco: declarative crowdsourcing. In *CIKM*, pages 1203–1212. ACM, 2012.

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

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

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

[94] J. Pearl. Reverend bayes on inference engines: A distributed hierarchical approach. In *AAAI*, pages 133–136, 1982.

[95] T. Pfeiffer, X. A. Gao, Y. Chen, A. Mao, and D. G. Rand. Adaptive polling for information aggregation. In *AAAI*, 2012.

[96] J.-C. Pomerol and S. Barba-Romero. *Multicriterion decision in management: principles and practice*, volume 25. Springer, 2000.

[97] L. Pournajaf, L. Xiong, V. Sunderam, and S. Goryczka. Spatial task assignment for crowd sensing with cloaked locations. In *MDM*, volume 1, pages 73–82. IEEE, 2014.

[98] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDBJ*, 10(4):334–350, 2001.

[99] V. C. Raykar and S. Yu. Eliminating spammers and ranking annotators for crowdsourced labeling tasks. *Journal of Machine Learning Research*, 13:491–518, 2012.

[100] V. C. Raykar, S. Yu, L. H. Zhao, A. K. Jerebko, C. Florin, G. H. Valadez, L. Bogoni, and L. Moy. Supervised learning from multiple experts: whom to trust when everyone lies a bit. In *ICML*, pages 889–896, 2009.

[101] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *SIGKDD*, pages 269–278, 2002.

[102] A. D. Sarma, A. Jain, A. Nandi, A. Parameswaran, and J. Widom. Jellybean: Crowd-powered image counting algorithms. Technical report, Stanford University.

[103] A. D. Sarma, A. G. Parameswaran, H. Garcia-Molina, and A. Y. Halevy. Crowd-powered find algorithms. In *ICDE*, pages 964–975, 2014.

[104] P. Smyth, U. M. Fayyad, M. C. Burl, P. Perona, and P. Baldi. Inferring ground truth from subjective labelling of venus images. In *NIPS*, pages 1085–1092, 1994.

[105] H. Su, K. Zheng, J. Huang, H. Jeung, L. Chen, and X. Zhou. Crowdplanner: A crowd-based route recommendation system. In *ICDE*, pages 1144–1155. IEEE, 2014.

[106] H. Su, K. Zheng, J. Huang, T. Liu, H. Wang, and X. Zhou. A crowd-based route recommendation system-crowdplanner. In *ICDE*, pages 1178–1181, 2014.

[107] R. S. Sutton and A. G. Barto. Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 9(5):1054–1054, 1998.

[108] K. Talamadupula, S. Kambhampati, Y. Hu, T. A. Nguyen, and H. H. Zhuo. Herding the crowd: Automated planning for crowdsourced planning. In *HCOMP*, 2013.

[109] H. To, G. Ghinita, and C. Shahabi. A framework for protecting worker location privacy in spatial crowdsourcing. *PVLDB*, 7(10):919–930, 2014.

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

[111] U. ul Hassan and E. Curry. A multi-armed bandit approach to online spatial task assignment. In *UIC*, 2014.

[112] M. Venanzi, J. Guiver, G. Kazai, P. Kohli, and M. Shokouhi. Community-based bayesian aggregation models for crowdsourcing. In *WWW*, pages 155–164, 2014.

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

[114] V. Verroios and H. Garcia-Molina. Entity resolution with crowd errors. In *ICDE*, pages 219–230, 2015.

[115] V. Verroios, P. Lofgren, and H. Garcia-Molina. tdp: An optimal-latency budget allocation strategy for crowdsourced MAXIMUM operations. In *SIGMOD*, pages 1047–1062, 2015.

[116] N. Vesdapunt, K. Bellare, and N. N. Dalvi. Crowdsourcing algorithms for entity resolution. *PVLDB*, 7(12):1071–1082, 2014.

[117] L. Von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum. recaptcha: Human-based character recognition via web security measures. *Science*, 321(5895):1465–1468, 2008.

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

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

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

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

[122] S. Wang, X. Xiao, and C. Lee. Crowd-based deduplication: An adaptive approach. In *SIGMOD*, pages 1263–1277, 2015.

[123] F. L. Wauthier, M. I. Jordan, and N. Jojic. Efficient ranking from pairwise comparisons. In *ICML*, pages 109–117, 2013.

[124] P. Welinder and P. Perona. Online crowdsourcing: rating annotators and obtaining cost-effective labels. In *CVPR Workshop (ACVHL)*, pages 25–32. IEEE, 2010.

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

[126] S. E. Whang, J. McAuley, and H. Garcia-Molina. Compare me maybe: Crowd entity resolution interfaces. Technical report, Stanford University.

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

[128] S. Wu, X. Wang, S. Wang, Z. Zhang, and A. K. H. Tung. K-anonymity for crowdsourcing database. *TKDE*, 26(9):2207–2221, 2014.

[129] T. Yan, V. Kumar, and D. Ganesan. Crowdsearch: exploiting crowds for accurate real-time image search on mobile phones. In *MobiSys*, pages 77–90, 2010.

[130] Y. Yan, G. M. Fung, R. Rosales, and J. G. Dy. Active learning from crowds. In *ICML*, pages 1161–1168, 2011.

[131] P. Ye, U. EDU, and D. Doermann. Combining preference and absolute judgements in a crowd-sourced setting. In *ICML Workshop*, 2013.

[132] J. Yi, R. Jin, A. K. Jain, S. Jain, and T. Yang. Semi-crowdsourced clustering: Generalizing crowd labeling by robust distance metric learning. In *NIPS*, pages 1781–1789, 2012.

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

[134] C. J. Zhang, Y. Tong, and L. Chen. Where to: Crowd-aided path selection. *PVLDB*, 7(14):2005–2016, 2014.

[135] X. Zhang, G. Li, and J. Feng. Crowdsourced top-k algorithms: An experimental evaluation. *PVLDB*, 9(4), 2015.

[136] 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.

[137] Z. Zhao, D. Yan, W. Ng, and S. Gao. A transfer learning based framework of crowd-selection on twitter. In *SIGKDD*, pages 1514–1517, 2013.

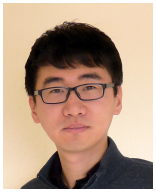
[138] Y. Zheng, R. Cheng, S. Maniu, and L. Mo. On optimality of jury selection in crowdsourcing. In *EDBT*, pages 193–204, 2015.

[139] 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.

[140] J. Zhong, K. Tang, and Z. Zhou. Active learning from crowds with unsure option. In *IJCAI*, pages 1061–1068, 2015.

[141] H. H. Zhuo. Crowdsourced action-model acquisition for planning. In *AAAI*, 2015.

Guoliang Li is currently working as an associate professor in the Department of Computer Science, Tsinghua University, Beijing, China. He received his PhD degree in Computer Science from Tsinghua University, Beijing, China in 2009. His research interests mainly include data cleaning and integration, spatial databases and crowdsourcing.



Jiannan Wang is a tenure-track assistant professor in the School of Computing Science, Simon Fraser University, Burnaby, Canada. He was a Postdoctoral Associate in the AMPLab at UC Berkeley during 2013-2015. He received his PhD degree in Computer Science from Tsinghua University, Beijing, China in 2013. His research interests mainly include data cleaning and crowdsourcing.

Yudian Zheng received his B.E. degree in Software Institute from Nanjing University in 2013. He is currently a PhD student at the Department of Computer Science in The University of Hong Kong. His main research interests include data analysis and data management in crowdsourcing.

Michael J. Franklin received his PhD degree in computer science from the University of Wisconsin-Madison in 1993. He is the Thomas M. Siebel professor of computer science at UC Berkeley, where he also serves as a director of the Algorithms, Machines and People Lab (AMP Lab). His research interests include large-scale data management and analytics, data integration, and hybrid human/computer data processing systems. He was a cofounder and CTO of Truviso, a real-time data analytics company acquired by Cisco Systems in 2012. He is a fellow of the ACM and two-time winner of the ACM SIGMOD Test of Time Award (2013 and 2004).