Challenges in Data Crowdsourcing

Hector Garcia-Molina, *Member, IEEE*, Manas Joglekar, Adam Marcus, Aditya Parameswaran, and Vasilis Verroios

Abstract—Crowdsourcing refers to solving large problems by involving human workers that solve component sub-problems or tasks. In data crowdsourcing, the problem involves data acquisition, management, and analysis. In this paper, we provide an overview of data crowdsourcing, giving examples of problems that the authors have tackled, and presenting the key design steps involved in implementing a crowdsourced solution. We also discuss some of the open challenges that remain to be solved.

Index Terms—Data crowdsourcing, data augmenting, data curation, data processing, crowdsourcing space, crowdsourcing design, crowdsourcing workflow, worker management

1 Introduction

Crowdsourcing, i.e., recruiting human workers to perform tasks, is an effective solution for problems that are inherently hard for computers to perform on their own. Such problems include sentiment analysis of text [49], image or video classification or tagging [56], matching data records that belong to the same entity [57], as well as creative tasks like writing novels, and software development. Just to give one example, consider a clothing store that wants to determine what clothing combinations fit well together (in terms of matching colors, compatible designs, shared purpose, and so on).[1] A computer vision program would have difficulty (at least in the foreseeable future) in determining what combinations will be aesthetically pleasing to people. Furthermore, there are too many clothing combinations for a single human expert to analyze. With a crowdsourcing solution a large pool of human workers can be asked to evaluate these combinations. In each worker task, the worker compares one or a small number of combinations, and ranks them in some way. A coordinating computer can then gather the results and select the overall winners.

The main reason why crowdsourcing has become popular over the last decade is the availability of internet platforms like Amazon Mechanical Turk that enable the large-scale participation of human workers. These platforms match available tasks to humans willing to work, and act as an intermediary for the financial compensation provided to workers. There are many other platforms available, such as UpWork [3] and Samasource [5], as well as companies specializing in implementing and optimizing crowdsourced applications, such as CrowdFlower [1]. (The site http://www.crowdsourcing.org/ is a good resource on industrial crowdsourcing.)

In this paper we focus on data crowdsourcing, where the goal is to address data processing and management problems, as opposed to applications related to say funding or ad design. We will discuss how data crowdsourcing relates to other types of crowdsourcing, but then we narrow down our scope to the challenges, opportunities and future of data crowdsourcing.

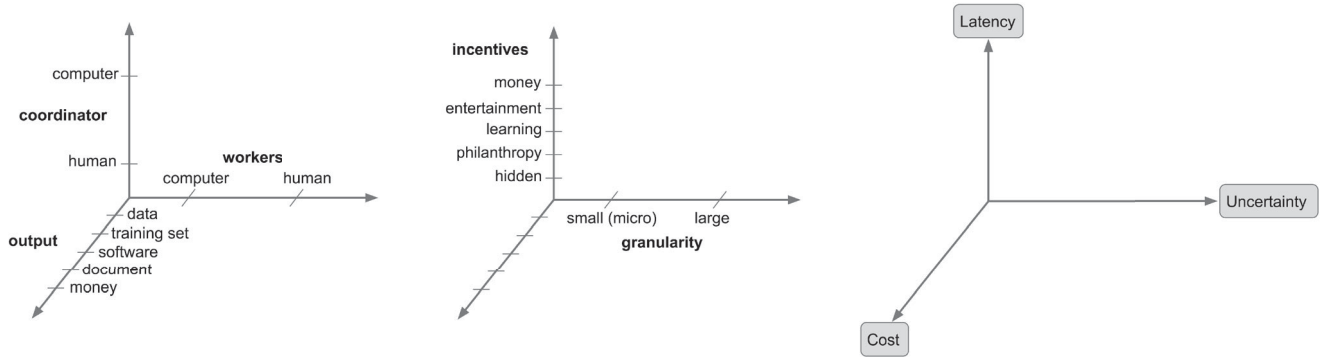
Crowdsourcing has the potential to improve the quality, timeliness and breadth of data. Human workers can report important information in a timely fashion, can verify its accuracy, and can help organize and classify it. The human-provided or processed data can be used in various ways. For instance, human-processed data can be used to enrich, enhance, or clean structured data repositories [18], [48], which in turn can be useful for applications such as keyword search, data integration, or analysis. Human-processed data can also be used as training data for machine learning tasks, enabling us to make progress on a variety of intractable problems in making sense of images, video, and text. Indeed, several companies are using crowdsourcing on a large scale for data crowdsourcing, spending millions of dollars every year [31].

But as we will discuss in this paper, there are also challenges to overcome in order to achieve these potential benefits. For one, human workers are typically slower and more expensive than computers, so they must be used judiciously. Also, human workers can give incorrect answers, for example, if they are not qualified for the work or if they are biased in some way. We also face the issue of "spammers", workers that simply want to make money without doing any actual work. Thus, a good data crowdsourcing system must implement worker evaluation techniques and somehow try to filter out incorrect answers. In addition, it must intelligently manage and split up the problem at hand into tasks, in order to yield good overall response time, cost and accuracy.

1. This is a problem actually faced by a clothing company, StitchFix [6], that uses crowdsourcing to compose clothing package recommendations for consumers.

- *H. Garcia-Molina, M. Joglekar, and V. Verroios are with the Department of Computer Science, Stanford University, Stanford, CA 94305. E-mail: {hector, manasrj, verroios}@cs.stanford.edu.*
- *A. Marcus is with the Unlimited Labs, San Francisco, CA 94103. E-mail: marcua@marcua.net.*
- *A. Parameswaran is with the Department of Computer Science, University of Illinois (UIUC), Champaign, IL 61820. E-mail: adityagp@illinois.edu.*

Manuscript received 17 Aug. 2015; revised 22 Dec. 2015; accepted 5 Jan. 2016. Date of publication 18 Jan. 2016; date of current version 3 Mar. 2016. Recommended for acceptance by J. Pei.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2016.2518669



(a) Coordinator, Workers, and Output (b) Incentives and Granularity (c) Uncertainty, Cost, and Latency

Fig. 1. The general crowdsourcing space, organized in different dimension decompositions.

In addition to those, more technical, challenges, there are also ethical, political and legal ramifications to crowdsourcing. For instance, should workers be entitled to traditional worker benefits (e.g., pensions), even if they work sporadically and in remote countries? What rights do workers have when we stop employing them because we allege their work is of low quality? While these are all very important questions, they are beyond the scope of this paper, where we focus on the more technical challenges surrounding data crowdsourcing.

The rest of the paper is organized as follows: In Section 2, we provide a brief overview of the general crowdsourcing space, narrow our focus to what we call data crowdsourcing, and describe the fundamental tradeoffs in designing data crowdsourcing solutions: accuracy, cost, and latency. In Section 3, we illustrate examples of technical problems in the data crowdsourcing space. These examples will then be used to motivate and explain the solution design choices of the following sections. (We use examples the authors are familiar with and are not meant to be a comprehensive set.) In Section 4, we discuss the steps that often need to be taken when designing a data crowdsourcing system. Finally, in Section 5, we discuss some of the challenges and problems that we believe are not sufficiently explored in current research.

2 General Crowdsourcing Space and Tradeoffs

Crowdsourcing can be used in many different scenarios and can come about in different ways. In this section, we give a brief overview of the "crowdsourcing space of options" by describing some of the main dimensions in this space. We also give some examples of crowdsourcing instances and where they sit in our space of options.

Workers, Coordinator, and Output. Fig. 1a illustrates three of the main dimensions in our space: a) the "workers" axis refers to the units or agents solving the tasks, b) the "coordinator" axis refers to the entity that coordinates the units solving the tasks, and c) the "output" axis refers to type of the outcome, the crowdsourcing solution generates. (Note incidentally we are assuming a single (central) coordinator; tasks can of course be managed by a distributed set of coordinators, but this is not very practical in our context.)

There are two possibilities for the "workers" and "coordinator" axes: human and computer. When both the coordinator and the workers are human, we have a person supervising the task execution, and hiring or sub-contracting other humans to do parts of the job (i.e., tasks). This type of crowdsourcing has been used for many years. For instance, in the "Wild West" (in the United States), a Sheriff looking for an outlaw would post "wanted" posters asking for people to help. In this case, the Sheriff is the coordinator, and the workers are paid a bounty when they complete the task. This human-human region in the space, is still active in modern days: systems like Turkomatic [29], CrowdForge [28], and Flash Teams [42], involve humans in the design of the tasks to be solved by human workers. In these systems there is some automation in the coordination role, so perhaps we can classify these systems as having a hybrid human-computer coordinator (not shown in our diagram).

Another important region in our diagram is the computer coordinator, human workers region. Here a computer (programmed by a human of course) partitions a problem into tasks and asks human workers to solve the resulting tasks. For example, if the problem is to tag a set of photographs, the coordinator can assign each photo to one or more workers for labeling, collect the answers, handle inconsistent tags, and so on. All of the examples we present in Section 3 fall in this category.

The remaining two combinations for the coordinator and worker axis are not that relevant for this paper. When both the coordinator and the worker roles are played by computers, we have in essence, a traditional distributed computing system. (For instance, a coordinator picking the next chess move asks other computers to analyze each of the possible moves.) The final region, a human coordinator and computer workers, could represent say a system administrator running a set of computers.

The output dimension in Fig. 1a refers to the eventual outcome or solution to the crowdsourcing problem. The figure illustrates some, but not all, of the possibilities. The output we focus on in this paper is data: digital records that represent people, products, purchases, maps, images, etc. This data has been analyzed, enhanced or manipulated by the crowd workers in some way. For instance, in our image tagging example, the output is a set of images with labels attached. The "training set" output is a special case of data that is subsequently used as a training set for some machine learning algorithm. (In our labeling example, the machine learning algorithm may attempt to label images automatically based on the human labeled images in the training set.) We make this distinction here simply because crowd

output is often used as a training set later on [47]. The output can also be a document (e.g., soylent [14], wordy.com, novel writing) or software (e.g., Flash Teams [42]). Our last output example is money, where a project, a movie, or a company is funded by a large number of people. This type of funding is referred to as crowdfunding.

Incentives and granularity. In Fig. 1b, we present two additional dimensions that can be used for defining the crowdsourcing space: "incentives" and "granularity". Granularity refers to the size of the tasks performed by workers. For example, in the image tagging example granularity is small (tasks are often called "micro-tasks" in such cases), as each worker performs a relatively small part of the overall problem (and is compensated accordingly). On the other hand, when workers develop the components of a large software system, we have a coarse granularity (tasks could be then called "macro-tasks").

The incentives dimension describes what motivates (human) workers to participate. The most common incentives are money, services, entertainment, learning and philanthropy. Most crowdsourcing platforms use money as incentive, with workers receiving financial compensation for their work. In other cases, the workers receive a service, as opposed to money. For example, a worker may be able to open an account at a website if she solves a puzzle (e.g., Captcha [55]). The puzzle is used by the site to ensure the account is requested by a person (and not a robot), but in addition, the puzzle can be part of a larger problem the site contributes in solving (e.g., converting images containing handwriting into text). Another incentive is entertainment or learning: workers answer crowdsourcing questions while playing games (e.g., Games with a purpose [56]) or while learning a new language (e.g., Duolingo [2]). Finally, philanthropy can also be an incentive: workers answer tasks for social good. For example, in the Christmas Bird Count (http://www.audubon.org/conservation/science/christmas-bird-count) citizen scientists perform a bird census each year; Galaxy Zoo [41] recruits volunteers to identify galaxies in astronomical photographs.

Our focus. As mentioned earlier, the focus of this paper is on crowdsourcing related to data management and analysis. Thus, after setting the stage with our overview of the crowdsourcing space, we now focus on the part of the space where data crowdsourcing is typically done: the coordinator is a computer, workers are humans, the output is data (or training sets), and there is a fine-granularity with small and simple tasks for humans to solve. In addition, the incentive is usually money.

Fundamental tradeoffs. As we consider ways to solve a data crowdsourcing problem, it is important to keep in mind three important conflicting factors: uncertainty, cost and latency (see Fig. 1c).

(a) Uncertainty refers to the possibility that our solution or final answer is incorrect. Human workers often make mistakes, either because the tasks are too hard and they do not have adequate expertise, or because they are not answering the tasks diligently (and are simply guessing). In all these cases, we need to make sure that individual worker mistakes do not affect the eventual solution. *(b) Cost* refers to the monetary cost of performing all tasks. Cost is important because human workers need to be compensated, and overall cost can be significant even if executing each task costs just a few cents but there are a large number of tasks. *(c) Latency* refers to the time taken to solve the problem. Latency is especially critical in a crowdsourcing environment where setting up tasks in a crowdsourcing marketplace and having them performed by humans can take significant time, often many orders of magnitude greater than computer processing times.

Unfortunately, these three factors conflict with each other, making the design of solutions for data crowdsourcing especially difficult. Consider the following examples: *a)* One method for reducing uncertainty is by asking multiple human workers to attempt the same task, and then aggregating the responses in some way. For example, if we are classifying images, we can ask multiple workers to classify each image, and take the majority response. However, asking multiple workers to attempt the same task will increase cost and latency, since more humans need to be compensated, and since we need to wait for more tasks to be completed. *b)* We can issue tasks in parallel, thereby reducing latency, or we can issue tasks serially, increasing latency, but at the same time potentially decreasing cost. To see this, consider the image classification example, with 1,000 images. If we ask, say, three workers to classify the same image, and then take the majority score, and if we issue tasks in parallel, we will always issue $1{,}000 \times 3 = 3{,}000$ tasks. On the other hand, if we issue tasks serially, we can avoid issuing the third task if the answers on the first two tasks match, i.e., the first two workers either classify the image positively or negatively. *c)* Another mechanism to reduce latency is to increase the monetary compensation per task, thereby attracting more workers to such tasks. However, increasing the monetary compensation per task will also proportionally increase the overall cost.

Thus, a crowdsourcing solution needs to balance the desire for low uncertainty, cost and latency. We return to this balance when we discuss optimization subsequently.

3 Example Problems in Data Crowdsourcing

In this section, we briefly illustrate some crowdsourcing problems in the data management space. These examples will then be used to motivate and explain the solution design choices of the following section. Our set of examples is not meant to be exhaustive or complete; they are used here mainly because the authors have some experience with these particular problems.

We divide our examples into two categories, along the lines of classical data processing. Note that these categories are mainly for exposition, and sometimes a particular problem or application cuts across categories. *a)* Data Augmenting: data is collected from sources. In this case the "sources" are human workers that have the needed information (e.g., papers [9], [10], [21], [30], [40]). *b)* Data Processing: existing data is transformed, cleansed, organized and analyzed. In the crowdsourcing case, the processing is done with the assistance of human workers (e.g., papers [13], [37], [57], [60]).

As discussed in the next section, all problems could in principle be solved by a single worker. That is, one crowdsourcing solution would be to ask a single (and very hard working) worker to gather all the data we need, cleanse all

the data we have, or answer all the queries we might have. However, in most cases, this approach is not feasible, because the one worker would be overloaded (or would not have the proper knowledge) and therefore prone to error [43], response time would be huge, and all our data could have similar biases [66]. Instead, our goal is to solve these problems using multiple workers concurrently, each solving a small part of the problem. In Section 4, we will discuss techniques for decomposing problems such as the ones we illustrate here into tasks that can be solved in such a manner.

3.1 Data Augmenting

Human workers are very useful information sources. Some of the information is directly stored in their head, e.g., the capital of Mongolia, or Chinese restaurants in San Francisco where they eat. In other cases, they do not have the information but they know how to get it, e.g., if they need the list of songs in the Beatles White Album, they go to Amazon.com, search for the album, and get the list of songs. Humans are especially good at getting data where the request is not well formed (e.g., we need something for trimming nose hair), the data needs to be up-to-date (e.g., what is the current wait time at a given restaurant), or the data reflects human judgment or opinion (e.g., what are good movies to watch on NetFlix this weekend).

Data can be collected from humans either implicitly or explicitly. The collection is implicit when the people are not aware, e.g., by watching a movie they are telling us this movie is popular. The gathering is explicit when the human gets a request for information. We do not focus here on implicit data gathering, since as mentioned earlier, we are looking at paid workers in our overview.

We can go about explicitly obtaining data from humans procedurally or declaratively. In a procedural approach, a programmer simply writes code that issues tasks to humans. There are a variety of systems that provide procedural mechanisms to decompose a problem into tasks issued by humans, including Turkit [30], Automan [12], Dog [7]. With a declarative approach, the data need is specified by an end user at a high level, and the crowdsourcing system translates that need into data requests to human workers. We give two examples of ways in which the data need can be declaratively defined, i.e., two ways in which the data augmenting problem can be specified:

Data gathering via declarative queries. Systems of this type include Deco [40], CrowdDB [21], and Qurk [33]. Here, a database administrator (DBA) defines one or more database tables, and then end users can issue queries that specify the data needed. (These end users are of course different from the workers who will eventually provide the data.) For example, say the DBA defines the table RESTAURANT (NAME, ADDRESS, CUISINE, RATING, NUMBER). Then, a user can issue the query SELECT * FROM RESTAURANT WHERE CUISINE="Chinese". In this case, the user is interested in populating the RESTAURANT table with Chinese restaurants. Typically the query would also indicate the maximum resources to devote (e.g., time, money) and a desired number of tuples. Here, the user is interested in gathering data *vertically*, i.e., getting new records. The user could also be interested in gathering data *horizontally*, i.e., gathering new fields. For instance, we may have the NAME, ADDRESS, CUISINE of various restaurants, and we want to augment it by adding the RATING and NUMBER (say, by looking at Yelp or the restaurant website).

The system must now translate this query into an execution plan, analogous to how a traditional database system transforms a query into a program that fetches data from files or databases to produce an answer. In a traditional system, the system relies on access plans that tell it how to get data items from specific files. In our case, the DBA needs to also provide access methods to the crowd data, and uncertainty resolution mechanisms. For instance, the DBA may have provided access methods for gathering restaurant name(s) and addresses given a cuisine, or for gathering ratings given a restaurant name and address. In both cases, these access methods translate to tasks that are issued on a particular crowdsourcing platform. Additionally, the DBA needs to specify uncertainty resolution mechanisms for each field, i.e., how to resolve disagreements between worker responses. For instance, for the rating of a particular restaurant, the uncertainty resolution mechanism can specify that the system needs to get responses from five workers, and then averages their votes.

Data retrieval via keyword search. Systems of this type include DataSift [38], [39] and SlowSearch [34]. In this version, the data need is defined by a natural language request that may include drawings or photographs. For instance, say an end user is looking for a specific type of cable. The user has the cable in hand, but needs a replacement because it is broken. The user does not know what type it is (Firewire, USB, USB2, USB3, mini, male-to-female, ...). So the user takes a photograph of both ends of the cable and poses the query "I need to buy a cable like this one, see photo." Note that the user is asking both for the place to buy this item, as well as a description of the product. The answer to the query would be a set of product description pages from various vendors. The system operates like a traditional search engine, except it handles queries that require human intelligence. Of course, a smart end user can search herself using traditional search engines, but this type of system is targeted at end-users who do not have the time or expertise to do the searching themselves.

As argued above, it is usually good to decompose problems. For search, the system may want to first ask workers to discover stores that carry this type of cable. Then it can ask workers to translate the initial query into a keyword search for a particular store. Then the system can (automatically) retrieve products (webpages) from that store using the keyword query. Finally, the system can ask other workers to check if the retrieved products indeed satisfy the original data need.

3.2 Data Processing

Once data is captured, it often needs to be further processed. In a traditional environment, examples include verifying if records satisfy a certain property, mapping product records to an existing taxonomy of products (e.g., paper [50]), grouping records that refer to the same real-world entity into a single composite record (e.g., papers [13], [53], [60]), flagging (labeling) records that are invalid or exceptional, combining (and cleansing) records into an aggregate or report (e.g., papers [17], [63]), and so on. Computers have been doing

this type of data processing for years. However, humans are much better than computers for processing that involves image analysis, natural language understanding, and other tasks that involve sophisticated judgment. We illustrate with three particular problems:

Filtering problem. In the filtering problem [36], [37], we are given a set of records, and our goal is to identify which records satisfy a given condition or property. For instance, spam classification, i.e., identifying which emails in a set of emails are spam, or content moderation, i.e., identifying which images or documents in a set are appropriate to be viewed or consumed by a general audience, are both examples of filtering. In filtering, the primary challenge is to balance cost and uncertainty across a collection of records. For instance, for an email that two independent workers have judged to be not-spam, should we issue another task to get a different worker's opinion, or should we instead issue a task for a second email that has two workers judging it to be spam, and one worker judging it to be not-spam? Should we issue multiple tasks for the same email simultaneously, or should we issue one task at a time? Thus, even this simple problem is non-trivial.

Maximum problem. In the maximum (or best-of) problem [11], [23], [52], [54], we are given a set of records, and a value function. For instance, the records can be applications for a job, and the value function may represent how well suited the candidate is for the job. The goal is to find the best suited candidate for the job (or the best k). Human workers can effectively perform two types of tasks here: comparisons or rankings [32]. In a pair-wise comparison, a worker can examine two records and indicate which one is the better of the two. (A variation would be for the worker to examine n records to find the best one, or to rank the n records.) In a rating task, a worker examines a single record and assigns it a coarse ranking, e.g., 1, 2 3 or 4 stars. The idea is that if a record has a higher rank than another, it is a better one. However, there may be multiple records with the same coarse rank.

The maximum problem is much more challenging than the simple filtering problem because the tasks are related. For example, if one worker determines that record a is better than record b, and another worker determines that b is better than record c, then we know (or at least we assume for this problem) that a must be better than c. Thus, we can avoid the third comparison. But if workers can sometimes make mistakes, then the probability of "a better than c" depends on the probability of errors for the first two comparisons. Furthermore, we may also get cycles in the better-than relationship.

Entity resolution problem. In the entity resolution problem [13], [53], [57], [60] we are given a set of records, where each record refers to a particular real-world entity. For example, each webpage or each photograph may refer to a person. However, the mapping from records to entities is unknown, i.e., records do not contain an entity identifier, and we do not even have a list of all the entities involved. However, humans can examine a pair of records and determine if they refer to the same entity or not. (They may also be able to view more than two records at a time and answer some questions about entities, e.g., are all these photographs of the same person, or which photos do not represent the same person as in photograph 1.)

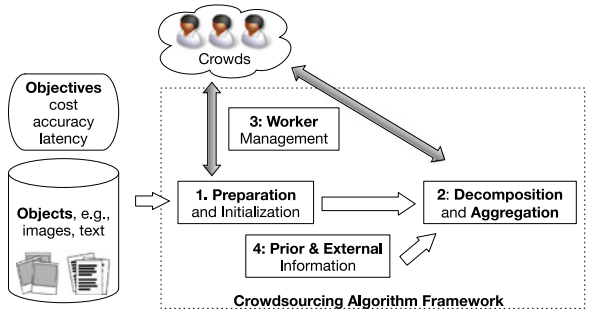


Fig. 2. Data crowdsourcing algorithm design.

As with the maximum problem, comparisons are interrelated, making this problem challenging. In both cases, a brute-force approach would be to ask human workers for N^2 comparisons, one for each possible pairing of the N records. The computer could then examine the results and find the maximum record or group together the records corresponding to the same entity. (Of course, as research in this area [13], [23], [32], [53], [57] suggests, there are much better ways to solve these problems, avoiding many of the unnecessary comparisons, while coping with human errors efficiently.)

The problems we have illustrated in Sections 3.1 and 3.2, have their own specific challenges (e.g., how to optimize a query, or how to find the maximum of a set), but they also share common features. For instance, to solve the problems we need to represent uncertain information, e.g., a worker's answer could be wrong or may take a long time to arrive. We also need mechanisms for interacting with one or more crowdsourcing platforms, sending tasks and collecting answers. There is also often commonality in the steps taken to implement a solution, as discussed in the next section.

4 Designing a Data Crowdsourcing Solution

Given a problem like one of those defined in Section 3, how does one go about building a system that solves the problem using crowd workers? Well, as the saying goes, "there is more than one way to skin a cat", so we do not present any one scheme. Instead, we discuss some of the design steps that are typically considered when implementing a crowdsourced solution.

These steps can be grouped together in the following way (summarized in Fig. 2):

1) *Preparation and initialization.* The first step identifies the building blocks necessary to assemble a data crowdsourcing solution.
2) *Decomposition and aggregation.* The next step selects techniques to decompose the overall problem into smaller tasks for which crowd input can be leveraged, and to aggregate crowd responses to get a consensus answer.
3) *Worker management.* The next step manages the interaction with workers.
4) *Prior & external information.* The last step incorporates external information, when available, into the data crowdsourcing solution.

4.1 Preparation and Initialization

Even before we begin the execution of the data crowdsourcing algorithm, we need to assemble some essential building

blocks. This part is often the trickiest, since it involves a lot of trial-and-error and developer intuition.

Input statement. The first step is to identify (a) the goal or problem, e.g., sort a collection of images, or apply a filter to a set of documents, and (b) the constraints, e.g., we have a monetary budget, or we need to complete the job within the day, or we need at least 95 percent accuracy, for some definition of accuracy. The input statement can dictate the design of the crowdsourcing solution; for instance, if cost is not important, then we can issue as many tasks as needed without regard to cost.

Determining tasks. A critical next step is to determine the type of tasks that will be presented to the human workers. These tasks constitute the "atomic actions" that will be used as building blocks to solve the data crowdsourcing problem. For some problems the choice is straightforward. For instance, for a content moderation task, where we need to tag each image with "inappropriate image(s)" or "acceptable", a task can ask one worker to examine an image and provide the tag. However, for more complex problems, there are more choices to consider. For instance, for the maximum problem, do we ask a worker to compare two records at a time, or to provide a rating (say, from 1-5) for one record at a time? And for each of these two choices there are many variations. For example, for comparisons, does the worker compare two records at a time, or a set of k pairs at a time? (Note that the latter may have more bias due to correlations between the answers for these pairs [66].) Or is the worker shown a set of n records and asked for the best one of the set? Or is the worker asked to order the set? And we can of course have more than one type of task (e.g., both pair-wise comparisons and rankings) for the same problem, in which case the type and particulars of the task are determined at run time, when the problem is decomposed (see below).

The choice of task types has implications, not just for how we decompose the problem, but also on accuracy of results and overall cost. For instance, if we ask a worker to compare n pairs of records, we may have to pay more per task. If we ask a worker to examine many records in one task, the error rate may increase due to fatigue or lack of focus. If we add a verification type of task (where the results of prior tasks are checked by other workers), the overall accuracy may improve, but so will the cost. Note that task identification can often not be done completely before considering the rest of the steps we discuss in this section. That is, the whole process is iterative, perhaps starting with a preliminary choice of tasks, and then refining the choice after subsequent issues (like decomposition) have been studied.

Task interface. For each task type we must also implement a way for a worker to view the relevant data and to provide answers. We refer to such an implementation as the *interface* for the task. For instance, say we are performing entity resolution to determine which photographs refer to the same person. We decide to show a worker six photos and ask him to group together the photos that refer to the same person. Does the worker move the photos on the screen, or does he label them with a group number? Do we allow the worker to zoom into photos to examine detail? Is he allowed to perform web searches (to lookup other related photos) while performing the task? Does the worker have a way to indicate that he is very certain or uncertain about the choices made? All these interface choices impact the quality of the results. They may even impact how easy or hard it is to recruit workers, as tasks that are more enjoyable attract workers.

Finding workers. Human workers for performing our tasks can be found at one of the existing marketplaces such as Mechanical Turk or UpWork. In some cases a private marketplace may be preferable. In such a private crowd, workers are hired by the organization needing the work (or a sub-contractor) to work on crowdsourced problems. One of the key challenges is finding workers that are appropriate for our tasks, especially if the task requires particular skills or knowledge. For example, if we are gathering data on home remedies for illnesses, we need to find workers who have experience with such home remedies. Moreover, for some tasks, the set of workers involved should be appropriate as a whole (e.g., for micro-tasks in terms of accuracy and cost [15], [16], [64] or teams of experts for complex tasks [42]). Some sites such as UpWork allow us to interview potential workers while in others we may need to have them perform some "identification" tasks to learn their skills. We discuss worker evaluation more below.

4.2 Decomposition and Aggregation

The next step is to use the building blocks selected previously (e.g., tasks and interfaces, marketplaces to access crowds) to repeatedly gather enough data until some output condition is reached.

Problem decomposition. Our problem needs to be decomposed into tasks that can be performed by workers. In many cases problems are decomposed into a sequence of *stages*, where each stage involves having a set of tasks performed by workers in parallel. For example, when we discussed search problems we suggested a three stage approach [38], [39]: translate the data need into a set of keyword queries, retrieve product pages, and verify that the products satisfy the original need. The second stage does not involve the crowd, while the first and third stages use different types of crowd tasks and interfaces. Another common pattern is a sequence of "elimination" stages. For example, say we are solving the maximum problem and we have 32 records. In the first stage, we group the records into 16 pairs and perform 16 crowd comparisons. In the next stage, the computer eliminates the 16 losing records, starting a new stage with eight crowd comparisons. We continue in this fashion until we have a single winner, the maximum.

Evaluating current state. An important component of most crowdsourced solutions is what we call the *current state evaluator.* This component combines all the information available at the current time—the crowd answers, external evidence (see Section 4.4), estimates of worker abilities (see Section 4.3)—and answers the questions "If execution were to stop now, what would be the best guess of the solution to our problem, and what would be our confidence in that solution?". The answers to these questions can be used to decide if indeed we can stop execution at this point. In addition, the best guess solution can be helpful in determining what additional evidence to gather in the next stage, i.e., what records to compare next or what data to gather next.

Current state evaluation can be simple in some cases. For instance, say we are augmenting a record with a binary field X that can have a value T or F. At this point we have asked

three workers to provide a value of X, and we have obtained F, F, T. Then our best guess for X (assuming no other information) would be the majority value F. We may also be able to assign a probability that this F value is correct by making some assumptions about the probability that a worked gives a correct answer. What we have done is to compute the *maximum likelihood* solution (F is this case).

In other cases, evaluating the current state can be challenging. For example, if we solve the maximum problem by asking pairwise comparisons, we can view the current state as a graph. In this graph, the nodes are the records, and there is an edge from record A to record B if a worker has determined that B is better than A. Each edge can have a probability of confidence associated with it. As noted earlier, the graph may have contradictory information. For instance, one worker may say that A is better than B, while another says the opposite. We can also have cycles, e.g., one worker says $A \rightarrow B$, another says $B \rightarrow C$ and a third says $C \rightarrow A$. To compute the maximum likelihood solution we can search for the records ordering that is most likely given the evidence, and then pick the record that is better than all records in this ordering.

Selecting next tasks. Another important component decides what tasks to give workers in the next stage, given the current state. Again, the decision can be simple or complex depending on the problem. To illustrate a more complex situation, consider an entity resolution problem where our current best solution is a clustering $\{A, B\}$, $\{C, D, E\}$. (That is, records A and B represent the same entity, the other three records represent a second entity.) If our confidence (or probability) in the $\{A, B\}$ cluster is significantly lower than the other cluster, then we may ask additional workers to compare the A, B pair. Similarly, say the B, C pair has the most similarity among the cross-cluster pairs. Then it may also be a good idea to ask workers to compare that pair (to verify that our decision to place B and C in different clusters is the correct one). The goal in general is to find the next questions that would give us the most "bang for the buck", i.e., the questions that could increase our confidence in our solution the most.

Optimization. So far, we have been describing approaches for determining which tasks to issue next in a "greedy" manner, delivering the most bang-for-the-buck. Ideally, what we would like to do is to globally tune the parameters of the solution to achieve lowest cost, best accuracy, or lowest latency, or some combination of these factors. The optimization problem can be posed in a variety of ways, e.g., (1) given a budget of crowd tasks, how do we achieve the best accuracy or the lowest latency, or (2) given a desired minimum accuracy, how do we minimize the number of tasks required. In many cases it is hard to find the true optimal solution, so we can use a traditional search approach. That is, we postulate a few execution plans, where each plan specifies a particular way of solving the problem. (Plans may differ in the type of tasks they use, the interfaces they use, the way they decompose the problem, the number of workers that perform each task, the way we evaluate workers, and so on.) Then we estimate the cost, accuracy and latency of each execution plan, and finally we select the preferable plan. In more constrained scenarios, where we are given the task type and interface, and we simply have to select some parameters (like the number of stages), it is possible to develop solutions that are provably optimal (for some definition of optimal). See [44], [54] for examples.

4.3 Worker Management

A major issue in any crowdsourcing system is worker quality: Workers can always perform their tasks incorrectly, but there are often workers that incorrectly perform more than their share of tasks. Some of the low quality workers may not have the necessary abilities for the tasks, some may not have adequate training, and some may simply be "spammers" that want to make money without doing much work. Anecdotal evidence indicates that the spammer category is especially problematic, since these workers not only do poor work, but they do a lot of it as they try to maximize their income.

To correct or compensate for poor worker quality, a crowdsourcing system typically implements some type of *worker quality control* (WQC) (e.g., papers [20], [24], [65]). There are two closely related aspects to this worker quality control:

Worker feature extraction. WQC studies and evaluates how workers are performing their tasks, in order to learn features that may be useful for identifying the quality of the worker. For instance, if a worker takes a long time to compete a task, it may be a sign that the worker is poor. In some cases, we may give workers "test" tasks (for which we know the correct answer) in order to evaluate their abilities. Or, we may simply judge how often a worker "disagrees" with other workers as a proxy for how poor the worker is.

Actions to compensate or correct for low quality. Based on the learned features, WQC can take two types of actions: *a)* Actions on Workers. The goal of these actions is to improve the quality of workers in future tasks. For instance, we may avoid hiring workers that have performed poorly in the past. *b)* Actions for Task Result Processing. The goal of these actions is to improve the quality of results for current or past tasks by somehow "downgrading" the results produced by poor workers. For instance, if the result of a task is the average of the scores provided by three workers, then we can use an average weighted by worker quality.

4.4 Prior and External Information

In addition to crowd-provided information that we gather on the fly, we may have prior information available to us in some cases, or information from other automated algorithms. It is in our best interest to use this information, to refine our answers and to better select tasks issued to the crowd.

Computer-generated evidence. It is sometimes useful to think of the crowd responses (answers) as *evidence*. As the system gathers more and more evidence, we are closer to solving our target problem. In some instances, in addition to the crowd evidence, we may also have computer generated evidence. For instance, if we are gathering restaurant data from the crowd, we may also have some pre-existing database of restaurants. This data may not have all the fields we need, but at least it can be helping to seed our crowd gathering. As a second example, in an entity resolution problem, we may be able to compute pairwise similarities of the records based on text and image similarity functions. This computer evidence may help us reduce the number of

comparisons we need to ask the crowd. For example, if a pair of records have a similarity below some threshold, we may rule out that they represent the same entity, and hence we need not ask the crowd to compare those pairs.

Active learning. When we are using crowdsourcing to obtain training data for a machine learning model, we can actually use the model being developed to guide task selection. That is, say we have already obtained from the crowd a partial training set and we have trained the model on that set. We can use that model to estimate a "utility" value for each data item that may be next evaluated by the crowd (yielding our next training value). Then we can use those values to select the next tasks for the crowd. This technique is called active learning [46]. For instance, suppose we are training an email spam classifier, and want to obtain more positive training examples. We can use the existing classifier to get a probability for each email being spam, and then ask workers to evaluate the ones that have the highest probability. The reason why a full integration with machine learning via an active learning algorithm may not always be the best idea is because the data obtained via active learning is biased to the trained model and cannot be reused, e.g., if the training algorithm changes. Typical work in active learning assumes that expert annotations are correct; typical work in crowdsourcing does not make that assumption.

5 Challenges for Future Work

We now describe some challenges that in our opinion have not been adequately addressed in prior work. (Again, our list is not meant to be comprehensive.) We group these under the same four steps we have identified in the previous section.

5.1 Preparation and Initialization

Interface design and testing. The design choices that go into a particular interface a worker sees can significantly improve worker satisfaction and efficiency, and at the same time can minimize worker errors. Making the appropriate design choices is challenging, requiring that designers take into account factors such as pop-out [59] and anchoring [27]. Industry users of crowdsourcing have called interface design a *dark art* [4]. A lot more research is needed to understand the available interface choices and their impact on performance and accuracy.

Only Mechanical Turk. Most current research on data crowdsourcing solutions is typically tailored to and validated on a Mechanical Turk-like marketplace. There are many marketplaces in existence, and further, there are many marketplaces that are internal to organizations [31]. The developed algorithms do not directly apply to these marketplaces, since they obey very different characteristics. For example, on UpWork [3], it is common to pay workers per hour instead of per task; in internal marketplaces, it is typical to hire workers from 9-5 and have them work on tasks non-stop, meaning that workers are resources that are available at any time. Designing crowdsourcing algorithms that are tailored for different platforms is an important direction for future work.

Fluidity in marketplaces. The composition of the workers in a marketplace varies widely with the time of day, as well as the day of the week, month, and year. For instance, studies have reported getting vastly different accuracies and latencies for their tasks if they deploy the tasks at different times during the day. This is probably due to the shifting of the demographic constitution of the marketplace. Anecdotal evidence also suggests that these effects are periodic: latencies on weekends are much higher than latencies during the week. Furthermore, the accuracies and latencies have also changed in various marketplaces over the years. These factors certainly affect the cost, accuracy, and latency of crowd-powered algorithms. Further, these factors make it challenging to benchmark, compare, and validate the performance of crowd-powered algorithms.

5.2 Decomposition and Aggregation

Difficulty. In practice, not all tasks issued to crowd workers are equally easy or difficult. For example, comparing two items that are very similar is a lot harder than comparing two items that are far from one another. Furthermore, difficulty is related to worker ability, e.g., what is hard for one worker may be easy for another. For sorting, there has been a limited exploration of difficulty, e.g., [8], [19], precisely capturing the idea described above. There have also been some empirical studies demonstrating how difficulty can affect error rates. In one empirical study [52], the number of dots in an image is varied, and task involves counting the number of dots. As the number of dots increases, the error rate of the estimate provided by crowd workers also increases.

Unfortunately for algorithm designers, the difficulty of a task and the ability of a worker are typically not known upfront. We can only infer that a task is difficult if workers end up disagreeing on its answer, and we can only infer that a worker has ability if she gets correct (majority) answers. Of course, to infer difficulty and ability this way we need to assume worker independence. If workers make correlated errors, then this is a lot harder to detect and automatically correct for task difficulty. It is even harder to provide theoretical performance guarantees when difficulty is taken into account.

Batching. Typically, on marketplaces like Mechanical Turk, requesters will batch a set of questions about different items into a single task. This has multiple advantages: by batching, the requesters can pay a higher amount, and therefore attract workers; workers can read instructions once and provide answers to a set of tasks all in one go; and lastly, workers have a bit more context as to the diversity of tasks. For example, if the goal is to rate a set of photos on a scale from 1-5 on how picturesque they are, having some context by providing additional photos can help crowd workers better gauge individual photos.

Unfortunately, batching also introduces bias, especially when datasets are skewed: if the true rating of each photo in a set of photos displayed to a crowd worker is 1, then the worker may incorrectly mark a few as 2 or 3 because they do not expect all of them to be 1. To the best of our knowledge, there has very little work, apart from [66], on trying to study the impact of and correct for batching bias in crowd algorithms.

Insufficient optimization. Most work on data crowdsourcing has limited optimization, typically either restricted to one stage or one step, as opposed to across multiple steps.

The issues stem from the fact that it is still unclear how to best involve crowds within algorithms; in fact, even if we fix the set of tasks or interfaces that we can use, it is still not straightforward.

5.3 Worker Management

Worker identities. Most work on data crowdsourcing assumes that workers are all essentially alike, in that their abilities, speed and error rates are similar. This is certainly not true in practice; workers have very different abilities, with different abilities and processing speeds for different types of tasks, and taking this into account can help us route tasks to appropriate workers.

Boredom, laziness, and experience. Most data crowdsourcing solutions do not take into account many human-specific factors. For example, human workers will often avoid doing work if they can; they will "satisfice" [45] by doing the least amount of work necessary. Workers have been known to answer questions without reading instructions completely, or by randomly guessing. Workers are also affected by boredom and fatigue, especially when they have answered many questions of the same type. In addition, workers often get more effective at tasks over time: typical crowdsourcing algorithms do not take experience into account. Lastly, we have heard reports of human workers colluding to all provide the same answer for a task [51]: collusion is very hard to detect automatically and correct for (typical algorithms assume independence between workers).

Biases. Bias is another important factor affecting crowd-powered algorithms. For example, if we have workers answering rating questions, e.g., rate a photo from 1-5, there will be workers who are more "stingy" or more "lenient". Taking into account these biases and correcting for them is important. While there has been some work in taking individual worker biases into account [25], [26], [61], many papers in crowdsourcing algorithms assume a uniform error rate for all workers, thereby ignoring all worker-specific bias.

Incentivization. While there has been some work in developing appropriate incentive mechanisms [22] for crowd workers, these learnings have not been integrated into the design of crowd-powered algorithms. Appropriate incentivization is important to ensure not just that workers are compensated according to the level of effort, but also entice them to put in more effort and not simply randomly guess.

5.4 Prior and External Information

Prior information. In many cases, we may be able to use simple automated mechanisms (such as machine learning algorithms) to give us prior probabilities for various tasks (e.g., crowd entity resolution [53], [57], [58], [60]). For example, for content moderation of images (i.e., checking if images depict content inappropriate for children), we may be able to use color histograms to detect skin tone to give a prior probability for different images. Not a lot of crowdsourcing papers take prior information into account. One such paper [36] treats the prior probability as just another worker with a known probability of error. Other ways of reasoning about and incorporating prior information into crowd results may be more powerful and may lead to more benefits.

Integration with active learning. There has not been enough crowdsourcing work trying to integrate the learning of a good machine learning model while at the same time performing the tasks at hand (with desired accuracy or cost). There has been initial work in this space [35], [62], but much more remains to be done. Overall, it remains to be seen what is the best use of crowdsourcing: as one-shot training data for machine learning algorithms, in an integrated format with active learning algorithms, as verifiers for the output of machine learning algorithms, or some combination.

In closing, crowdsourcing can be a valuable tool for improving the quality and usefulness of our data, combining the best of computer data management and analysis techniques with the power of "natural intelligence." However, a lot of careful thought must go into the design of a crowdsourced application, and there are still many challenging issues that need to be resolved.

References

[1] CrowdFlower (Retrieved 22 July 2013) [Online]. Available: http://crowdflower.com

[2] Duolingo Inc. (Retrieved 14 August 2013) [Online]. Available: http://www.duolingo.com

[3] Elance-oDesk (Retrieved 16 March 2015) [Online]. Available: http://elance-odesk.com/

[4] Microsoft Wants to Turn Crowdsourcing from an Art to a Science (Retrieved 10 January 2014) [Online]. Available: http://www.crowdsourcing.org/editorial/microsoft-wants-to-turn-crowdsourcing-from-an-art-into-a-science/21233

[5] Samasource (Retrieved 22 July 2013) [Online]. Available: http://samasource.com

[6] StitchFix (Retrieved 5 August 2015) [Online]. Available: https://www.stitchfix.com//

[7] S. Ahmad, A. Battle, Z. Malkani, and S. D. Kamvar, "The jabberwocky programming environment for structured social computing," in *Proc. 24th Annu. ACM Symp. User Interface Softw. Technol.*, 2011, pp. 53–64.

[8] M. Ajtai, V. Feldman, A. Hassidim, and J. Nelson, "Sorting and selection with imprecise comparisons," in *Proc. 36th Int. Colloq. Automata, Languages Program.: Part I*, 2009, no. 1, pp. 37–48.

[9] Y. Amsterdamer, Y. Grossman, T. Milo, and P. Senellart, "Crowd mining," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 241–252.

[10] Y. Amsterdamer, A. Kukliansky, and T. Milo, "NL2CM: A natural language interface to crowd mining," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 1433–1438.

[11] A. Anagnostopoulos, L. Becchetti, A. Fazzone, I. Mele, and M. Riondato, "The importance of being expert: Efficient max-finding in crowdsourcing," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 983–998.

[12] D. W. Barowy, C. Curtsinger, E. D. Berger, and A. McGregor, "AutoMan: A platform for integrating human-based and digital computation," in *Proc. ACM Int. Conf. Object Oriented Programm. Syst. Languages Appl.*, 2012, pp. 639–654.

[13] K. Bellare, S. Iyengar, A. G. Parameswaran, and V. Rastogi, "Active sampling for entity matching," in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2012, pp. 1131–1139.

[14] M. S. Bernstein, G. Little, R. C. Miller, B. Hartmann, M. S. Ackerman, D. R. Karger, D. Crowell, and K. Panovich, "Soylent: A word processor with a crowd inside," in *Proc. 23rd Annu. ACM Symp. User Interface Softw. Technol.*, 2010, pp. 313–322.

[15] C. C. Cao, J. She, Y. Tong, and L. Chen, "Whom to ask? jury selection for decision making tasks on micro-blog services," in *Proc. VLDB Endowment*, vol. 5, no. 11, pp. 1495–1506, 2012.

[16] C. C. Cao, Y. Tong, L. Chen, and H. V. Jagadish, "Wisemarket: A new paradigm for managing wisdom of online social users," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2013, pp. 455–463.

[17] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye, "Katara: A data cleaning system powered by knowledge bases and crowdsourcing," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 1247–1261

[18] N. N. Dalvi, R. Kumar, B. Pang, R. Ramakrishnan, A. Tomkins, P. Bohannon, S. Keerthi, and S. Merugu, "A web of concepts," in *Proc. 28th ACM SIGMOD-SIGACT-SIGART Symp. Principles Database Syst.*, 2009, pp. 1–12.

[19] S. B. Davidson, S. Khanna, T. Milo, and S. Roy, "Using the crowd for top-k and group-by queries," in *Proc. 16th Int. Conf. Database Theory*, 2013, pp. 225–236.

[20] J. Fan, G. Li, B. C. Ooi, K.-l. Tan, and J. Feng, "iCrowd: An adaptive crowdsourcing framework," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 1015–1030.

[21] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin, "CrowdDB: Answering queries with crowdsourcing," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2011, pp. 61–72.

[22] A. Ghosh, "Game theory and incentives in human computation systems," in *Handbook of Human Computation*. New York, NY, USA: Springer, 2013.

[23] S. Guo, A. G. Parameswaran, and H. Garcia-Molina, "So who won?: Dynamic max discovery with the crowd," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2012, pp. 385–396.

[24] N. Q. V. Hung, D. C. Thang, M. Weidlich, and K. Aberer, "Minimizing efforts in validating crowd answers," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 999–1014.

[25] M. Joglekar, H. Garcia-Molina, and A. Parameswaran, "Comprehensive and reliable crowd assessment algorithms," in *Proc. IEEE 31st Int. Conf. Data Eng.*, 2015, pp. 195–206.

[26] D. R. Karger, S. Oh, and D. Shah, "Efficient crowdsourcing for multi-class labeling," in *Proc. ACM SIGMETRICS/Int. Conf. Meas. Model. Comput. Syst.*, 2013, pp. 81–92.

[27] A. Kittur, E. H. Chi, and B. Suh, "Crowdsourcing user studies with mechanical turk," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2008, pp. 453–456.

[28] A. Kittur, B. Smus, and R. Kraut, "CrowdForge: Crowdsourcing complex work," in *Proc. CHI Extended Abstracts*, 2011, pp. 1801–1806.

[29] A. P. Kulkarni, M. Can, and B. Hartmann, "Collaboratively crowdsourcing workflows with turkomatic," in *Proc. Conf. Comput. Supported Cooperative Work*, 2012, pp. 1003–1012.

[30] G. Little, L. B. Chilton, M. Goldman, and R. C. Miller, "TurKit: Human computation algorithms on mechanical turk," in *Proc. 23rd Annu. ACM Symp. User Interface Softw. Technol.*, 2010, pp. 57–66.

[31] A. Marcus and A. Parameswaran, "Crowdsourced data management: Industry and academic perspectives," *Found. Trends Databases*, vol. 6, nos. 1/2, pp. 1–161, 2015.

[32] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller, "Human-powered sorts and joins," *Proc. VLDB Endowment*, vol. 5, no. 1, pp. 13–24, 2011.

[33] A. Marcus, E. Wu, S. Madden, and R. C. Miller, "Crowdsourced databases: Query processing with people," in *Proc. 5th Biennial Conf. Innovative Data Syst. Res.*, 2011, pp. 211–214.

[34] M. R. Morris and J. Teevan, *Collaborative Web Search: Who, What, Where, When, and Why*. San Rafael, CA, USA: Morgan & Claypool Publishers, 2009.

[35] B. Mozafari, P. Sarkar, M. J. Franklin, M. I. Jordan, and S. Madden, "Active learning for crowd-sourced databases," *arXiv preprint arXiv:1209.3686*, 2012.

[36] A. Parameswaran, S. Boyd, H. Garcia-Molina, A. Gupta, N. Polyzotis, and J. Widom. (2013). Optimal crowd-powered rating and filtering algorithms. Stanford Univ., Stanford, CA, USA, Tech. Rep. http://ilpubs.stanford.edu:8090/1078/

[37] A. G. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom, "Crowdscreen: Algorithms for filtering data with humans," in *Proc. SIGMOD Conf.*, 2012, pp. 361–372.

[38] A. G. Parameswaran, M. H. Teh, H. Garcia-Molina, and J. Widom, "DataSift: An expressive and accurate crowd-powered search toolkit," presented at the 1st AAAI Conf. Human Comput. Crowdsourcing, Palm Springs, CA, USA, Nov. 7–9, 2013.

[39] A. G. Parameswaran, M. H. Teh, H. Garcia-Molina, and J. Widom, "Datasift: A crowd-powered search toolkit," in *Proc. Int. Conf. Manage. Data*, Snowbird, UT, USA, Jun. 22–27, 2014, pp. 885–888.

[40] H. Park, R. Pang, A. Parameswaran, H. Garcia-Molina, N. Polyzotis, and J. Widom, "An overview of the deco system: Data model and query language; query processing and optimization," *ACM SIGMOD Rec.*, no. 41, no. 4, Dec. 2012.

[41] M. J. Raddick, G. Bracey, P. L. Gay, C. J. Lintott, P. Murray, K. Schawinski, A. S. Szalay, and J. Vandenberg. (2009, Sep.). Galaxy zoo: Exploring the motivations of citizen science volunteers [Online]. Available: http://arxiv.org/abs/0909.2925

[42] D. Retelny, S. Robaszkiewicz, A. To, W. S. Lasecki, J. Patel, N. Rahmati, T. Doshi, M. Valentine, and M. S. Bernstein, "Expert crowdsourcing with flash teams," in *Proc. 27th Annu. ACM Symp. User Interface Softw. Technol.*, 2014, pp. 75–85,

[43] A. D. Sarma, A. Jain, A. Nandi, A. Parameswaran, and J. Widom. (2015). Surpassing humans and computers with jellybean: Crowd-vision-hybrid counting algorithms. Stanford Univ., Stanford, CA, USA, Tech. Rep. [Online]. Available: http://ilpubs.stanford.edu:8090/1128/

[44] A. D. Sarma, A. Parameswaran, H. Garcia-Molina, and A. Halevy, "Crowd-powered find algorithms," in *Proc. IEEE 30th Int. Conf. Data Eng.*, 2014, pp. 964–975.

[45] L. Schmidt, "Crowdsourcing for human subjects research," in *Proc. CrowdConf.*, 2010.

[46] B. Settles, *Active Learning*, Synthesis Lectures on Artificial Intelligence and Machine Learning. San Rafael, CA, USA: Morgan & Claypool Publishers, 2012.

[47] V. S. Sheng, F. J. Provost, and P. G. Ipeirotis, "Get another label? improving data quality and data mining using multiple, noisy labelers," in *Proc. 14th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2008, pp. 614–622.

[48] A. Singhal, "Introducing the knowledge graph: Things, not strings," *Official Google Blog* , May, 2012.

[49] R. Snow, B. O'Connor, D. Jurafsky, and A. Y. Ng, "Cheap and fast—but is it good?: Evaluating non-expert annotations for natural language tasks," in *Proc. Conf. Empirical Methods Natural Language Process.*, 2008, pp. 254–263.

[50] C. Sun, N. Rampalli, F. Yang, and A. Doan, "Chimera: Large-scale classification using machine learning, rules, and crowdsourcing," *Proc. VLDB Endowment*, vol. 7, no. 13, pp. 1529–1540, 2014.

[51] B. Trushkowsky, T. Kraska, M. J. Franklin, and P. Sarkar, "Crowdsourced enumeration queries," in *Proc. IEEE 29th Int. Conf. Data Eng.*, 2013, pp. 673–684.

[52] P. Venetis and H. Garcia-Molina. (2012, Aug.). Dynamic max algorithms in crowdsourcing environments. Stanford Univ., Stanford, CA, USA, Tech. Rep. [Online]. Available: http://ilpubs.stanford.edu:8090/1050/

[53] V. Verroios and H. Garcia-Molina, "Entity resolution with crowd errors," in *Proc. 31st IEEE Int. Conf. Data Eng.*, Seoul, South Korea, Apr. 13–17, 2015, pp. 219–230.

[54] V. Verroios, P. Lofgren, and H. Garcia-Molina, "tdp: An optimal-latency budget allocation strategy for crowdsourced MAXIMUM operations," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Melbourne, Vic., Australia, May 31–Jun. 4, 2015, pp. 1047–1062.

[55] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford, "Captcha: Using hard ai problems for security," in *Proc. Int. Conf. Theory Appl. Cryptographic Techn.*, 2003, pp. 294–311.

[56] L. von Ahn and L. Dabbish, "Designing games with a purpose," *Commun. ACM*, vol. 51, no. 8, pp. 58–67, 2008.

[57] J. Wang, T. Kraska, M. J. Franklin, and J. Feng, "CrowdER: Crowdsourcing Entity Resolution," *Proc. VLDB Endowment*, vol. 5, pp. 1483–1494, 2012.

[58] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng, "Leveraging transitive relations for crowdsourced joins," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 229–240,

[59] Q. Wang, P. Cavanagh, and M. Green, "Familiarity and pop-out in visual search," *Perception Psychophys.*, no. 56, no 5, pp. 495–500, 1994.

[60] S. E. Whang, P. Lofgren, and H. Garcia-Molina, "Question selection for crowd entity resolution," in *Proc. VLDB Endowment*, no. 6, pp. 349–360, Apr. 2013.

[61] T. Wu, L. Chen, P. Hui, C. J. Zhang, and W. Li, "Hear the whole story: Towards the diversity of opinion in crowdsourcing markets," *Proc. VLDB Endowment*, vol. 8, no. 5, pp. 485–496, 2015.

[62] Y. Yan, R. Rosales, G. Fung, and J. G. Dy, "Active learning from crowds," in *Proc. 28th Int. Conf. Mach. Learn.*, Bellevue, WA, USA, Jun. 28–Jul. 2, 2011, pp. 1161–1168.

[63] C. J. Zhang, L. Chen, Y. Tong, and Z. Liu, "Cleaning uncertain data with a noisy crowd," in *Proc. 31st IEEE Int. Conf. Data Eng.*, Seoul, South Korea, Apr. 13–17, 2015, pp. 6–17.

[64] Y. Zheng, R. Cheng, S. Maniu, and L. Mo, "On optimality of jury selection in crowdsourcing," in *Proc. 18th Int. Conf. Extending Database Technol.*, 2015, pp. 193–204.

[65] Y. Zheng, J. Wang, G. Li, R. Cheng, and J. Feng, "QASCA: A quality-aware task assignment system for crowdsourcing applications," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 1031–1046.

[66] H. Zhuang, A. Parameswaran, D. Roth, and J. Han, "Debiasing crowdsourced batches," *Proc. 21st ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2015, pp. 1593–1602.

Hector Garcia-Molina received the BS degree in electrical engineering from the Instituto Tecnologico de Monterrey, Mexico, in 1974. He received the MS degree in electrical engineering in 1975 and the PhD degree in computer science in 1979 from Stanford University, Stanford, CA. He holds an honorary PhD degree from ETH Zurich in 2007. He is the Leonard Bosack and Sandra Lerner professor in the Departments of Computer Science and Electrical Engineering, Stanford University, Stanford, CA. He was the chairman in the Computer Science Department from January 2001 to December 2004. From 1997 to 2001, he was a member the President's Information Technology Advisory Committee (PITAC). From August 1994 to December 1997, he was the director in the Computer Systems Laboratory at Stanford. From 1979 to 1991, he was on the faculty of the Computer Science Department, Princeton University, Princeton, NJ. His research interests include distributed computing systems, digital libraries, and database systems. He is a fellow of the Association for Computing Machinery and of the American Academy of Arts and Sciences. He is a member of the National Academy of Engineering; received the 1999 ACM SIGMOD Innovations Award. He is a Venture Advisor for Onset Ventures, a member of the Board of Directors of Oracle, and a member of the State Farm Technical Advisory Council. He is a member of the IEEE.

Manas Joglekar received the BTech degree in computer science and engineering from the Indian Institute of Technology Bombay, in 2011. He is a fifth year PhD student in the Computer Science Department, Stanford University. His advisor is Prof. Hector Garcia-Molina. His research interests include crowdsourcing, data quality evaluation, and database theory.

Adam Marcus received the PhD degree in computer science from MIT in 2012, where his dissertation was on database systems and human computation. He is the cofounder of Unlimited Labs, a company dedicated to the future of how creative and analytical people do work. Prior to that, he led the data team at Locu, a startup that was acquired by GoDaddy. He received the US National Science Foundation (NSF) and NDSEG fellowships, and has previously worked at ITA, Google, IBM, and FactSet. In his free time, he builds course content to get people excited about data and programming.

Aditya Parameswaran received the PhD degree from Stanford University, advised by Prof. Hector Garcia-Molina. After receiving the PhD degree, he visited MIT CSAIL and Microsoft Research New England in 2013-2014. He is an assistant professor of computer science at the University of Illinois (UIUC). He is broadly interested in data analytics, with research results in human computation, visual analytics, information extraction and integration, and recommender systems. He received the Arthur Samuel award for the best dissertation in CS at Stanford (2014), the SIGMOD Jim Gray dissertation award (2014), the SIGKDD dissertation award runner up (2014), a Google Faculty Research Award (2015), the Key Scientific Challenges Award from Yahoo! Research (2010), three best-of-conference citations (VLDB 2010, KDD 2012 and ICDE 2014), the Terry Groswith graduate fellowship at Stanford (2007), and the Gold Medal in Computer Science at IIT Bombay (2007). His research group is supported with funding from by the NIH, the US National Science Foundation (NSF), and Google.

Vasilis Verroios received the BS and MS degrees in computer science from the University of Athens, in 2006 and 2008, respectively. He is currently working toward the PhD degree in the Computer Science Department, Stanford University. His advisor is Hector Garcia-Molina. In the past, he has been a member of the "Management of Data, Information, & Knowledge Group" in the University of Athens, and he has worked at oDesk and Microsoft Research. His primary interests include crowdsourcing, data mining/exploration, and pervasive computing.