

Foundations and Trends® in Databases
Vol. 6, No. 1-2 (2013) 1–161
© 2015 A. Marcus and A. Parameswaran
DOI: 10.1561/19000000044



Crowdsourced Data Management: Industry and Academic Perspectives

Adam Marcus
Unlimited Labs
marcua@marcua.net

Aditya Parameswaran
University of Illinois (UIUC)
adityagp@illinois.edu

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 2 |
| 1.1 | Chapter Summaries | 6 |
| 1.2 | Crowdsourcing Background | 8 |
| 1.3 | Crowdsourcing Best Practices | 14 |
| 1.4 | Assumptions in this Book | 16 |
| 2 | Related Work | 17 |
| 2.1 | Surveys | 18 |
| 2.2 | Human Computer Interaction | 18 |
| 2.3 | Machine Learning and Artificial Intelligence | 20 |
| 2.4 | Social Science | 23 |
| 2.5 | Game Theory and Pricing | 24 |
| 2.6 | Systems and Programming Models | 25 |
| 3 | An Overview of Crowd-Powered Algorithms | 28 |
| 3.1 | Crowd Workers As Data Processors | 29 |
| 3.2 | Executive Summary | 31 |
| 3.3 | A Survey of Crowd-Powered Algorithms | 32 |
| 3.4 | Design Choices for Crowd-Powered Algorithms | 36 |
| 3.5 | Optimization Objectives | 40 |
| 3.6 | Other Confounding Factors | 41 |

| | | |
|----------|--|------------|
| 3.7 | Summary | 44 |
| 4 | An Overview of Crowd-Powered Systems | 45 |
| 4.1 | Executive Summary | 47 |
| 4.2 | Summary of Three Declarative Systems | 49 |
| 4.3 | Design Decisions | 57 |
| 4.4 | Unresolved Issues | 61 |
| 4.5 | Summary | 63 |
| 5 | Survey of Industry Users: Summary and Methodology | 64 |
| 5.1 | Executive summary | 65 |
| 5.2 | Survey and recruitment methodology | 67 |
| 5.3 | Three personas of industry users | 69 |
| 6 | Survey of Industry Users: Crowd Statistics and Management | 71 |
| 6.1 | Size of systems-building teams | 72 |
| 6.2 | Task throughput | 73 |
| 6.3 | Monetary spending | 74 |
| 6.4 | Task redundancy | 75 |
| 6.5 | Recruiting | 75 |
| 6.6 | Crowd worker tenure | 77 |
| 7 | Survey of Industry Users: Use cases and Prior Approaches | 79 |
| 7.1 | Use Cases | 79 |
| 7.2 | Prior approaches | 89 |
| 7.3 | Benefits of crowdsourcing | 91 |
| 8 | Survey of Industry Users: Task Quality, Worker Incentives, and Workflow Decomposition | 94 |
| 8.1 | Ensuring Quality | 94 |
| 8.2 | Incentives | 101 |
| 8.3 | Task design | 104 |
| 9 | Survey of Marketplace Providers of Crowdsourcing | 111 |
| 9.1 | Executive Summary | 112 |
| 9.2 | Marketplace Details | 114 |

| | |
|---------------------------------------|------------|
| 9.3 Implementations | 118 |
| 9.4 Quality Control | 123 |
| 10 Conclusion | 127 |
| Acknowledgements | 130 |
| Appendices | 131 |
| A Industry Users Survey | 132 |
| B Marketplace Providers Survey | 138 |
| References | 144 |

Abstract

Crowdsourcing and human computation enable organizations to accomplish tasks that are currently not possible for fully automated techniques to complete, or require more flexibility and scalability than traditional employment relationships can facilitate. In the area of data processing, companies have benefited from crowd workers on platforms such as Amazon's Mechanical Turk or Upwork to complete tasks as varied as content moderation, web content extraction, entity resolution, and video/audio/image processing. Several academic researchers from diverse areas ranging from the social sciences to computer science have embraced crowdsourcing as a research area, resulting in algorithms and systems that improve crowd work quality, latency, or cost. Given the relative nascence of the field, the academic and the practitioner communities have largely operated independently of each other for the past decade, rarely exchanging techniques and experiences. In this book, we aim to narrow the gap between academics and practitioners. On the academic side, we summarize the state of the art in crowd-powered algorithms and system design tailored to large-scale data processing. On the industry side, we survey 13 industry users (e.g., Google, Facebook, Microsoft) and 4 marketplace providers of crowd work (e.g., CrowdFlower, Upwork) to identify how hundreds of engineers and tens of million dollars are invested in various crowdsourcing solutions. Through the book, we hope to simultaneously introduce academics to real problems that practitioners encounter every day, and provide a survey of the state of the art for practitioners to incorporate into their designs. Through our surveys, we also highlight the fact that crowd-powered data processing is a large and growing field. Over the next decade, we believe that most technical organizations will in some way benefit from crowd work, and hope that this book can help guide the effective adoption of crowdsourcing across these organizations.

1

Introduction

We are drowning in information, while starving for wisdom.

— E. O. Wilson

With the advent of the “data deluge” [176], organizations world-wide have been struggling with designing algorithms and systems to better process and analyze the massive quantities of data collected every day. It is estimated that 80% of this data is unstructured [205, 196], consisting largely of images, videos, and raw text. While there have been significant advances in automated mechanisms for interpreting and extracting information from unstructured data, algorithms to fully comprehend unstructured data have not been developed yet. It is widely acknowledged that we are at least several decades away from this goal [162, 120].

Humans, on the other hand, are able to analyze certain aspects of unstructured data with relative ease. Humans have an innate understanding of language, speech, and images; they are able to process, reason about, and provide solutions to problems faced often in managing and processing unstructured data. Moreover, the abundance of cheap and reliable internet connectivity throughout the world has given rise to *crowdsourcing* or *crowd work* marketplaces, such as Mechanical Turk [10] and Upwork [17], enabling the inclusion of human crowd workers in on-demand data processing tasks.

In particular, crowdsourcing has been applied in the following large-scale unstructured data processing applications (among others):

- **Content Moderation.** Workers in crowdsourcing marketplaces are often consulted for content moderation of images uploaded on web sites [5]. That is, humans are asked to determine whether user-uploaded images are appropriate for viewing by a general audience.
- **Web Extraction.** Crowd workers also contribute to tasks like information extraction from web sites. That is, workers are asked to provide specific information by looking up web sites and finding, say, phone numbers or prices at restaurants [91]. Workers can aid machines in semi-automatic information extraction systems—for instance, companies like Yahoo! [18] use crowdsourcing to build web extraction wrappers, and to verify extracted information [40, 87, 88, 142, 60].
- **Search Relevance.** Most companies with a search engine, e.g., Bing [11], Google [9], and Yahoo!, include crowd workers in evaluating the performance of their search algorithms [26].
- **Entity Resolution.** Entity Resolution, or deduplication [78] refers to the problem of identifying if two textual records refer to the same entity. Groupon and Yahoo! both use crowdsourcing for entity resolution [105, 104, 34].
- **Text Processing.** Crowdsourcing is used in spam identification [137], text classification [30, 172], translation [199], and text editing [36]. Crowdsourcing is also being used commercially for transliteration of documents [20].
- **Video and Image Processing.** Crowdsourcing is used in video analysis [53], for image labeling [160, 185], and as a visual aid [39].

Unfortunately, in all of these applications, and overall, crowdsourcing can be subjective or error-prone; it can be time-consuming (crowd workers take longer than computers); and it can be relatively costly (human workers need to be paid). Moreover, these three aspects—accuracy, latency, and cost—are

correlated in complex ways, making it difficult to optimize the trade-offs among them while designing data processing algorithms and systems.

As an example of these tradeoffs, consider content moderation of images. We can ask one human worker to verify if each image is appropriate, but they may make mistakes. As a result, we may need to ask multiple humans to verify each image. However, asking multiple human workers has higher monetary cost, and might incur higher latency. Furthermore, we can ask multiple human workers to verify each image in parallel, or ask humans in sequence. The former option can incur lower latency, while the latter might have lower monetary cost since we can choose to not ask subsequent questions based on worker agreement on answers to previous ones.

With nearly a decade passing since crowdsourcing marketplaces have become commonplace, academic researchers and industry users alike have explored various mechanisms for orchestrating large scale data processing work by assembling human workers in workflows that attempt to optimize the three aspects described above (accuracy, latency, and cost), while also expanding our understanding of what is actually feasible using human workers. On the one hand, academic researchers have proposed programming languages, frameworks, systems, and algorithms, and have prototyped creative solutions to problems that are just now feasible to solve with the advent of crowdsourcing. On the other hand, several companies have been founded whose core business is to explore the use of crowd work for various “unsolvable” tasks, and many companies have embraced crowd work as a mechanism for accomplishing what was previously infeasible or inefficient.

However, *progress in academia and industry on how to best leverage crowd work for large scale data processing has largely proceeded independently*. It is essential that these two communities work in concert with one another. Industrial users and marketplace providers have a lot of wisdom to share about the problems that are the most crucial to solve, which techniques work well in practice and which don’t, as well as “best-practice” implementations of workflows involving crowds. Academia has much to say about how to leverage large scale data processing in an optimized fashion in many settings.

The primary goal of our book is to bridge the gap between crowdsourcing practitioners and academic crowdsourcing researchers. With this goal in mind, we will:

- summarize the state of the art in research on crowd-powered algorithms and systems for data processing, and
- survey industry users and marketplace providers of crowd work to identify their accomplishments and highlight the unsolved problems they struggle with.

By describing the state-of-the-art in crowd-powered data processing from academia, we hope to provide a reference for industry participants to see if academia have solved their problems, and to articulate the areas that have the most potential for future research. By engaging industry users and marketplace vendors, we hope to highlight their chief pain-points and concerns, identify the status quo, and articulate which areas of future research have the most potential for impact. Identifying the “tried-and-true” methods that work well in industry settings that are yet to be formally analyzed in academia would also be valuable for academics. Furthermore, industry and marketplace vendors can see if they all face the same challenges, or if other industry or marketplace participants have solved the problems that they face.

Overall, by connecting the marketplace providers, industry users, and academia, we hope that these groups are educated about the problems and solutions that each of them has been working on, in order to facilitate more transparency, more openness, and also the ability to begin a frank dialog about the problems and the future of crowdsourcing.

A secondary goal of this book is to argue that *crowdsourcing is here to stay*. A common criticism in academia is that crowdsourcing is a fad; that not too many industry users care about crowdsourcing; and that the recent interest in crowdsourcing is going to disappear in a few years. Our thesis is that this is simply not the case. As we will find out in the industry portions of this book, crowdsourcing is *an essential ingredient for any company working with large datasets*. Companies are sometimes not willing to talk about how much they use crowdsourcing because they are either ashamed about admitting that they rely on crowds instead of sophisticated software or hardware, or paradoxically because they consider it to be their “secret sauce.” Through our conversations with industry users, we will highlight the hundreds of employees and tens of millions of dollars that companies invest into crowd work.

A reader might note that in our coverage of industry users and marketplace providers of crowdsourcing, we do not dedicate attention to an impor-

tant third group in crowd work: the crowd workers themselves. We first note that the study of crowd workers is relatively well-explored, with several seminal and ongoing surveys of different crowds over time [161, 99, 177, 165]. Second, our focus in this study is on the gap between industry and academia, especially as it relates to large-scale data processing, and we did not view workers as having a large influence on this gap. Understanding and designing for crowd workers is of utmost importance for the health and future of crowd work, but given the existing studies of the crowd and our specific research aims, it will not be the focus of our attention.

1.1 Chapter Summaries

We have structured the book into the following chapters¹:

- **Background (Remainder of this chapter).** To establish fluency in crowdsourcing or crowd work, we present the lifecycle of an example task, touching on terminology we will use throughout the book.
- **Related work (Chapter 2).** The research literature has over half a decade of contributions on various aspects of crowdsourcing, and we summarize many of the fields and papers that have influenced crowd-powered data processing.
- **Crowd-powered algorithms (Chapter 3).** At its core, data processing relies on a set of algorithms to filter, sort, summarize, categorize, enumerate, and join datasets. In this chapter, we summarize the state of the art of making these algorithms crowd-powered, and highlight some core models and considerations for crowd-powered algorithm design.
- **Crowd-powered systems (Chapter 4).** Some of the earliest contributions to crowd-powered data processing research were database systems that integrated the concept of humans to optimize and perform

¹As you explore the chapters, keep in mind that crowd-powered data processing is an active and fast-moving field. As new developments arise, we hope to make updates. If you disagree with anything in the book, or if you as an industry user or marketplace provider wish to tell us about how this book compares or contrasts with your experiences with crowd work, please reach out to us at marcua@marcua.net and adityagp@illinois.edu.

data processing. We summarize these key systems (CrowdDB, Deco, and Qurk), and identify their approaches to facilitating declarative data processing.

- **Industry user survey: summary (Chapter 5).** To get an industry perspective, we survey 13 industry users of crowd work ranging from large Fortune 500 companies to small single-purpose startups. While we find both creative and common uses, and best-practices around crowd work, we also identify several areas for future research and development. In this chapter, we describe our methodology and participants, and summarize our key findings.
- **Survey of industry users: crowd statistics and management (Chapter 6).** Some of our participants have invested tens of millions of dollars into thousands of crowd workers and dozens of full-time employees to refine their crowd-powered data processing workflows. In this chapter, we provide summary statistics describing the scope of these operations and their management.
- **Survey of industry users: use cases and prior approaches (Chapter 7).** To better understand the benefit of crowd work, we ask participants what their crowd-powered data processing use cases are. We also ask them to describe prior approaches, if they existed, to solving these problems.
- **Survey of industry users: task quality, worker incentives, and workflow decomposition (Chapter 8).** We conclude our industry survey by summarizing various design and implementation decisions that participants told us about. Specifically, we summarize participants' approaches to managing quality, worker incentives, and task decomposition. One key learning was that the most advanced approaches coming out of academia do not appear to be making their way into industry.
- **Marketplace provider survey (Chapter 9).** We survey four of the largest marketplaces that connect crowd workers and industry users to understand their view of the market. The four providers differ significantly in their methods, scope, and scale, resulting in very different use

cases, approaches, and problems. We shed light on the problems facing marketplace providers, which are not always the same as those facing industry users.

1.2 Crowdsourcing Background

In this section, we describe the basic concepts underlying crowd work, and define some common terms we will use throughout the book. We follow this with a short introduction to crowdsourcing and crowdsourcing marketplaces using an example task.

1.2.1 Fundamental Concepts

There are many conflicting opinions [153] on how to define crowdsourcing, and whether crowdsourcing is indeed the same concept as *human computation*. We avoid this debate by relying on a paired definition of crowdsourcing and human computation:

From Luis Von Ahn’s Ph.D. Thesis [182]: “*Crowdsourcing (or Human Computation) is a paradigm that utilizes human processing power to solve problems that computers cannot yet solve.*”

We often use crowd work instead of crowdsourcing or human computation, which also refers to the same concept: using human input to solve problems.

We now describe how we can leverage crowd work. Crowd work typically operates via *crowdsourcing marketplaces*, a market-based approach in which requesters monetarily compensate contributors (or crowds). Alternatively, *voluntary or game-based mechanisms* provide other motivating factors that incentivize human input. In this book, we focus primarily on paid market-based approaches to crowd work.

Crowdsourcing Marketplaces. There are a number of online crowdsourcing marketplaces. The canonical example of a crowdsourcing marketplace is Amazon’s Mechanical Turk [10] (also referred to as MTurk for short); other examples include Samasource [14], Upwork [17], Clickworker [2], and Crowdflower [6]. There are estimated to be over 30 crowdsourcing marketplaces, and these marketplaces are growing rapidly. In addition, as we will see

in subsequent sections, many large companies leverage crowdsourcing via internal crowdsourcing marketplaces, where the scenario is similar, i.e., workers get monetarily compensated for their work, but the workers are employed in-house or through contractual relationships that companies and workers establish. Note that these are not strictly crowdsourcing marketplaces in the traditional sense since these workers have longer-term relationships with companies and are paid a 9–5 wage to work on tasks.

The structure of marketplaces vary, but below, we describe one representative design that is similar to the design adopted by MTurk. There are two interfaces for accessing a typical crowdsourcing marketplace. The first is seen by *task requesters*, the second is seen by *workers*.

- The first interface is the one used by the task requesters or *task designers*—these are the individuals or teams who have tasks for which they would like to leverage crowd work. Tasks are typically introduced with a task definition or description, and often provide a form consisting of text boxes, drop-down menus, or radio buttons to elicit meaningful information from workers. Task designers design suitable tasks, and they typically specify the monetary reward or compensation associated with these tasks to be paid upon completion. Optionally, they may specify: (a) the *assignment*, i.e., the number of identical copies of the same task to be attempted by different individuals independently, (b) the amount of time allocated for that task before the task “expires,” or (c) additional criteria (e.g., a spoken language) that individuals who want to work on these tasks must satisfy.
- The second interface is the one used by crowd workers, or simply workers, to access the entire set of tasks for which they are eligible, and to complete work on those tasks. Workers can browse the list of available tasks, pick up tasks that they wish to attempt, and work on them. In some cases, the matching or assignment to tasks is done automatically. The same task may be attempted by multiple crowd workers. If so, the workers work on tasks independently, and each one is compensated on completion of the task within the specified time limit.

Voluntary or Gaming-based Crowdsourcing. In addition to paid crowdsourcing marketplaces, there are other mechanisms by which humans are

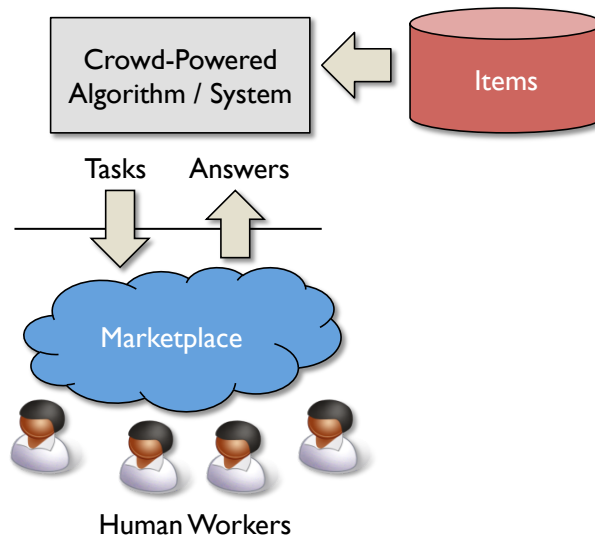


Figure 1.1: Interacting with a Marketplace





incentivized to work on tasks. One such mechanism is to solicit volunteers to work on tasks for a worthy cause. As an example, volunteers were asked to help translate tweets during the Haiti earthquake [206], or help identify galaxies in astronomical images [154, 195]. Yet another mechanism relies on games [185]. In this mechanism, people play games for fun, without realizing that the games are, in fact, tasks that need to be solved.

Even though our focus is on crowdsourcing marketplaces, the crowd-powered algorithms and systems that we talk about can also be used in conjunction with voluntary or gaming mechanisms, since there is still a limited budget of human attention that those mechanisms require that can be treated as analogous to monetary cost in crowdsourcing marketplaces.

1.2.2 Interacting with a Crowdsourcing Marketplace

We now describe how crowd-powered algorithms or systems interact with a marketplace to create tasks for crowd workers. An informal diagram of the interaction is shown in Figure 1.1. The algorithms and systems we describe operate on data items like images, videos, or text, and construct tasks to be asked to workers. These tasks are generally expressed using HTML markup

Do the following images satisfy **no watermark (no text or logo on top of image)** ?

| Thumbnail | Image Details | Rate |
|--|--|--|
|  | Large Image: http://www.clipartpal.com/_thumbs/005... Context: http://www.clipartpal.com/clipart/sch... | <input type="button" value="Yes"/> <input type="button" value="No"/> |
|  | Large Image: http://www.clipartpal.com/_thumbs/034... Context: http://www.clipartpal.com/clipart/sch... | <input type="button" value="Yes"/> <input type="button" value="No"/> |
|  | Large Image: http://www.clipartpal.com/_thumbs/041... Context: http://www.clipartpal.com/clipart/sch... | <input type="button" value="Yes"/> <input type="button" value="No"/> |
|  | Large Image: http://www.illustrationsof.com/royalt... Context: http://www.illustrationsof.com/95541-... | <input type="button" value="Yes"/> <input type="button" value="No"/> |

(Click to get paid)

Figure 1.2: Filtering Task

for descriptions or examples, and HTML forms for input. Tasks are posted on the crowdsourcing marketplace using an API specific to the marketplace, along with worker requirements and payment policies. These tasks are answered by workers independently. Once answers to these tasks are provided back to the crowd-powered algorithm or system, the algorithm or system may choose to issue additional tasks once again, or may instead terminate.

Since workers may be concurrently working on different tasks, we can view the algorithm or system as having workers work on tasks in parallel, waiting for their responses, then having workers work on additional tasks in parallel, and so on. However, note that the system can in fact issue new tasks to the crowdsourcing marketplace before the outstanding ones are complete.

Example Tasks

We show two example tasks, as seen by workers, in Figures 1.2, and 1.3. Once a crowd worker completes either of these tasks, the worker can submit their responses to receive compensation for their effort.

Rate each image on **how funny it is** :

Rate on a scale of 1 to 5, from not funny (1) to very funny (5).

| Thumbnail | Image Details | Rate |
|-----------|--|-----------|
| | Large Image: http://i410.photobucket.com/albums/pp... Context: http://s410.photobucket.com/albums/pp... | 1 2 3 4 5 |
| | Large Image: http://icanhascheezburger.files.wordp... Context: http://eyeonlifemag.com/a-hat-for-all... | 1 2 3 4 5 |
| | Large Image: http://cdn.uproxx.com/wp-content/uplo... Context: http://www.uproxx.com/gammasquad/2012... | 1 2 3 4 5 |

(Click to get paid)

Figure 1.3: Rating Task

The first task consists of a batch of four *filtering questions*. These questions check if specific items (in this case, images) satisfy a given filtering predicate (in this case, whether they do or do not have a watermark). In this task, notice that only the last image does not have a watermark; while it is easy to make out the watermark in the first and third images, the watermark in the second image is much harder to distinguish from the rest of the image, and crowd workers may be more likely to make a mistake on this image compared to the other images. Thus, ensuring that we get correct answers for filtering questions on some items may be more difficult than others.

The second task consists of a batch of four *rating questions*, or questions requesting ratings for specific items (once again, images) for the predicate *how funny it is*. In this task, since humor is subjective, different crowd workers may have different opinions on what constitutes a funny image. Furthermore, some workers may be much more generous than others in providing high ratings. Thus, given various worker answers, inferring the true rating for each image is not trivial.

1.2.3 Terminology

There are several terms we use throughout the book; we collect them here to serve as an easy reference:

- **Crowdsourcing/Human Computation/Crowd Work.** Leveraging human processing power to solve problems that computers cannot yet solve.
- **Marketplace/Platform.** The online forum where requesters can post tasks, and workers can pick up tasks and work on them. We will use both marketplace and platform to refer to both popular forums such as Mechanical Turk (see below) or CrowdFlower, as well as in-house operations where workers work on tasks from 9–5.
- **MTurk/Mechanical Turk.** One of the popular crowdsourcing marketplaces, often used by academics.
- **Marketplace Provider.** Companies like Mechanical Turk and CrowdFlower that provide a marketplace or platform for crowdsourcing.
- **Worker/Contributor/Crowd Worker/Human Worker/Contractor.** The human being completing the task at hand.
- **Requester/Designer/Developer.** The human being or team designing and developing the task for crowd workers to complete.
- **Task definition.** The high-level description and implementation of the task being completed (e.g., *Please identify the gender of the person in each of the following images*).
- **Task/Item/Unit/Question.** A unit of work that a crowd worker must complete (e.g., *Identify the gender of the person in the following image: (image 1)*).
- **Interface.** This is the view presented to the crowd worker when they choose to work on a task. This could involve textual descriptions, as well as forms.
- **Answer/Response.** The response given by a crowd worker for a task.

- **Assignment.** A matching of a worker to a task—this may be done automatically by the marketplace, or on-demand by the workers, or on-demand by the requester. Tasks are often assigned redundantly to multiple workers.
- **Microtask.** The most popular form of task in traditional crowd work environments, in which short, relatively precise and often limited responses are allowed (e.g., multiple choice questions, yes/no questions).
- **Macrotask.** A task that is higher-level and more freeform, and takes longer to elicit a response (e.g., *Research and write up three pages on the British banking system*).
- **Reward/Compensation.** The incentive provided to the workers upon completion of the task.
- **Crowd-Powered Algorithm.** An algorithm where the unit operations are performed by crowd workers as an integral component. For example, sorting images where crowd workers compare pairs of images.
- **Crowd-Powered System.** A system or framework that uses crowd work as an integral component.
- **Latency.** The time taken by a crowd-powered algorithm or system to complete.
- **Error Rate.** The rate at which workers end up answering tasks incorrectly. This is typically a number between 0 and 1.
- **Worker Quality/Worker Accuracy.** One minus the error rate of workers. This is how often workers end up answering tasks correctly.

1.3 Crowdsourcing Best Practices

In as much as there is deep science and research behind effective crowdsourced task design, there are also some practices to follow that should provide good results. Recent work has also cataloged similar best practices specifically for information retrieval tasks [24]. Here are a few practices to follow when designing tasks:

- **Decomposition.** Break larger tasks down into smaller ones. For example, say you wish to find images of cats in a large collection of animal photos. Avoid asking workers to spend an hour searching for an example image of a cat in a stream of photos. Instead, show workers one image at a time, and ask them whether the photo contains a cat.
- **Closed-Ended, Easy to Answer Responses.** Opt for well-defined, closed-ended responses where possible, and pick interactions that make it as hard to answer a question incorrectly as it is to answer correctly. Imagine that you wish to identify the key character in a paragraph excerpted from a book. If you ask workers to fill in the name of the character in a free response text field, it is easier to leave the field empty or with unhelpful text than it is to fill in the correct character. Further, in filling in the correct character, the workers may unwittingly end up making errors. If you instead create a multiple choice interface where the characters of the book are pre-populated, selecting the key character is as simple as providing an incorrect response.
- **Instructions and Examples.** Write detailed instructions, and provide several examples. Most workers appreciate thoughtful step-by-step instructions to complete tasks correctly, and find nuanced examples helpful so that they can acclimate themselves to how you would complete various tasks. Providing a list of “do’s” and “don’ts” is also helpful.
- **Debug.** After you have prototyped a task, have a colleague who is not familiar with your work complete the task. Watch them complete it and have them talk you through their understandings and actions to identify any places for improvement in your interfaces or terminology.
- **Pay Fair.** Fair pay is as critical in crowd work as it is in any other form of work. Once you have settled on a task design and implemented it, find a different colleague that has not seen the task before. Time their completion of several tasks, and from that, determine how many tasks per hour you can expect someone to complete. Keep in mind that your colleagues might have certain subject matter expertise that allow them to complete tasks faster, and be prepared to correct for poor estimates.

Based on the expected tasks completed per hour, price your tasks such that they result in a fair hourly rate. Rates differ by platform and task, but expect to pay a rate that is higher than the American minimum wage.

- **Respond to Feedback.** Either through the platform or through forums that workers use (e.g., TurkerNation [16]), seek out worker feedback and respond to it quickly. Expect to iterate on your task design and implementation as you learn from your collaboration with workers [25].
- **Manage Quality.** Because your instructions might be misleading, and because workers might make mistakes, you should expect multiple workers to answer each question/task. If the responses to the task you have created are closed-ended, send each task assignment to multiple workers and combine redundant responses. Combine their responses with simple techniques like majority voting, or more complex ones that we describe in Section 2.3.2. If instead your task is open-ended (like typing up free-response text), take multiple workers' responses and show them to a different set of workers that can identify the best responses [36]. Once you have determined which workers tend to effectively answer questions, provide them with bonuses for their good work, and offer them future work with you as a reward.

Note that much of this advice applies mostly to microtask-based work, and won't all be relevant as tasks become more complex. At a high level, iteratively testing your designs and establishing trusted relationships with crowd workers [165] will improve your experience and theirs, and this advice applies to any form of crowd work.

1.4 Assumptions in this Book

Crowdsourcing has come to encompass a large corpus of work distribution mechanisms. For the purposes of this book, we focus primarily on paid microtask-based crowd work. While our surveys and interviews touch on other areas of the design space, our primary areas of study for crowd-powered data processing systems assume small, well-defined tasks that many workers have access to on a paid basis through a marketplace provider of crowd work.

2

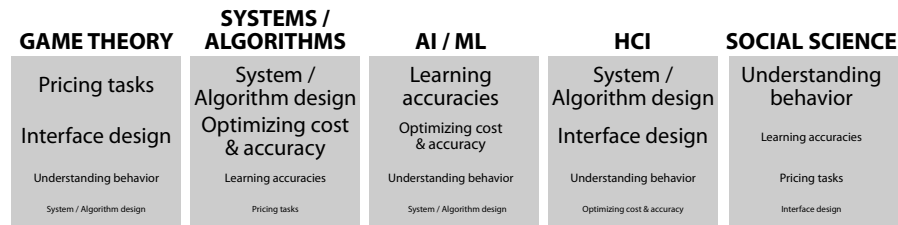
Related Work

We now present a brief overview of how different academic communities approach crowdsourcing research. In Table 2.1, we list these communities along with the section in which we discuss the work done by that community. As communities overlap in research interests, we sometimes arbitrarily assign a line of work to a community, even though it might not be a perfect fit. We omit the discussion on crowd-powered algorithms. We will cover these papers in Chapter 3.

Figure 2.1 presents another depiction of the concerns across communities. For each community, we display in order, the importance of six different interest areas: pricing tasks, interface design, system/algorithm design, learning accuracies, optimizing cost & accuracy, and understanding behavior. For instance, both the systems/algorithms and HCI communities consider system/algorithm development important, but the HCI community also considers interface design important, while the systems/algorithms community also considers optimizing cost and accuracy important.

| Discipline | Perspectives |
|---|--|
| Human Computer Interaction (Section 2.2) | Better interface design; novel interaction mechanisms |
| Machine Learning (Section 2.3) | Crowd-provided training data; improving crowdsourcing |
| Social Science (Section 2.4) | Behavioral experiments; motivations; demographic studies |
| Game Theory (Section 2.5) | Pricing; incentives; game design |
| Systems and Algorithms (Section 2.6, Chapter 3) | Humans as data processors; workflows; optimization |

Table 2.1: Summary of Perspectives across Communities



KEY (MULTIPLE SELECTIONS PERMITTED)

Primary interest area Secondary interest area Tertiary interest area Quaternary interest area

Figure 2.1: Comparison of Concerns Across Fields

2.1 Surveys

There are a number of recent surveys describing various aspects of crowd-sourcing research. The article that coined the term *crowdsourcing* first appeared in Wired [33]. Quinn et al. [153] present a taxonomy of the various terms (e.g., crowdsourcing, social computing, human computation) used to describe the different ways humans may participate in computer algorithms and systems. Perhaps the most comprehensive survey on crowdsourcing is the monograph by Law and Von Ahn [122]. Doan et al. [74] provide another survey on crowdsourcing technologies.

2.2 Human Computer Interaction

The Human Computer Interaction (HCI) community has been focused on developing new platforms for interacting with and understanding crowd workers. We divide the HCI work into two subsections. First, we cover the de-

velopment of novel crowd interfaces and their usage in applications for both end-users and supervisors of crowd-powered tools. Second, we cover the development of games as a mechanism to interact with human workers.

2.2.1 Novel Worker and User Interfaces

We now describe a collection of novel interfaces considered by the HCI community to get useful input from workers.

- **Bounding-box interfaces.** Noronha et al. show with Platemate [139] that in addition to labeling images with which foods they contain, MTurk workers are also effective at estimating the nutritional properties of the food, such as how many calories the food contains.
- **Reposting and workers-on-standby.** Bigham et al. [39, 39, 139] describe a mobile phone application that allows blind users to have various crowds, ranging from Mechanical Turk workers to their Facebook friends, to label objects they take pictures of. In such a scenario, latency is important, and the authors present a system called quikTurkit that, for a fixed price, can drastically reduce the latency of results. Latency reduction is accomplished through several techniques, including reposting tasks with regular frequency and posting more tasks than are immediately available to entice workers. Similarly, Bernstein et al. [35] describe how to keep workers on standby to ensure low latency on tasks like assisted photography.
- **Collaborative Error-Finding, Fixing, and Verification.** In Soy-lent [36], Bernstein et al. add crowd-powered text shortening, proof-reading, and macro functionality to Microsoft Word. The authors present a programming paradigm they name *Find-Fix-Verify*, which is designed to elicit several small bits of creative work from the crowd and then have other crowd workers rate the alternative contributions to separate good from bad.
- **Collaborative Constraint Satisfaction.** In Taskplan [200], Zhang et al. present an interface for collaborative constraint satisfaction for complex tasks such as planning a trip.

- **Tools for Application Developers.** There is also work from the HCI community on improved interfaces for task requesters to supervise and evaluate workers, including visualizing worker behavior [164], supervising their work [77], performing analytics on worker retention and fatigue [95], and relinquishing control of GUIs to remote workers [121].

2.2.2 Games With a Purpose

Work by Von Ahn’s group has explored the design of game-based interfaces to extract useful data from human players without monetary rewards [182, 185, 183, 184, 186]. As an example, the ESP game has players collaboratively guess tags for images while simultaneously providing useful labels for images as a by-product. ReCAPTCHAs have users transcribe or decipher one known word and one unknown word to authenticate themselves as human beings while simultaneously providing transcriptions for digitized books or audio clips. Peekaboom has players identify the portions of an image are the most evocative of a specific tag while simultaneously providing useful image understanding datasets.

Creating enticing games for the purpose of extracting useful data has been applied in other fields as well—FoldIt [57] has humans identify stable 3-D configurations for proteins, while Duolingo [8] has humans translate sentences while learning a new language. In a mix of human and machine computation, Branson et al. integrate humans and learning algorithms into a visual 20 questions game [45]. In this game, a coordinating algorithm uses machine vision to select and put an ordering on the questions asked of humans to assist in classifying pictures of birds.

This research shows that crowd algorithms and optimization techniques can prove helpful in domains where workers are not compensated monetarily for their contributions, but are instead enticed to continue contributing via “gamification.”

2.3 Machine Learning and Artificial Intelligence

The Machine Learning (ML) and Artificial Intelligence (AI) communities have been studying how crowds may be used to get better training data, and how machine learning algorithms may be used to improve crowdsourcing.

2.3.1 Active Learning

The field of Active Learning studies the problem of adaptively selecting training data to be labeled to improve the performance of machine learning algorithms. The survey by Settles et al. [169] provides a good overview of the area. Most papers in Active Learning do not assume that the labels for training data may be human-provided (or at least that they are provided by experts rather than error-prone workers) and therefore may contain mistakes, although a small fraction of papers do take mistakes into account.

Papers focusing on the theory of active learning [56, 64, 111, 31, 92, 38, 37, 28, 63, 198, 136] show that the adaptive selection strategies proposed (i.e., the procedures that select, at any point, a training example to be labeled by a worker) provably converge to the optimal machine learning model, under some assumptions on the selection of training data to be labeled, as well as the noise in the underlying model. Some of these schemes suffer from a severe computational barrier: they explicitly maintain all models that are still under consideration at each point during adaptive selection of training data points. Recent approaches, such as Importance Weighted Active Learning [38] try to eliminate this computational barrier while providing comparable guarantees.

Other papers have suggested many adaptive selection schemes that work well in practice (but provide limited to no theoretical guarantees), including, uncertainty sampling (picking the training data point that the current model is least certain about) [171, 123], query by committee (picking the training data point that a “committee” of current models disagree about) [170], or error reduction (picking the training data point that is most likely to reduce the error).

2.3.2 Quality Estimation

One of the key goals in building crowdsourced workflows is to identify workers that produce high quality output, and tasks that are reliable. We now cover several approaches to achieving this goal in the literature.

Expectation Maximization, or EM [66, 70] has been studied and used by the statistics and machine learning communities for several decades now, with many textbooks and surveys on the topic [90, 135, 187]. Expectation Maximization provides maximum likelihood estimates for hidden model pa-

rameters based on a sequence of E and M steps or iterations that converges to a locally optimal estimates for the hidden model parameters.

There have been a number of recent papers that study the use of EM to simultaneously estimate the answers to tasks and error rates of workers. These papers consider increasingly expressive models for this estimation problem, including worker bias [100], difficulty of tasks and worker expertise [193, 157, 106, 108], adversarial behavior [156], and online evaluation of workers [191, 128, 155]. There has also been some work on selecting which items to get evaluated by which workers in order to reduce overall error rate [171, 75].

There has been some work that adapts techniques different from EM to solve the problem of worker quality estimation, also with no global guarantees. For instance, Chen et al. [54] adopts approximate Markov Decision Processes to perform simultaneous worker quality estimation and budget allocation. Liu et al. [127] uses variational inference for worker quality management for filtering, and Zhou et al. [202, 203] use minimax entropy.

There has been a lot of recent work on providing partial probabilistic guarantees or asymptotic guarantees on accuracies of answers or worker estimates, for various problem settings and assumptions. The papers draw from various techniques, including spectral methods [59, 83], message passing [112], a combination of spectral methods and message passing [113], or a combination of spectral methods and EM [201]. Joglekar et al. [107] consider the orthogonal problem of finding confidence bounds on worker error rates.

Worker quality could also be predicted independent of their performance on particular tasks. For example, Rzeszotarski et al. [163] train a model with worker behavioral information (e.g., scrolling, mouse movements, completion time) to classify suspect responses with 80% accuracy.

In practice, to ascertain worker quality, CrowdFlower [6] requests that users provide gold standard data with which to test worker quality, and disallows workers who perform poorly on the gold standard. For categorical data, often practitioners adopt simple strategies such as asking multiple workers each question and selecting a majority vote of responses can improve results.

Overall, crowd-powered algorithms and systems could certainly benefit from using some of these techniques to better assess the quality of the work provided by workers.

Thus far, techniques for measuring worker quality as a proxy for result quality have assumed a strong worker identity system and no cross-worker coordination. These algorithms are susceptible to Sybil attacks [76] by workers with multiple identities or workers who act in concert to avoid being identified as spammers while finishing tasks faster than high-quality workers. Sybil attacks are well-researched in systems design, but have only been sparsely studied in the field of human computation [130].

2.3.3 Decision Theory

Recent work has leveraged decision theory for improving cost and quality in crowdsourcing workflows. Weld et al. have used POMDPs (Partially Observable Markov Decision Processes) to design optimized workflows [124, 125, 46, 58] In particular, they model worker behavior, task difficulty, and output quality to dynamically choose the best decision to make at any step in the workflow (refine, improve, vote, or stop), and also to dynamically switch between workflows to improve the overall “utility.”

Kamar et al. [109] use POMDPs to study how to best utilize participation in voluntary crowdsourcing systems, specifically, Galaxy Zoo, an astronomical data set verified by workers.

Unlike the previous papers that have no guarantees, Parameswaran et al. [143, 141] identify optimal policies for MDPs for filtering and rating. However, they assume the worker accuracies are provided in advance, unlike the papers given above.

2.4 Social Science

We now evaluate several applications and studies of crowdsourcing in the social sciences.

There is a wealth of work on exploring social and behavioral aspects of crowdsourcing, typically by running experiments on crowdsourcing marketplaces (primarily Mechanical Turk [10]). These include studies on how honest workers are [174], what kinds of tasks workers enjoy [97, 177], whether crowdsourcing marketplaces are a good testbed for user studies [115, 117], how pricing impacts worker behavior [134, 96], and how often spam or bias occurs [100, 137].

One of the first researchers to study crowd platforms such as Mechanical Turk was Ipeirotis, who offers an analysis of the marketplace [98]. From this study, we learn that between January 2009 and April 2010, Mechanical Turk saw millions of HITs, hundreds of thousands of dollars exchanged, and that the most popular tasks performed were product reviews, data collection and categorization, and transcription. HIT prices range from \$0.01 and \$10.00, with approximately 90% of HITs paying less than \$0.10. Ipeirotis also surveyed Turkers to collect demographic information. Through these surveys, we learn that between 2008 and 2010, the Mechanical Turk population demographics resembled the US internet populations', with a bias toward more females (70%), younger workers, lower incomes, and smaller families [99].

As crowdsourcing platforms open up to a more global population, however, we see several demographic shifts. With an increasing crowd worker population from the developing world (in particular, India), Ross et al. show that with changes in demographics come changes in motivations for performing work [161]. Workers from India are more likely to see crowd work as a primary source of income, whereas US workers treat it as an income supplement or source of entertainment. This study showed that workers for whom crowd work is their primary source of income, pay becomes a larger incentive for performing tasks.

Mason and Watts [134] studied the effects of price on quantity and quality of work. They find that workers are willing to complete more tasks when paid more per task. They also find that for a given task difficulty, result accuracy is not improved by increasing worker wages.

2.5 Game Theory and Pricing

The algorithmic game theory community has been addressing economic issues in crowdsourcing: for instance, ensuring that the marketplace is “efficient,” that there is “fairness” in worker compensation, and that workers are incentivized to put in their best effort, or “truthfulness.” In particular, the community has studied incentive structures in crowdsourcing marketplaces [96, 52]; they have also studied how to improve the efficiency of crowdsourcing [110], games with a purpose [103, 102], crowdsourcing contests [50], Question-Answer (QA) forums [101], and user-generated con-

tent [84]. A recent survey [82] summarizes the recent developments in this field. While our focus has been on minimizing the number of questions asked to human workers in designing crowd-powered algorithms and systems, such efforts may leverage research results from this community to ensure that workers are paid a fair price for their work, and are incentivized to answer truthfully.

2.6 Systems and Programming Models

There are three types of crowd-powered systems (note that we also covered some crowd-powered systems with novel interfaces within Section 2.2): full-fledged database systems that support crowdsourcing, specialized crowd-powered toolkits targeted at specific application domains, and generic programming toolkits that allow requesters or application developers to use crowdsourcing within programs. We will cover full-fledged database systems in Chapter 4; we cover the rest here. A tutorial on crowd-powered systems can be found in [73].

2.6.1 Basic Programming APIs

Crowdsourcing platforms such as Mechanical Turk offer low-level interfaces for posting HTML forms or iframes around HIT content that is hosted elsewhere. The APIs allow requesters to generate new HITs, specify a price per HIT assignment, and set a number of assignments per HIT. Requesters manage their own multi-HIT workflows, poll the API for task completion, and gauge the quality of the responses on their own.

Mechanical Turk-style APIs are akin to filesystem and network APIs on which databases and other data and information management platforms are built. Building robust crowdsourced workflows that produce reliable, error-free answers on top of these APIs is not easy. One has to consider how to design the user interface (an HTML form) the crowd worker sees, the price to pay for each task, how to weed out incorrect answers, and how to deal with latency on the order of minutes to hours of various HITs that crowd-powered programs generate. Several startups, such as CrowdFlower [6] and MobileWorks [21] aim to make crowdsourced workflow development easier by offering simplified APIs (CrowdFlower) or task-specific ones (MobileWorks).

2.6.2 Generic Toolkits

Further up the stack are crowdsourced language primitives, supported typically by generic toolkits that allow application developers to leverage crowdsourcing within code. However, none of these toolkits provide the functionality of optimization. In effect, it is up to the programmer or application designer to manually optimize the workflow while using these programming toolkits.

Little et al. present TurKit [126], which supports a process in which a single task, such as sorting or editing text, might be implemented as multiple coordinated HITs, and offers a persistence layer that makes it simple to iteratively develop such tasks without incurring excessive HIT costs. Much like low-level parallelization frameworks such as *pthread*s [47] allow developers to fork multiple tasks off to workers in parallel, TurKit offers several parallelization primitives. Like low-level threading primitives, however, low-level crowd programming libraries require care in correctly using *fork/join*-style parallelization primitives.

At a higher level of abstraction, toolkits such as CrowdForge [116] and Jabberwocky [22], and Automan [32] help specify crowdsourced workflows. CrowdForge by Kittur et al. provides a MapReduce-style programming model for task decomposition and verification. Jabberwocky[22] from Ahmad et al. provides a full stack: Dormouse for low-level crowdsourcing primitives, ManReduce for a MapReduce-style task decomposition, and Dog as a Pig-like programming environment [19] for specifying workflows at a high level. Finally, systems like Legion [121] make it easier for developers to build applications that integrate crowd worker feedback with low latency. Automan [32] enables application developers to leverage crowdsourcing via subroutines in regular programs.

As the crowdsourced work becomes more complex, researchers have proposed more sophisticated task workflows and worker organization schemes. The literature thus far on complex work has focused more on building novel workflows than on evaluating their quality. This research proposes multi-stage workflows to break work into manageable subtasks [119], maintain global constraints across multiple tasks [200], iteratively refine previous workers' efforts [126], or assemble teams of expert workers to collaborate on difficult or creative tasks [159]. Worker hierarchies have been used to

organize crowdsourced managers and employees in the context of micro-tasks [139] and more complex work [118].

2.6.3 Domain-Specific Toolkits

There are other domain-specific systems that gather data from crowds:

- Reference [51] leverages crowdsourcing for feedback in information integration pipelines. Other work has addressed algorithmic questions underlying which questions to ask in the information extraction setting [61, 142]. pipelines.
- CrowdSearcher [41, 42, 43] provides a declarative platform to leverage the user's social network as well as QA forums to solve user tasks. In recent work, CrowdSearcher has been enhanced with active rules that enable better user-driven control of crowds.
- Trivia Masster provides a declarative approach to leverage humans for data cleaning [72].
- DataSift explores workflows for crowd-powered search [148, 147].

3

An Overview of Crowd-Powered Algorithms

In this chapter, we cover the state of the art in algorithms that take human worker input into account. A common abstraction adopted by the work on crowd-powered algorithms is that crowd workers act as *data processors*, much like traditional processors within computers.

We will first introduce the notion of crowds as data processors and provide an executive summary of the rest of the chapter. We will then survey the literature on crowd-powered algorithms, and close with a summary of design choices and confounding factors in creating crowd-powered algorithms. We note that since there is a large diversity in the types of algorithms that researchers have designed, for any given problem, we will not delve into specific algorithmic details, and instead will describe the high level ideas. We encourage readers to look up the corresponding papers to find out more.

In this chapter, we do not describe the target use cases for these algorithms or how they are integrated into other computation; we will cover these aspects when conducting surveys of users of crowdsourcing in industry, in Sections 7.1 and 8.3 respectively. In brief, crowdsourcing could be used as a precursor to machine learning (i.e., to generate training data), integrated with machine learning (i.e., via active learning), or after machine learning, to verify the output of machine learning or automated algorithms. Alternatively,

crowdsourcing could be used in a stand-alone manner, not integrated with machine learning in any way.

3.1 Crowd Workers As Data Processors

Traditional processors support both arithmetic operators (e.g., adding, subtracting, multiplying, or dividing large numbers) and logical/boolean operators (e.g., less than, greater than, equal to, or AND, OR, and NOT). Each of these is defined as a unit operation within the processor’s instruction set. It is thus very clear what a processor can and cannot do, and is explicitly specified in the instruction set of the processor—this is a conscious design decision that every processor manufacturer must make.

Given that crowd workers are complex, creative beings, we are not yet sure what crowd workers are capable or not capable of answering within a unit time. Thus, the human “instruction set” is largely unknown. Crowd workers are certainly able to perform arithmetic and logical/boolean operations, especially on small numbers. Crowd workers have a harder time than computers, however, performing operations on larger numbers.

On the other hand, crowd workers may be able to quickly perform operations on entities other than numbers (something traditional processors are incapable of doing). For instance, crowd workers can operate on images, videos, or text; they can quickly compare two images, for example. Here is an incomplete list of operations that crowd workers can complete on images alone:

- compare two images on some aspect, e.g., pick the prettier picture,
- rate an image on some aspect, e.g., rate how funny a meme is,
- evaluate a predicate on an image, e.g., is this an image of a cat,
- categorize an image into categories, e.g., is this an image of an animal, a vehicle, or a fruit,
- count items in an image, or count the number of images obeying some property, e.g., count the number of images that have a cat in them, and
- sort a number of images on some aspect, e.g., sort 5 images based on the quality of each image.

The “performance” of a crowd worker on an operation set of items in exchange for some reward can include several factors. The time a crowd worker takes and the correctness of the answers they provide depend on multiple factors, including

- the specific crowd worker under consideration: some crowd workers may be more effective, more accurate, or faster than others at a task, or may have some inherent biases;
- the items under consideration: operations on some items may be harder than others;
- the reward under consideration: higher rewards may induce quicker or more thorough responses; and
- the operation under consideration: different operations may be harder or simpler than others.

For the purposes of crowd-powered algorithms, we focus on operations that can be done “quickly,” i.e., within a few seconds or minutes. These operations are what we call *microtasks*, and this class of operations does not include longer and more elaborate tasks, such as writing an essay—those tasks are called *macrotasks*. In this chapter, we mostly consider microtasks because macrotasks are less amenable to optimization, and less applicable in a large-scale data processing scenario.

Even though we present a number of aspects that the literature on crowd-powered algorithms do take into account, modeling crowd workers as data processors is certainly a gross simplification: crowd workers are incredibly complex, and simple models or abstractions are likely to not be accurate depictions of human behavior. Nevertheless, even with these simple models we will find that (a) the resulting problems are still challenging to reason about, and (b) even the simple models often lead to substantial benefits in practice. It remains to be seen if more intricate abstractions or models of human involvement will provide additional benefits.

3.2 Executive Summary

Current Progress. Given the abstraction of humans as data processors, there have been a number of papers written on using relatively simple data processing operations, primarily centered on pairwise comparisons and predicate evaluations. The predicates are not always boolean, but also more complex ones like rating (e.g., rate an item from 1—5). This work includes progress on sorting, max, and top-k [65, 181, 89], where the focus is primarily on pairwise comparisons, but also expands to multi-way comparisons and predicate evaluations [132]. Further work on filtering and finding [143, 167] and categorization [146] focuses on predicate evaluations. A good body of work has developed around entity resolution and clustering [189, 86], where the focus is also on pairwise comparisons. Some papers opt for more complex human data processing abstractions, such as work on counting [130], enumeration [179], or association rule discovery [27].

Design Choices and Objectives. Given a crowd-powered algorithm design problem, an algorithm designer needs to take into account a variety of factors, such as:

- The tasks that will be completed by workers, e.g., predicate evaluations, comparisons.
- The amount that workers will be compensated for completing each task, also called the reward in the literature.
- The mechanism used to reason about errors made by workers. One option is to ignore errors. Another option is to assume that all workers are equally “good,” and to take a majority vote. Other more sophisticated mechanisms are possible as well.
- The mechanism used to reason about the time taken by workers to complete tasks. Typically, workers are assumed to take the same amount of time per task, which may not be realistic.

Given these choices, the algorithm designer can then optimize for various objectives, including optimizing for expected or worst case monetary cost, accuracy of answers, and latency of completion.

Other Confounding Factors. In designing algorithms, there are a range of factors typically ignored by algorithm designers, but are also important; we expect these factors will be explored more in future algorithm development in this area. This includes

- Task-specific factors, including vastly varying difficulties of tasks, the impact of batching tasks together on the quality of answers, and interface design and testing, which is largely seen to be a dark art.
- Machine-learning factors, including the incorporation of information from primitive machine learning algorithms, as well as the integration with active learning techniques.
- Human-specific factors, including the effects of fatigue, boredom, and/or experience of workers as they attempt more tasks, and varying the reward on the quality of answers.

3.3 A Survey of Crowd-Powered Algorithms

We now describe a survey of the literature in crowd-powered algorithms. We organize this literature in subsections corresponding to different fundamental algorithms that need to be revisited when the unit operations are performed by crowd workers instead of computers. Necessarily, since crowd-powered algorithms is a rapidly evolving field, we may have missed out on some of the newer references. This section is not meant to be exhaustive, but more illustrative of the breadth of the work in crowd-powered algorithms.

3.3.1 Sorting, Max, and Top-K

Here, the goal is to either sort a number of items, find the maximum (or best) item, or find the best k items. This could be relevant, for instance, if we wanted to sort a set of tweets on how positive they are toward a certain brand, or if we wanted to find the best profile photograph to be uploaded on a restaurant website.

At a high level, most of these algorithms use some form of pairwise comparison questions, whose results can be visualized as a directed graph between items. The algorithms generally focus human attention on areas of high “uncertainty.”

Guo et al. [89] describe online algorithms to select the best next pairwise comparison question to ask the crowd in order to identify the max. They formulate this as a maximum-likelihood problem, and show that even just computing the maximum given a number of pairwise comparison votes is #P-Hard.

Marcus et al. [132] study crowd-powered sorting: they empirically find that a hybrid algorithm that uses ratings to get a rough idea of how “good” items are, and then pairwise comparisons between items in the same rating class, needs much less cost than an algorithm that uses ratings alone or pairwise comparisons alone.

Davidson et al. [65] design a pairwise-comparison-based structured tournament for crowd-powered top-k and maximum problems, and demonstrate that their algorithms are optimal under certain assumptions about worker error rates. Venetis et al. [181, 180] also study tournament-based max algorithms, and identify “good choices” for the fanout of the tournament tree under various error models. Polychronopolous et al. [152] devise algorithms for top-k by merging lists in a manner similar to the quick-sort algorithm.

3.3.2 Filtering, Rating and Finding

Here the goal is to either identify which out of a set of items satisfy a certain predicate or condition (filtering), or assign every item in a set of items a score (rating), or identify k items from a set of items that satisfy a certain predicate or condition. For instance, filtering could be relevant, say in content moderation, when we want to identify all photos that contain inappropriate content from a large dataset of photos; rating could be relevant when associating every web search result a score representing how appropriate it is for a given search query; finding could be relevant when we want to use crowd workers to identify 20 travel photos from a dataset of 10,000 photos to display on a travel website.

Typically, the algorithm to beat for filtering or rating is the “majority vote”: asking a fixed number of workers to provide a score or value for each item, and then accepting the majority response as the true score or value for each item. Most of the algorithms designed for these problems reduce cost and increase accuracy relative to majority vote by dynamically reducing the number of workers assigned to items where there is clear agreement between

workers (so additional opinions are not necessary), and increasing the number of workers assigned to items where there is disagreement between workers.

Parameswaran et al. [143, 141] propose the use of Markov Decision Processes to solve the problem of filtering and rating by optimally trading off cost and quality. They show that often simple techniques such as majority vote can perform very poorly especially when applied to large numbers of items. Liu et al. [128] use binary search to identify the right number of workers to ensure a certain degree of accuracy, while Cao et al [48] carefully select a set of workers that optimally balance cost and quality.

Das Sarma et al. [167] demonstrate how, for finding, it is not simply sufficient to use techniques optimized for filtering (in fact these can be arbitrarily expensive). Here, the emphasis is on focusing on a small set of items that are most likely to be useful in satisfying the condition. They propose algorithms that best trade-off monetary cost and latency, given a threshold on accuracy.

The work on filtering and rating in this section is related to the work on worker quality estimation and management as described in Section 2.3.2. There, the primary emphasis is on estimation of worker quality; here, the primary emphasis is on how to best exploit worker quality estimates to reduce cost and provide guarantees on accuracy, *assuming that* worker quality is given as a black box. That said, there are some papers that jointly try to estimate worker quality and reduce cost; these papers are described in Section 2.3.2.

3.3.3 Entity Resolution and Clustering

There has been some work on using crowdsourcing to assist Entity Resolution [78], i.e., the task of identifying duplicate entities in a set of entities. This work has focused primarily on combining machine learning methods with crowdsourcing; crowdsourcing is typically used for verifying predicted matches [189, 68, 85]; in some cases, it is also used to provide training data for matching algorithms [138, 192, 85]. Active sampling [34] addresses a similar problem, but makes use of a classifier that is learned using human input via active learning.

Crowd-clustering [86] tackles the question of what tasks crowd workers should be used for when clustering a large set of items. Data Tamer [173] uses a crowd of domain experts to resolve uncertain examples in schema

integration, data cleaning and entity resolution. Tamuz et al. [175] address the problem of identifying perceptual similarity (by asking questions such as “is a more similar to b or c”) using the crowd, while Heikinheimo et al. [94] identify the “median” or the “centroid” item in a set of items.

At a high level, most of these algorithms represent both prior beliefs about similarities between pairs of items, as well as crowd responses as weighted edges in an undirected graph between items, with the goal of verifying uncertain edges, or splitting the graph into clusters of high density.

3.3.4 Categorization

Here, the goal is to design algorithms for categorizing items (e.g., images or products) into a taxonomy (e.g., concept taxonomy or product hierarchy, respectively).

Parameswaran et al. [146] focus on questions of the form “does this item belong to a specific category?” Asking questions at category nodes close to the root are more likely to receive a positive answer, while asking categorization questions close to the leaves are more likely to receive a negative answer. Asking categorization questions in the “middle” nodes may give more information. They articulate three key dimensions: the type of taxonomy, the objective (eliminate as many nodes or precisely find the target category), and number of target categories (one or many), and provide efficient solutions. Chilton et al. [55], and Bragg et al. [44] describe techniques to crowdsource the creation of taxonomies.

3.3.5 Counting, Estimation, and Enumeration

Count-based aggregates are fundamental to database operation. In their simplest form, they help users summarize row aggregates (e.g., “How many employees do I have?”). In their most complex, they provide database optimizers with selectivity estimation on how many rows have a given property. Marcus et al. [130] study how to benefit from the crowd when the items being counted are challenging for computers to interpret (e.g., “How many photos contain smiling men with red hair?”), and how to benefit from the concept of pop-out [190, 197] from the visual perception literature to batch process such questions with crowd workers.

The goal of enumeration is to recover a set of related items using the crowd (e.g., ice cream flavors, countries, restaurants). Trushkowsky et al. [179] use statistical techniques (inspired by the coupon collector problem) to estimate exactly how many times crowd workers must be asked to provide items before the entire set is collected. Recent work aims to improve on these techniques by leveraging a hierarchy on attributes [158].

Crowd-Fill [151] addresses a similar problem in a relational setting, where multiple concept types can be crowdsourced. A recent paper demonstrates how decomposition can be used to count objects in images, leading to an optimal competitive under certain assumptions [166].

3.3.6 Other Algorithms

Amsterdamer et al. [27] use crowd workers to verify data mining “association rules,” Lotosh et al. [129] use crowd workers to generate optimized plans for workflows, and Demartini et al. [69] use crowds to verify entities and relationships to enable better pattern matching for search queries.

3.4 Design Choices for Crowd-Powered Algorithms

Now that we have the abstraction of crowd workers as data processors, and have a motivating set of crowd-powered algorithms, we can describe typical design choices and variations in crowd-powered algorithms. To accomplish this, we contrast the design choices of two papers on filtering [143] and sorting [132], referring to other papers where it is helpful.

3.4.1 Unit Operations: Controlled by Algorithm Designer

To setup the problem, the requester considers the types of operations and feedback that crowd workers can provide. For instance, for filtering [143], the only operation that can be performed by crowd workers is a predicate evaluation, essentially answering YES/NO depending on whether an item satisfies a filter or not.

The space of operations that can be answered by the crowd to solve a given problem is generally small. Whereas filtering considers only a single predicate evaluation operation, for sorting [132], two different operations are

suggested. In the first, a crowd worker provides a rating for each item, allowing us to compare items by comparing their aggregate ratings (e.g., average 1-5-star funniness rating). The second operation outright asks crowd workers to compare two or more items, allowing a more direct form of sortability.

3.4.2 Cost Model: Controlled by Algorithm Designer

The next aspect a crowd-powered algorithm designer must consider is the reward for operations performed by the crowd. The simplest design choice for the cost model is that the reward is fixed for each task. This is the setup considered in both the filtering and sorting papers, and is often the standard practice for industry users. There are three ways the cost model may vary:

- The costs for two different operation types are different. For example, a comparison operation might cost more than a rating operation, since the former may require understanding of two items, while the latter only requires the understanding of one item.
- When the operations performed by crowd workers are on multiple items at a time, the cost may vary depending on the number of items. For example, comparing 10 items at a time might be twice as expensive as comparing 5 items at a time. This cost model is employed, for example, in a paper on identifying the largest elements in a dataset [181].
- The cost could depend on the human worker who is answering the task; a more competent worker may require higher compensation than a less skilled worker. This cost model is considered for filtering in [141]. In MTurk, there are different categories of workers (with different qualifications) who are typically paid differently.

3.4.3 Error Model: Hidden Parameters

The next aspect affecting the design of crowd algorithms is the model of error, through which we capture the likelihood that crowd workers may give incorrect answers. Unfortunately, this is not something the algorithm designer has control over (unlike cost and latency). If the algorithm designer does not wish to worry about how to reason about the errors made by workers, then they simply use an aggregation method, such as getting the answers of 3 or

5 different human workers, and then taking the majority. Once the majority is taken, the answer is assumed to be correct. However, this ignores crucial information about the answer inherent in the responses. If all 5 workers agree on a boolean value being YES instead of NO, then this is a much stronger statement than if 3 workers thought the boolean value was YES and 2 thought it was a NO. In the first case, we have a lot more confidence in the answer. Furthermore, we may obtain conflicting information from the workers: it may be the case that majority of the workers think A is better than B, majority of the workers think B is better than C, and majority of the workers think C is better than A. Now, there is no easy way to resolve this conflict. Thus, the research literature has considered various ways of reasoning about the worker errors, and figuring out ways of correcting them.

Typically, most papers in the research literature use an appropriate error model that is amenable to analysis, train the model on available data by identifying the best fit for the hidden parameters in the error model, and then design algorithms while being cognizant of the error model. Various algorithms and error models can be tested to see if they lead to tangible benefits in practice: a model that does not yield benefits in practice is useless. Picking an error model is an art—similar to picking a class of machine learning model prior to identifying appropriate parameters on training data. Prior work has analyzed and designed crowd algorithms under various error models, described below:

- The weakest assumption made by algorithms is to make no assumption about crowd worker error rates. This is the approach taken in the sorting paper. The advantage of this error model is that analysis is easy; the disadvantage is that not much can be said about the eventual accuracy of the algorithms.
- The simplest error model is that every human worker has the same probability of error on every operation on every item. This is the error model adopted in the filtering paper.
- A slightly more complex error model states that every crowd worker has a distinct probability of error for each operation across different items; this is the error model adopted by [141].

Virtually no papers consider more intricate error models, but they certainly exist:

- The error rates of crowd workers vary as they answer more questions. This certainly happens in practice, since crowd workers may end up being more fatigued or distracted over time, or get more experienced in answering questions.
- The error rates of crowd workers depend on the item being operated on. While [141] does consider this to a limited extent, this area is largely unexplored.

While work on crowd algorithms focuses on simple, analyzable error models, the work on worker quality estimation (considered in Section 2.3) considers a variety of more intricate models (e.g., different workers have different accuracies; workers have different accuracies on each task; tasks have different difficulties, etc.): however, none of these models come with guarantees; i.e., the EM-like [66, 100] algorithms that these papers use lead to locally optimal solutions. The papers with guarantees, such as [112], use the assumption of fixed worker error rates independent of item. For the most part, there has been no work on incorporating sophisticated error models in crowd algorithms.

We emphasize that worker quality estimation is an essential component of crowdsourcing, but the perspective we adopt in this book is that worker quality estimation is a “black box” that we apply to our algorithms; the emphasis in this chapter, is, for example, on how best to use the estimates from this black box. In the future we expect more papers to do worker quality estimation and data processing (via crowd algorithms) simultaneously.

3.4.4 Latency Model: Hidden Parameters

Like the error model, the latency model is a hidden aspect that affects the design of crowd-powered algorithms. Latency refers to the time taken for crowd workers to complete tasks. Latency depends on factors such as the reward promised to crowd workers, as well as the time of day, as both affect the distribution of workers willing to work on tasks. It also depends on the operation performed, and the items being operated on. Prior work has considered various models for latency:

- Many papers do not consider latency as an optimization objective at all, recognizing that the primary concern is minimizing cost (representing real money investments) and maximizing accuracy (representing the quality of the results that are consumed by the next step of the data processing pipeline); as long as the tasks are eventually completed, that is sufficient.
- Many papers model latency for every operation performed by crowd workers to be the same and fixed, and sometimes assume that as many operations as desired can be performed by crowd workers all in parallel. Thus, in this case, the latency of an algorithm is directly proportional to the number of round-trips to the crowdsourcing marketplace—with the computation cost being completely ignored. This model makes sense because the the latency of crowd worker responses is typically much higher than automated computation. This model is adapted by papers on finding [167] and categorization [146].
- A couple of papers model the latency as being inversely correlated with the reward on offer [81, 93].

Generally, the literature to this point has focused on modeling error more highly than modeling latency. This in part because latency is a moving target even on a single marketplace, and it is challenging to get reproducible and explainable results.

3.5 Optimization Objectives

Now that we have described the various aspects affecting the design of crowd algorithms, we now describe typical objectives that research in crowd algorithms typically optimizes for. Typically, the objectives optimize for cost and at least one of latency, accuracy, or both. The objectives vary in form:

- *Complete vs. Incomplete*: One type of goal involves a fixed set of tasks that need to be completed, and we want to minimize cost, error, or latency. There are many examples of this type: The papers on filtering [143] and sorts [132]: minimize expected cost to finish filtering or finish sorting, given an expected accuracy threshold. On the other hand,

Gao et al. [81] and the paper on finding [167] minimize expected cost to finish a set of tasks or find k desired items, given a fixed deadline.

Yet another type of goal involves a fixed set of tasks, but does not specify if all of these tasks need to be completed, just that “as much as possible is completed”: For example, enumeration [179], max-finding [89], categorization [146], and crowd-fill [151] aim to gather as many answers as possible, improve estimates on the best item, minimize categorization candidates, and enumerate as many unknown data items respectively, within a specified budget.

- *Worst-case vs. Expected Case*: Another common decision point is whether the optimization goals are of the worst-case or of the expected-case variety. If there is an intricate accuracy model (with workers making mistakes with some probability), then typically, at least some or all of the objectives are of the expected case variety. For example, there may be an expected bound on accuracy, or an expected bound on cost.

3.6 Other Confounding Factors

There are many other factors at play that typical crowd algorithms papers do not take into account. This is not because these factors are not important, but because they are hard to analyze and provide guarantees for. We expect that taking these factors into account in designing crowd-powered algorithms would provide even more effective algorithms.

3.6.1 Task-Specific Factors

Difficulty. In practice, not all questions that are asked of crowd workers are equally easy. For example, comparing two items that are very similar is a lot harder than comparing two items that are far from one another. For sorting, there has been a limited exploration of difficulty, e.g., [23, 65], precisely capturing the idea described above. There have also been some empirical studies demonstrating how difficulty can affect error rates. In one empirical study [180, 166], 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 is typically not known up-front; we can only infer that a task is difficult if workers end up disagreeing on its answer—we make the assumption that given the inherent difficulty, workers provide independent answers. If workers make correlated errors, then this is a lot harder to detect and automatically correct for.

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 the 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. There has been limited work on mitigating bias introduced by batching [204].

Interface Design and Testing. The design choices that go into a particular interface a worker sees can significantly improve worker satisfaction and efficiency and minimize worker errors. Making the appropriate design choices is challenging, requiring that designers take into account factors such as pop-out [190] and anchoring [115]. One paper that shows how perception-related factors like pop-out can affect result quality is on counting items in a set with a given property [179]. In it the authors find that crowd workers can better estimate high-pop-out factors such as image color more effectively than they can estimate low-pop-out factors such as the categorization of a block of text.

3.6.2 Machine Learning-Specific Factors

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. 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. Very few papers take prior information into account. One such paper [141] treats the the prior probability as just another worker with a known probability of error. Other ways of reasoning about and incorporating prior information may be more powerful and may need to more benefits.

Integration with Active Learning. The field of Active Learning [169] concerns itself with the identification of training examples to be labeled by crowd workers to identify a “good” machine learning model. Typical goals for crowd-powered algorithms are to classify, categorize, or sort the tasks at hand. However, there has been no 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). The work in these two communities has largely proceeded independently of each other. There has been initial work in this space [138], but much more remains to be done.

3.6.3 Human-Specific Factors

Boredom, Laziness, and Experience. Most crowd-powered algorithms do not take into account many human-specific factors. For example, human workers will often avoid doing work if they can; they will “satisfice” [168] 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: state of the art crowd-powered 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 [179]: collusion is very hard to detect automatically and correct for.

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, many papers in crowd-powered 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 [82] 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.

3.6.4 Marketplace-Specific Factors

State of the Marketplace. A major unaccounted for factor in crowd-powered algorithms is the state of crowdsourcing 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.

Specific Marketplaces. The other issue with current research on crowd-powered algorithms is that it is typically tailored to and validated on a Mechanical Turk-like marketplace. There are many marketplaces in existence, and further, there are many internal marketplaces (as we will see in Section 5.1). These algorithms do not directly apply to these marketplaces, since they obey very different characteristics. For example, on Upwork [17] it is common to pay workers per hour instead of per task.

3.7 Summary

To summarize, while there has been a lot of interesting and impactful work done on crowd-powered algorithms, much remains to be done. In particular, reasoning about the factors described in Section 3.6 will lead to significant savings in latency and cost, and contribute to much more accurate results.

4

An Overview of Crowd-Powered Systems

Embedding crowdsourcing in large-scale applications or workflows is often very time consuming and challenging for programmers or application developers. The onus falls on these individuals or teams to manage which tasks are created and evaluated by humans, how they are priced and how the interfaces look, how to deal with discrepancies in the human answers, and how these answers are combined with other information or other parts of the workflow. While the algorithms described in the last chapter do address portions of this problem, they still do not solve the problem holistically.

Thus, recent work has identified the need for an end-to-end “declarative” approach: the programmer specifies which tasks need to be evaluated by humans, and the system transparently optimizes and manages the details of the evaluation, often while using the algorithms described previously as building blocks. This situation is analogous to data management, where while it is certainly possible to hand-optimize the execution of a data processing task, it turns out to be easier and more efficient to allow users to specify queries at a high level, and empower the database (in this case) to do the optimization.

The overall optimization objectives for declarative crowdsourcing workflows are similar to those of the individual algorithms they include: typically one looks to optimize cost, accuracy, quality, and completeness of the task.

Declarative crowdsourcing systems come in many flavors, mirroring systems in data management:

- **Imperative Toolkits.** Analogous to imperative programming APIs, there are some toolkits that essentially act as a wrapper over an existing crowdsourcing marketplace, making it easier to access crowdsourcing marketplaces, but ultimately leaving optimization, the evaluation of tasks, and redundancy in the hands of the application programmer. TurKit [126], perhaps the first toolkit of its kind, is an example of an imperative toolkit that was designed as a wrapper over Mechanical Turk. TurKit primarily focused on a workflow called *iterative improvement*, considering simple one-item-at-a-time tasks like text editing or design, and did not do much in the way of optimization. In other words, the designer still had to hand-optimize systems like TurKit. Automan [32] is a more powerful version of TurKit, wherein application developers can leverage crowdsourcing via subroutines in regular programs. Once again, the control flow logic is something the application designer has to decide on.
- **Mixed Imperative-Declarative Systems or Toolkits.** Analogous to recent data processing systems such as MapReduce [67] and Pig [140], there are some systems that enable programmers to mix imperative and declarative specification. While the control flow is still imperative, the individual building blocks are declarative. In such cases, the system is given the power to optimize the individual building blocks of the workflow. Jabberwocky [22] and CrowdForge [116] both exemplify this type of system by enabling application developers to write parallel data processing work-flows using humans.
- **Declarative Systems.** Analogous to traditional relational database management systems, there are three declarative systems that allow application developers to leverage crowd work by declaratively specifying goals at a high level, while the systems perform the underlying optimizations. The three primary systems are: CrowdDB [79, 80], Qurk [131, 133, 132, 130], and Deco [145, 144, 149, 150].

In this chapter, we will focus on the third category of declarative systems. At a high level, all of these systems treat human workers as both a data source and

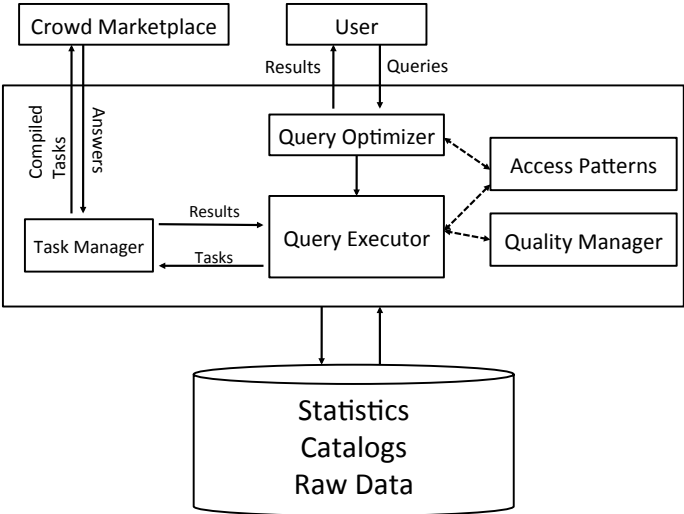


Figure 4.1: Typical Crowdsourcing System Architecture

a data processor—both bringing in new data into the system, and processing the existing data.

There are also a number of domain-specific declarative/imperative systems for crowdsourcing, which are covered in Chapter 2.

We begin with an executive summary of the chapter, then provide details about the individual declarative crowdsourcing systems surveyed.

4.1 Executive Summary

We now provide an executive summary of this chapter.

All declarative crowdsourcing systems have an architecture that looks like Figure 4.1. All systems allow users to pose queries and get results (top of the figure). The query optimizer consults the list of access methods available to gather data from the crowd to cost different plans on monetary cost, latency, and accuracy. The query executor interfaces with the task manager to issue tasks and get results. The task manager in turn compiles tasks into a format

that is issued to the marketplace, and returns answers, whenever available, to the query executor. Lastly, the query executor interfaces with the quality manager to ascertain the accuracy of the returned results, and issues additional tasks if need be. The quality manager might maintain statistics about workers' contribution history in order to identify work quality trends. All components of the system have access to a backend storage system containing data (both crowdsourced and directly input into the system), schema information, and statistics about tables.

The three primary declarative crowdsourcing systems are CrowdDB, Deco, and Qurk. While all three systems use an architecture similar to Figure 4.1, they differ in various ways. We have summarized the various design choices made by these systems in Table 4.1.

In particular, the three systems differ in the kind of data crowdsourced, the way data is represented and combined with existing data (Data Model & Integration Mechanism), the way human errors are captured and reasoned about (Error Model), the way we gather data from human workers (Access Patterns), the amount the database needs to treat crowdsourced data and non-crowdsourced data differently (Database Awareness), the simplicity of understanding of end users (Simplicity), the extent to which optimization and objectives are supported (Optimization Ability & Objectives), and the extent to which users are allowed to specify how much data needs to be crowdsourced (Query Specification).

At a high level, CrowdDB opts for a simple, easy-to-understand design. This simplicity, however, could lead to performance penalties (high cost, time, or error) for some types of queries. Deco opts for a general, more sophisticated design, leading to more optimization opportunities for some types of queries. Qurk, unlike CrowdDB and Deco, opts to capture crowd-specific functionality outside the database via UDF invocations, making it less necessary to modify the internals of the database, but at the same time, sometimes leading to limited query optimization.

While the three systems have made significant headway toward facilitating declarative crowdsourcing, there are still some unresolved issues, warranting further investigation in future work. In particular, none of the systems support full query optimization. This includes, but is not limited to: (a) choosing amongst a collection of query plans for executing the same query by being

| Comparison | System | | |
|-----------------------|--|---|--|
| | CrowdDB | Deco | Qurk |
| Integration Mechanism | Data Source | Data Source | UDF |
| What is Crowdsourced | New data and operations on existing data | New data and operations on existing data | Primarily operations on existing data |
| Data Model | Cleansed data stored: Database view is the user view | Dirty data stored: Database view is not the user view | Dirty or cleansed data stored: Database view is the user view |
| Error Model | In-built | User-specified (Resolution rule) | User-specified (Combiner function) |
| Access Patterns | In-built (Primary key to crowd columns or entire crowd table tuples) | User-specified, flexible | User-Specified, Flexible |
| Database-Awareness | Database aware of crowd tables and columns, CNULL values | Database aware of crowdsourced data, different semantics (fetch-resolve-join) | Limited awareness: functionality captured in UDF |
| Simplicity | Easy to understand | Complex data model and semantics | Easy to understand |
| Optimization Ability | Limited by way of storing cleansed data, fixed error resolution mechanisms | Not limited | Limited by way of lack of database awareness of crowdsourced data (less relevant for more recent papers) |
| Query Specification | No CNULLs in outputted data; LIMIT clause | constraints in SQL queries: AT LEAST / MIN COST / MIN TIME | Queries with UDFs |
| Objectives | No explicit user control of objectives | Limit the unspecified constraint (cost or time) | User explicitly controls redundancy and cost, database does not optimize this process |
| Typical Use-cases | Filling in missing values; data gathering | Filling in missing values; data gathering; data processing | Data processing; complex workflows |

Table 4.1: Summary of the declarative crowdsourcing systems

able to estimate the cost, latency, and error associated with each query plan; (b) the selection and tuning of different access methods (or tasks) to get data from the crowd; and (c) the satisfaction of optimization objectives, spanning cost, latency, error, or amount of data gathered.

4.2 Summary of Three Declarative Systems

In this section, we cover the three declarative systems. For each, we will cover the data and storage model, the error model, and the crowdsourcing model.

4.2.1 CrowdDB

We first cover CrowdDB, published first in SIGMOD 2011 [80].

Data Model. CrowdDB is like a traditional database, except that some columns (in one or more tables) are designated CROWD COLUMNS, and some tables are designated CROWD TABLES. The former designates the case where entities are fully known, but a few attributes of these entities have values that are unknown. The latter is when the entities themselves are not known. In the former case, only values of these attributes are crowdsourced, with the rest of the data assumed to be correct and fully known in advance. In the latter case, entire tuples are crowdsourced.

Consider the following example.

Department (University, Name, **Url**, Phone)
Professor (Name, Email, University, Department)

An assumption CrowdDB makes is that every table must have a primary key. In the example above, the Department table has primary key (University, Name), while the Professor table has primary key Name. The Department table is a regular table with the Url field (in bold) serving as the CROWD COLUMN, meaning that only the Url attribute is crowdsourced. The entire Professor table is a CROWD TABLE, therefore entire tuples are crowdsourced at a time.

Given these tables, the way the data is stored in CrowdDB is rather straightforward. Each regular table or CROWD TABLE is stored as a separate table, and missing attribute values (say of CROWD COLUMNS) are designated a special value CNULL (standing for CROWD NULL). Overall, the goal of query processing is to return tuples with no CNULL values.

Error Model. CrowdDB opts for a very simple mechanism to handle errors made by human workers. In particular, it expects that each CNULL value is crowdsourced from k separate workers, and then the plurality (i.e., the value that occurs the most number of times) replaces the CNULL value.

Crowdsourcing Model. CROWD COLUMNS are crowdsourced in the following way: When, for example, a particular Url of a given department is to be crowdsourced, crowd workers are provided the (University, Name) pair that makes up the the primary key, and asked to provide the Url.

CROWD TABLES are crowdsourced in the following way: workers are asked to provide entire tuples in one go. In the Professor example, workers are asked to provide the Name, Email, University, and Department of a professor all together. Note here that it is unlikely that someone try to crowdsource a table of all Professors in the world. Instead, a user might query for all professors at “Brown University,” and therefore crowd workers will be prompted to provide information specifically for records that match that value on the University field.

CrowdDB also admits the notion of Foreign Keys and uses the crowd to ensure that Foreign Keys are respected while new information is being added. We describe this using the example above, where the pair (University, Department) in the Professor table references (University, Name) in the department table. When adding a new Professor tuple for the University and Department columns, a crowd worker is only allowed to select values from a drop-down menu that already exist in the Department table. In this manner, the resulting database after the professor tuple has been added continues to respect Foreign Key constraints.

If the Department table was also a CROWD TABLE, we could ask workers to augment that table when new tuples are added to the Professor table, much like triggers that enforce continued adherence to Foreign Key constraints. For instance, if a worker suggests a new (University, Department) pair, they could be asked to also provide a Url and Phone for that pair such that it can be added to the Department table as a new tuple.

Overall. CrowdDB opts for a simple and easy-to-understand data model that can provide a lot of useful functionality without being overly rigid or losing out on performance.

4.2.2 Deco

We now cover Deco, whose first version was published at CIDR 2011 [145].

Data Model. Deco, like CrowdDB, is a traditional database system, augmented with crowdsourcing capabilities. Deco opts for a more general data model than CrowdDB, but is possibly harder for users to understand.

Consider the following example:

Restaurant(Name, Address, [Rating], [Cuisine])

AddrInfo(Address,[City, Zip])

In the Restaurant example, Name and Address of the restaurant are designated as *anchor* attributes, while Rating, and Cuisine form two groups of *dependent* attributes. In the AddrInfo example, Address is the anchor attribute, while City and Zip together form a dependent attribute group. At a high level, the anchor attributes describe the entities of interest, while each dependent attribute group (all of which are independent of each other) describe some properties of the entities of interest.

These two relations represent the *conceptual schema of the database*, i.e., these are what the end user (i.e., the requester / developer) sees and interacts with. However, what is actually stored, i.e., the *actual schema*, is not simply the two relations listed above. This represents a departure from CrowdDB, whose conceptual and actual schemas are identical. Thus, we expect that there is a *schema designer*, who designs the schema, and specifies some other information (described below), and there are many *end users*, who use the schema to pose queries to retrieve data on demand. These two roles may overlap, i.e., the end users may be the same as the schema designers.

Given these anchor and dependent attributes, the actual schema is the following:

RestaurantA(Name, Address)
 RestaurantD1(Name, Address, Rating)
 RestaurantD2(Name, Cuisine)
 AddrInfoA(Address)
 AddrInfoD1(Address, City, Zip)

As can be seen in the actual schema, there is a table corresponding to each of the dependent attribute groups: one for Rating, one for Cuisine, and one for City, Zip, as well as the anchor attribute groups: Name, Address in the first case, and Address in the second. For each of the tables containing the dependent attribute groups, one or more of the anchor attributes are present. For example, in RestaurantD1, both Name and Address are present, while for RestaurantD2, only Name is present. We describe the reason for this next.

Error Model. For each dependent attribute group, the schema designer provides a *resolution rule*. For our first table, we have:

$$\begin{aligned} \emptyset &\rightarrow \text{Name,Address} : \text{canonicalize}() \\ \text{Name,Address} &\rightarrow \text{Rating} : \text{avg}() \\ \text{Name} &\rightarrow \text{Cuisine} : \text{dupElim}() \end{aligned}$$

Here, the schema designer is specifying that any Name and Address that are crowdsourced are simply canonicalized; the Ratings that are crowdsourced are averaged, and the Cuisines that are crowdsourced are duplicate-eliminated. Consider $\text{Name, Address} \rightarrow \text{Rating} : \text{avg}()$. Here, the schema designer is specifying that for all sets of tuples that share the same value of Name and Address, we will take the average of all rating values that have been extracted or gathered from the crowd. That is, for all tuples that share the same value on the left hand side of the resolution rule, the right hand side values are aggregated using the function $\text{avg}()$. On the other hand, consider $\text{Name} \rightarrow \text{Cuisine} : \text{dupElim}()$. Here, the schema designer is specifying that for all tuples that share the same Name, all Cuisine values that have been gathered from the crowd are duplicate-eliminated. Lastly, for $\emptyset \rightarrow \text{Name, Address} : \text{canonicalize}()$, the schema designer has specified that all Name Address pairs are simply canonicalized. Notice that the schema designer can specify any error resolution mechanism they desire for each anchor or dependent attribute group, with the understanding that the resolution mechanisms for different attribute groups may be different: e.g., for string values like Cuisine, we may want duplicate elimination, while for numerical values like rating, we may want averaging or majority of k .

Also notice that the left hand side for the Cuisine resolution rule is simply Name: this is because Cuisine is only dependent on the Name of the restaurant and not the branch (and therefore Address), and therefore we do not need the Address on the left hand side of the resolution rule.

For the second conceptual relation, we have the following resolution rules.

$$\begin{aligned} \emptyset &\rightarrow \text{Address} : \text{identity}() \\ \text{Address} &\rightarrow \text{City,Zip} : \text{majority}() \end{aligned}$$

Overall, there is a one-to-one mapping between each resolution rule and each actual table in the underlying database. Readers interested in theory will notice if we treat each resolution rule as a MVD (multi-valued dependency), then the actual schema is a 4NF decomposition of the conceptual schema.

A key benefit of normalization is that we can store the raw values in the actual schema instead of the conceptual ones, and simply generate the “aggregated/cleansed” values on demand. We illustrate that below.

Crowdsourcing Model. Given the actual and conceptual schema, the schema designer provides a number of *fetch rules*, which are nothing but access methods or interfaces to get data using the crowd. Here are some examples of fetch rules:

$$\begin{aligned} &R.Name, R.Address \Rightarrow R.Rating \\ &R.Name \Rightarrow R.Cuisine \\ &R.Cuisine \Rightarrow R.Name \\ &\emptyset \Rightarrow R.Name, R.Cuisine \\ &R.Name, R.Address \Rightarrow \emptyset \\ &R.Address \Rightarrow R.Name, R.Rating, R.Cuisine \end{aligned}$$

For instance, the second fetch rule provides the Name of a restaurant, and asks the human worker to provide a Rating. The third fetch rule is precisely the opposite: it provides a Cuisine, and asks the human worker to provide a Name of a restaurant. We can also have \emptyset on the LHS or the RHS: the former indicating that the crowd worker should provide a new tuple without being provided any information, and the latter indicating that the crowd worker should confirm the tuple being provided as a YES/NO.

These different fetch rules allows users to specify whether fields should result in other fields, how to generate brand new records, and how to verify that the new records are legitimate. Fetch rules are quite powerful and expressive, and allow schema designers to specify a range of interfaces to get data from the crowd, which the system can then choose between.

The data obtained from these fetch rules is then added as tuples to the set of six actual relations: RestaurantA, RestaurantD1, RestaurantD2, AddrInfoA, AddrInfoD1, along with metadata, such as which worker provided that tuple, when that tuple was provided, and so on. The conceptual relation is materialized on demand, by taking the six actual relations, applying the resolution rules, and then performing a full outer join. This sequence of operations is referred to as the *fetch-resolve-join* sequence.

Overall. Deco uses a more powerful and flexible data, error, and crowdsourcing model than CrowdDB, with a range of expressive interfaces for gathering

data from the crowd, as well as expressive mechanisms to resolve mistakes made by the crowd. That said, Deco is also more complex than CrowdDB, and the additional power that comes from these aspects may not be necessary or warranted for simple applications.

4.2.3 Qurk

We now discuss Qurk, which was first published at CIDR 2011 [133].

Qurk, like CrowdDB, also opts for a simple design, layered on top of traditional database systems, by abstracting the involvement of the crowd as UDF (user defined function) invocations. The results of the UDF invocations can then be cached and used to avoid making future queries to the crowd, but are treated like any other data in the database. Qurk's design places the participation of the crowd even further outside the database than Deco and CrowdDB. In fact, Qurk's query processor could be completely unaware of the fact that it is operating on crowdsourced or non-crowdsourced data, unlike Deco and CrowdDB. All of the crowdsourcing-specific functionality captured within the UDF.

Consider the following example schema:

```
Photos(Id PRIMARY KEY, Picture IMAGE)
```

We will provide a query example using SQL, although the most recent instances of Qurk also facilitated development in the Pig[19] programming language:

```
SELECT * FROM Photos WHERE isSmiling(Photos.Picture);
```

The query above provides a list of Photos in the Photos table in which someone is smiling. The predicate `isSmiling` is a crowdsourced UDF. We will describe subsequently how Qurk makes it simpler to specify code for these UDFs using a higher level language.

Data Model. Qurk's data model is simple: it is the relational data model, along with the functionality to cache responses given by the crowd. The advantage of this data model is that Qurk can be used with all the traditional database systems, with the crowdsourcing functionality abstracted out into the UDF invocations.

Error Model. Once again, the error model adopted by Qurk is fully integrated into the UDF invocations. Qurk provides some basic error checking or resolution functionality inbuilt into some of its task templates, but this basic functionality can be overridden by designers. Further, Qurk does not explicitly specify whether the data stored in the database is cleansed or unclesed data. For instance, Qurk could explicitly store all the values gathered from the crowd (like Deco) or could store just the cleansed or resolved versions (like CrowdDB): this decision is made by the UDF.

Crowdsourcing Model. Instead of expecting the application developer to write code for UDFs, these UDFs can be described at a high level using *tasks*, which are high-level templates for common crowdsourced data processing operations, including filtering, sorting and joins.

Here is an example of a task for the UDF we saw above.

TASK isSmiling(Picture) TYPE FILTER

Prompt: “< img src='%s'>
 Is the person above smiling?”,

Picture

Combiner: MajorityVote

The Prompt keyword corresponds to the question that is asked to the crowd (akin to a fetch rule in Deco) taking as input the Picture being processed by the crowd. The question returns a YES/NO response. These responses are then combined using a MajorityVote, which is a special function that takes a number of YES/NO answers and takes the majority. This is similar to the resolution rule in Deco. If the developer does not specify a Combiner, the query calling this UDF will receive a complex response containing not only individual crowd workers' answers, but also information on the workers' identities, submission times, and work histories. These additional pieces of information allow Qurk users to use more advanced Combiners to aggregate responses.

Thus, a task consists of an interface description for a crowdsourcing operation at a per-tuple level, coupled with an error resolution mechanism also at a per-tuple level. Other tasks are provided for operations such as sorts and joins.

Overall. Qurk is different from Deco and CrowdDB in that the crowd-specific functionality is captured outside the database via UDF invocations. This has its advantages and disadvantages. The advantages include the following:

- The database need not be aware of the existence of crowdsourced data, and doesn't need to be modified for the sake of accommodating crowdsourcing operations such as error resolution, or crowdsourced access patterns. Also due to this, the database can be replaced by a different or newer one, and we can seamlessly inherit all the improvements that come from the new version when operating on traditional relational data.
- To crowdsource, we do not need to touch the database layer at all. All the crowd-specific functionality can be captured within the UDF, and the UDF takes care of accessing the crowd and resolving mistakes made by the crowd.

The major downside is that the lack of database awareness of crowdsourcing aspects limits query optimization involving both crowdsourcing and relational data processing. Thus, the optimization decisions are encapsulated within the UDF.

In recent work by the Qurk team [130], they have extended Qurk allowing the database to be more aware of the existence of crowdsourced UDFs, enabling better overall query optimization. This move brings Qurk closer to a hybrid between the former UDF-centric design, and the designs of CrowdDB and Deco.

4.3 Design Decisions

We now describe the design decisions made by each of the three systems in a variety of dimensions, shown in Table 4.1.

4.3.1 Integration and Data Crowdsourced

CrowdDB and Deco treats crowds as a *data source*, i.e., an access method to gather data that can be added to the database. The CrowdDB team was the first to articulate that unlike traditional data sources, this data source violates the *closed world assumption*: tables do not start off with a finite set of records that represent all of the records the crowd will eventually provide. Instead, we operate under the *open word* paradigm, where at any point there might exist

more data to be crowdsourced outside of a table. On the other hand, Qurk abstracts interactions with the crowd into a user-defined function (UDF), rather than an access method.

The different integration mechanisms lead to differences in the type of data that is crowdsourced: for CrowdDB and Deco, the view of crowds as data sources allows them to both gather new data from the crowd (e.g., gathering 100 flavors of ice cream) and coordinate crowds operating on data (e.g., checking if an email message is spam). For Qurk, the view of crowds as predicate evaluators focuses the attention on the second type of data, and makes it more challenging to gather new data with the crowd. It is indeed possible to instrument Qurk to get data of the first type, but this would require UDFs to produce entirely new tuples and table-valued functions, which might be challenging for end-users. We list these two aspects in the first two columns of Table 4.1.

4.3.2 Data, Error, and Crowdsourcing Model

As described above, CrowdDB opts for simple choices for the data, error, and crowdsourcing models: the data model is the relational model, with CNULLs capturing NULL values to be filled in with crowdsourced values and only cleansed data is stored; the error resolution mechanism is a majority vote; and the crowdsourcing model involves two types of access methods: one for crowdcolumns and one for crowdtuples.

Deco opts for more complex/general choices for the data, error, and crowdsourcing models: the conceptual schema is not the same as the actual schema, with dirty data stored and resolved; the error resolution mechanism is specified by the user on a per-column group basis; and the data can be crowdsourced using a variety of access methods.

Lastly, Qurk can store either the cleansed or the dirty data; uses a user-specified error resolution mechanism; and like Deco, allows data to be crowdsourced in many ways. We list these three aspects as columns 3–5 in Table 4.1.

The advantage of storing dirty data is that data can be cleaned to the specifications of each query: for instance, if a higher accuracy or a different resolution rule is needed, then existing crowdsourced responses can be utilized. Furthermore, the dirty data that may have gotten stale (e.g., crowdsourced

phone numbers of restaurants) can be removed by the system periodically. Lastly, storing dirty data allows us to reason about the relative confidence of the answers: which only storing cleansed data does not permit us to do.

The disadvantage of storing dirty data is that there is the need for a resolution mechanism to translate this dirty data into cleaned data: this means the schema designer needs to reason about two different schemas and provision for both. Furthermore, this dirty data needs to be “resolved” on demand when queries are posed. Both Qurk and Deco apply resolution rules as answers come in rather than waiting until all the answers are gathered.

Notice that there is a wealth of work in probabilistic databases: Trio [194] handles data with uncertain values and lineage, and presents a language for querying this data. Dalvi and Suciu explore efficient queries over probabilistic databases [62]. BayesStore [188] takes this a step further, adding complex inference functionality to databases with probabilistic data. MauveDB [71] explores generating model-based views over tables in order to model, clean, and perform inference over the data.

None of the three declarative crowdsourcing systems opted to use those as an underlying data model: this is because application designers using crowdsourcing often have very specific resolution models in mind, e.g., if 3 or more out of 5 of the workers agree on this answer, then I am willing to accept it as the “final” answer. Thus, reasoning in probabilities is often not helpful in practice. Instead, the systems described here allow programmers to specify how to convert multiple potentially different worker responses for the same task into a single definitive answer, for example by keeping the response indicated by a majority vote.

Access methods form another important aspect of declarative crowdsourcing systems: the greater the variety of the access methods, the quicker the system can return results for a query. For example, if our query requested four mexican restaurants in Seattle, Deco could use expressive fetch rules from `Cuisine ⇒ Name` to return results quickly, while CrowdDB would need to keep getting complete tuples of restaurants until four mexican restaurants happened to be gathered. Qurk does not limit the kind of access methods specified or used within the UDF; however, by virtue of crowdsourcing being limited to a UDF, it does not easily permit the crowdsourcing of entirely new data.

4.3.3 Complexity, Optimization Ability

In terms of complexity, Deco is clearly the most complex, and therefore the hardest to understand, with the user-facing conceptual schema being different from the actual schema. CrowdDB is the simplest, by not surfacing any of the crowdsourcing aspects to the end user: the end user does not need to provide access methods or fetch rules, they do not need to describe how to deal with errors made by human workers. Qurk is somewhere in the middle, because it surfaces a lot of the crowdsourcing aspects to the end user, such as specification of the error resolution function, which interfaces are used, and so on.

Due to the many interface types it offers along with the resolution mechanisms, Deco offers the most optimization abilities. CrowdDB is limited because it only uses a small set of access methods to gather data from the crowd. Lastly, Qurk is less limited than CrowdDB on interfaces but is limited by the lack of the database awareness of crowdsourced data. Thus, holistic optimization is challenging in Qurk, though its authors did look at certain optimizations such as crowd-powered selectivity estimation in a later paper [130]. We list these aspects in columns 7 and 8 in Table 4.1.

4.3.4 Specification, Objectives, and Use Cases

In CrowdDB, queries are specified as standard SQL, with possible LIMIT clauses that limit the amount of tuples that are output. (This is necessary because the open world assumption means that the number of tuples could be unbounded.) There is no explicit user control of other objectives.

In Deco, queries are specified as standard SQL, with one of either AT LEAST, MIN TIME, or MIN COST, the first of which means that at least a certain number of tuples are produced, the second means that at most a certain amount of time is taken, and the last means that at most a certain amount of cost is used.

In Qurk, queries are specified as standard SQL, with UDFs capturing the crowdsourcing aspects. There is no explicit user control of other objectives, except via the UDF.

Overall, CrowdDB is tailored toward applications that require data completion—filling in missing values or attributes, or gathering unknown

data. Deco is tailored toward data completion or data processing. Qurk is tailored toward data processing, as well as composing more complex workflows.

4.4 Unresolved Issues

Even though the systems described thus far have captured a range of essential functionality, and form an important first step in tackling the problem of declarative crowdsourcing, there are still many unresolved issues and inefficiencies. We describe each of these in turn.

4.4.1 Insufficient Optimization

None of the systems described above are able to find a globally optimal query plan. It is still unclear how to best involve humans within computation; in fact, even if we fixed the set of operators or interfaces that we can use, it is still not straightforward.

All three systems do employ some limited forms of query optimization, described below:

- All systems use some kind of asynchronous execution to deal with the varying times for crowdsourced execution versus traditional computation. That is, a batch of tasks is generated, and then the system “reacts” to responses as they come in. Furthermore, due to the long response time for crowd work, it makes sense to issue more questions in parallel, and then respond when the answers come in.
- All systems use some kind of predicate pushdown, to ensure crowdsourcing is only used if absolutely necessary; for example, if there is an automatically evaluable predicate, it should be evaluated first before crowdsourcing is used to avoid extraneous crowdsourcing work (which is both time consuming and costly).
- Deco picks a query plan with appropriate choices of interfaces or fetch rules to ensure that as little monetary cost is used as possible in completing the query. A sub-problem of interface selection was shown to be NP-Hard [150].

- All systems use some form of batching of questions that are sent out to crowd workers to reduce cost. CrowdDB also identified that worker response times depend on the number of outstanding tasks in Mechanical Turk (crowd workers appear to prefer task types that are bountiful), so sending out one task at a time is inefficient and time-consuming.
- Qurk uses features to estimate selectivity of various crowdsourcing predicates on-the-fly, and uses that to order predicates within query plans.

4.4.2 Capturing Metadata

Another aspect that none of the systems have completely solved is that of capturing worker metadata, like worker IDs, times at which the data items were retrieved, the time taken for workers to respond, and so on. It is essential to record this information to reason about quality of workers and worker responses holistically.

CrowdDB does not record any metadata. Deco records some of this information along with each response in the actual schema, which contains raw information. Qurk also records some of this information in its cache.

4.4.3 Grouped Resolution

All three systems perform error resolution on a per-tuple level—CrowdDB takes a majority of k , Deco has its resolution rules, and Qurk has its combiner functions. In practice, however, when employing EM-based quality management techniques which jointly identify optimal estimates for worker and answer quality (described in detail in Section 2.3.2), it is essential to reason about all answers for all workers simultaneously. None of the systems allow us to do that because they process data in a more streaming per-record fashion. The current EM-style algorithms for quality management are *one-shot*, meaning that they can be only used once. Any further decisions made based on the result of the algorithms are biased and lose accuracy over time. To the best of our knowledge, there is no work on adaptively estimating worker quality while simultaneously deciding on the next questions to ask.

Another place where per-tuple treatment affects us is when we want to perform Entity Resolution [192] across all values of, say, restaurants. Even

if we are not interested in the worker quality, and are simply interested in resolving the set of answers, none of the systems natively support the joint estimation of worker quality and answer quality — in a manner similar to that performed by the Expectation Maximization algorithm.

4.5 Summary

As indicated above, there is still a lot of work to be done in building the ‘ideal’ declarative crowdsourcing system, much of it in exploring the optimization issues. One of the inherent challenges is that we are as yet not aware of the capabilities of human workers; but we can certainly build a useful and usable system even by using a limited set of operators.

5

Survey of Industry Users: Summary and Methodology

In this chapter, we begin describing our survey of industry users of crowdsourcing. We begin with an executive high-level summary of our findings (Section 5.1), discuss our survey methodology (Section 5.2), and describe three categories of industry users we identified (Section 5.3).

Over the next few chapters, we describe the detailed survey results, specifically:

- **STATISTICS:** the statistics of the crowdsourcing deployments (Section 6),
- **USE CASES:** the applications crowdsourcing is typically used for, approaches adopted before the advent of crowdsourcing, and the benefits provided by crowdsourcing (Chapter 7),
- **DETAILS:** ensuring quality, incentivization, task design, and other deployment challenges. (Chapter 8)

We begin with a high level summary of our findings; given the extent of our undertaking, it is impossible to summarize all of our takeaways into one section. We therefore encourage even the impatient reader to at least skim all four chapters, with a special eye towards the excerpted quotes and tables; the

former containing special anecdotes, use-cases, or lessons learned, the latter containing summaries of the findings of the sections within the chapters.

5.1 Executive summary

Crowdsourcing is common. Crowdsourcing is alive and well at the companies we spoke with: none are indicating that they are decreasing their investment in it, and almost all of them were looking for new ways to use it.

Crowdsourcing deployments are large-scale. The statistics on crowd management and the teams that build crowd-powered systems were illuminating (Section 6). At their largest, participants reported hundreds of employees deploying hundreds of thousands of tasks per week, with overall spending in the millions of dollars per year. The number of employees building crowdsourcing-oriented tools ranged from 1 company-wide to “tens to hundreds” at larger organizations. The participant with the highest paid task throughput reported processing about 400,000 tasks per week, and the most popular response across participants was in the low tens of thousands of tasks per week. At the low end, participants reported spending \$300-\$1000 per week. The two largest participants that provided us with numbers reported spending approximately \$10,000 and \$30,000 per week respectively.

Many users host their own platforms, with long worker relationships. Five participants hosted their own crowd work platforms (i.e., they use an intermediary or outsourcing company to hire workers who work on tasks provided by the participant 9–5). The ubiquity of internal crowd work platforms was one of the most surprising findings from our study; indicating that academic research, which is focused on popular platforms like CrowdFlower and Mechanical Turk, has summarily ignored one of the most common industry mechanisms for employing crowdsourcing. Apart from the five that only use their internal platform, five utilized an external provider’s platform, and two did a mix of both. When participants hosted their own platform, the length of relationship with workers was high: The most common response for maximum worker tenure was 3 years, with medians between 1 and 2.5 years. At the low end, participants interacted with 50-100 workers per week, and at the high end, participants interacted with 100s to low 1000s of workers in a week.

Many new and novel uses of crowds. Some companies are doing interest-

ing and novel things with the crowd (Section 7.1). For example, one participant has such a trusting relationship with a few hundred crowd workers that the participant feels comfortable pre-paying the workers to monitor the news for updates on companies of interest and updates crowd workers' balances as they send back "new" facts. Yet another participant uses crowdsourcing frameworks on in-house employees, who work on tasks whenever they get time free from their regular work.

Classification and entity resolution are most popular uses of crowds. While there are many interesting use cases, most of the ones participants described are relatively standard. The two most popular use cases are classification and entity resolution. There are often large teams that use crowdsourcing for only one targeted application (e.g., categorization, or data extraction); they have spent many months tuning their deployment for this application, and use it periodically.

Most problems solved by crowdsourcing are unsolvable without it. We wanted to understand how crowdsourcing was perceived at various companies. When asked how they solved problems before the advent of crowd work, a little less than half of the responses were of the form "our company didn't exist at that point," or "we didn't solve this problem before crowdsourcing" (Section 7.2). When we asked participants to explain some of the benefits of crowd work, the top three responses pointed to the flexibility to scale work up and down, the low cost, and that crowdsourcing enabled previously difficult or impossible tasks (Section 7.3). One participant told us that the main benefit he derives from crowd work is that he doesn't need to "argue with management" to hire the manpower to get things done; he can simply use small amounts of money as the need arises.

Quality management schemes are somewhat primitive. Most participants use very simple schemes, such as majority vote over multiple worker responses, to remove errors. However, more than half of the participants do use some form of simple Expectation-Maximization scheme to reason about worker error rates and task answer quality (Section 8.1). Most participants do not do any optimization to reduce cost while keeping accuracy fixed.

Incentivization schemes are primitive. The most common methods for incentivizing workers were financial: per-task payment was the most popular,

followed by bonuses and hourly payments (Section 8.2). Less tangible incentives, like gamification, leaderboards, and promotions, were less popular.

Industry users rarely use workflows/toolkits from academia. Finally, while we have so far described how participants successfully used crowd work, researchers might also wonder how some of the more complex approaches from the literature have fared in practice (Section 8.3). Surprisingly, no participants utilized third-party frameworks to simplify their crowd-powered data processing workflows. Many participants reported that none of their workflows have more than crowdsourcing step, suggesting that participants are looking for simple tasks to be completed rather than the more in-depth multi-stage workflows prescribed by crowdsourcing researchers. In support of this observation, less than a third of participants claimed to use at least one crowdsourcing “design” pattern (iterative refinement, find-fix-verify, or do-verify—we will describe these later) from the literature.

As you read this chapter, watch out for various tables and pull quotes to guide you through our findings. Much of the summary above has been extracted from these pull quotes, and we hope they can serve as guideposts for your reading.

5.2 Survey and recruitment methodology

Here, we describe our survey methodology for both the survey of industry users, as well as the survey in Chapter 9 of marketplace providers.

With a goal of identifying the key use cases, existing solutions, and open research problems in the field of crowd-powered data processing, we conducted surveys of two groups of stakeholders: 1) industry users of crowdsourcing, and 2) operators of crowd labor marketplaces. Each survey consisted of approximately 45 minutes worth of questions. Participants had an option of taking the survey synchronously by phone or asynchronously over email. When a participant requested a phone interview, the authors took notes and coded the responses on behalf of the participant.

To generate the surveys, we first created a list of topic areas we wished to learn more about, and then iteratively generated questions, recategorizing questions into topic areas as appropriate. We received feedback on the questions from an expert in survey generation. For the industry survey, we piloted

| Description of Survey Section | # Questions | Paraphrased Example Questions | Chapter |
|-------------------------------|-------------|--|---------|
| Crowd Use Cases | 4 | — Which of the following use cases of crowds apply to the tasks your team is solving? (examples include classification, text generation, etc.) — How did you solve these problems before crowdsourcing? | 7 |
| Crowd Management Statistics | 6 | — How many people in your organization work on crowdsourcing? — How many tasks per week do you generate? | 6 |
| Quality of Work and Workers | 5 | — How do you evaluate worker quality? — Do you provide feedback to workers? | 8 |
| Incentives/Payment Mechanisms | 3 | — Do you pay workers hourly or per task? — Are there different classes/tiers of workers? | 8 |
| Task Design/Decomposition | 7 | — What crowd management frameworks do you use? — Do you primarily create microtasks or macrotasks? | 8 |

Table 5.1: A summary of the types of questions we asked participants either through a survey they filled out on their own time or through phone interviews. The example questions provided are paraphrased descriptions. Detailed questions can be found in Appendix A.

the survey with one participant by voice and one by email and further clarified the questions based on confusion that arose during the interviews, but kept the responses from the two pilot participants. There were too few marketplace participants (4) for a pilot phase, but we conducted the marketplace surveys after the industry ones, and were able to clarify marketplace questions from that experience.

The industry survey consisted of six sections: use cases, infrastructure for managing workers, tools for inferring worker and work quality, incentive and payment mechanisms, task design and decomposition, and a ranking of challenges. The marketplace survey covered crowd demographics, descriptions and summary statistics of common implementations/use cases, and worker and work quality assurance. The chapters that cover the results of each section of the survey are also listed. Table 5.1 describes each section in more detail, and the industry and marketplace surveys can respectively be found in Appendices A and B.

To recruit participants, we surveyed our own social networks and the crowdsourcing literature (conferences, workshops, and blog posts) for industry participants, and contacted them for their participation. To expand the

| Company | Team | Persona |
|------------|--|-------------------------------------|
| Amazon | Product classification | Largely single-case user |
| Captricity | Focus of large part of company | Largely single-case user |
| Dropbox | Single person consulting several teams | Multi-case user / Internal provider |
| Facebook | Entities team | Multi-case user |
| Flipora | Startup CTO | Multi-case user |
| GoDaddy | Small business data extraction | Multi-case user |
| Groupon | Merchant data team | Multi-case user |
| Google | Internal crowdsourcing team | Internal provider |
| Google | Web knowledge discovery team | Multi-case user |
| LinkedIn | Single person consulting several teams | Multi-case user / Internal provider |
| Microsoft | Internal crowdsourcing team | Internal provider |
| Microsoft | Search relevance team | Multi-case user |
| Youtube | Crowdsourcing team | Largely single-case user |

Table 5.2: A summary of the company and persona of the team that we spoke with in that company. Some organizations (e.g., Microsoft, Google) are so large that we were able to speak with both a multi-case user and an internal provider. Note that some teams (e.g., Dropbox, LinkedIn) were largely composed of a single person that both implemented crowdsourcing solutions and consulted other teams on crowdsourced implementations.

scope of the survey, we asked that initial set of participants for any relevant connections that they had. Participants were ensured that their responses would only be reported in aggregate, except for meaningful quotes that they would be allowed to review. Table 5.2 identifies our survey participants.

5.3 Three personas of industry users

After evaluating responses to the industry user survey, we identified three team personas that we later use to summarize some of our findings. While these personas don't always utilize crowd work similarly, their behaviors of teams with the same persona are often similar.

Internal providers (4/13). These teams serve as tool- and service-builders for other crowdsourcing users within their company. They are often the go-to team that provides consulting in addition to the tools that they build. As a sign of the breadth of their experience, three of the four internal providers we surveyed saw every data processing use case that we listed. These specialized intermediary teams are more common in larger companies with varied needs.

Multi-case users (8/13). These teams directly solve problems with crowd workforces, rather than serving as intermediaries as the internal providers do. Forming the largest set of participants by far, these teams do not see as many use cases as the internal providers, but have end-to-end experience solving several problems for their companies.

Largely single-case users (3/13). These users use crowds in a small number of workflows for primarily one task within their company or team, and can often reflect on several iterations of their solutions to this one problem. The small number of use cases should not be conflated with less experience with crowd work: one of our largely single-case participants consistently generates hundreds of thousands of tasks per week, amongst the largest task generation volume of any participant.

Some organizations (e.g., Microsoft, Google) are so large that we were able to speak with both a multi-case user and an internal provider. Note that some teams (e.g., Dropbox, LinkedIn) were largely composed of a single person that both implemented crowdsourcing solutions and consulted other teams on crowdsourced implementations.

6

Survey of Industry Users: Crowd Statistics and Management

We now turn to descriptive statistics around how different organizations manage crowd work, including the makeup of the teams building crowd-powered data processing systems, the scale of such systems, and various payment and recruitment methods. When possible, we describe trends we observed amongst the three personas described in Section 5.3.

We asked participants to identify their use of both explicit crowd work (e.g., paid workers, volunteer workers) and implicit crowd work (e.g., posthoc analysis of user email spam tagging or user search logs). In the former case, workers are aware that they are working on “tasks” and are explicitly recruited for that purpose, and in the latter case, some by-product of user activity is used to generate useful data. Every single participant had at least one explicit crowd workflow, whereas only a subset made use of implicit user traces. For the remainder of this section, we explore participants’ responses to their uses of explicit crowd work, while also noting the fact that implicit crowd work is also used by many of the participants.

In the next few sections, we cover various statistics surrounding real-world crowdsourcing deployments.

- **SIZE OF TEAMS.** We describe how large teams building large scale crowdsourcing pipelines or workflows can get in Section 6.1.

- **TASK THROUGHPUT.** We describe the number of tasks performed by workers working for each of these participants in Section 6.2.
- **SPENDING.** We describe how much money these teams spend on crowdsourcing in Section 6.3.
- **REDUNDANCY.** We describe how different users use redundancy to ensure worker quality in Section 6.4.
- **RECRUITING.** We describe where these teams recruit workers from in Section 6.5.
- **TENURE.** We describe length of employment in Section 6.6.

6.1 Size of systems-building teams

The number of employees ranged from 1 company-wide to “tens to hundreds” at larger organizations, with a median of 4 and a mode of 2.

One of the most notable observations was that, in some cases, the cost of compensating the internal teams that are using crowdsourcing for various tasks or building crowdsourcing platforms was often comparable to the amount of money spent on paying crowd workers for completing tasks.

This was initially very surprising to us. That said, this observation is understandable given the nascency of crowd work: until more generally applicable systems (possibly advanced, more optimized versions of the systems described in Section 4) are available, organizations will invest deeply in the technology to get their crowd-powered workflows right. That said, it is also possible that building and iterating on crowd workflows will remain a “dark art,” and that individuals who do a good job designing such workflows or deployments will remain well compensated.

Thus, while crowdsourcing enables organizations to scale their data processing tasks beyond what would be possible with in-house employees, building crowd-powered data processing systems requires an investment in full-time engineering and operations teams. We asked participants how many full-time employees in their organization work on these systems.

The number of employees ranged from 1 company-wide to “tens to hundreds” at larger organizations. Overall, a median of 4 full-time employees worked on crowdsourcing systems development teams, with a mode of 2 full-time employees.

Amongst the internal providers persona, team sizes ranged from 1 to 30. The largest internal provider maintained a team size in the range of 20-30, with approximately 150 employees building products and tools that depended on the internal crowdsourcing platform. For the multi-case user persona, team sizes range from 2 to “tens to hundreds,” with team size commensurate with the size of the company we spoke with. Finally, the largely single-case crowdsourcing user teams ranged in size from 2 to 21.

These numbers are striking in their variance. While we make no claims to statistical significance, it is worth noting that we saw no obvious relationship between team size does and persona. Team size seems to be more dependent on the particular solution set and company organizational style than the scale of the problem faced by each organization. In some smaller organizations with largely one use case, an entire company might see itself as a crowdsourcing organization, whereas larger organizations might depend on a single in-house expert.

In some of the organizations making a larger team size investment in crowdsourcing, their investment is quite high. If we conservatively underestimate a cost of \$100,000 per year to employ a full-time engineer, the largest organizations are investing \$1-\$10 million dollars per year in talent alone.

6.2 Task throughput

The participant with the highest paid task throughput reported issuing about 400,000 tasks per week, and the most popular response across participants was in the low tens of thousands of tasks per week.

We asked participants how many tasks per week workers complete for their purposes, not including redundant tasks for quality purposes (We cover this in Section 6.4). Six participants responded, with a minimum of around 1000 tasks a week and a maximum of millions of tasks per week. The par-

participant processing or issuing millions of tasks per week makes heavy use of volunteer explicit crowd work. The participant with the second-highest throughput processes about 400,000 paid tasks per week, and the most popular response across participants was in the low tens of thousands of tasks per week. Note that none of the internal provider persona participants answered this question; one can only guess at the scale of these internal operations.

One participant (multi-use case persona) underscored the non-uniformity in task throughput. When training models based on crowd worker input, they complete around 2000 tasks per day for several months, going to zero once they have enough training data.

6.3 Monetary spending

At the low end, participants reported spending \$300-\$1000 per week. The two largest participants that provided us with numbers reported spending approximately \$10,000 and \$30,000 per week respectively.

We asked participants approximately how much money they spent per week on paying crowd workers. At the low end, participants reported spending \$300-\$1000 per week. The two largest users that provided us with numbers reported spending approximately \$10,000 and \$30,000 per week respectively, and we note that most of the largest teams elected not to report a figure. Several organizations reported spikes in their spend rates: it was not uncommon to hear of temporary order-of-magnitude increases in spend rates to solve a particular problem in a short period of time.

We see that some organizations spend as much or more on building crowd-powered systems than they do on compensating the crowd for their work.

Given our previous estimates that the largest organizations spend upward of \$1-\$10 million dollars a year compensating their crowdsourcing system-builders, we see that some organizations spend as much or more on building crowd-powered systems than they do on compensating the crowd for their

work. From these numbers, it is clear that crowdsourcing can be a relatively large investment for an organization, and that organizations are willing to make those investments for the benefits for crowd work. The numbers also indicate that there is room for libraries and systems that might simplify and reduce the investment required for organizations build crowd-powered data processing workflows.

6.4 Task redundancy

In addition to the number of tasks completed, we also prompted participants to explain how much redundancy they built into their tasks. These answers varied by application area. For those participants that utilized redundancy to improve result quality of objective responses, participants tended to seek out a minimum of 3 and a maximum of around 10 redundant responses per task. Other respondents that were looking for a broader sample of responses of subjective questions or ratings asked anywhere from 15 to hundreds of respondents to provide feedback.

One participant, in “exploration mode” would set task redundancy between 3–10 until they identified good workers, and then had little or no redundancy within the pool of vetted workers.

When participants could plan for a longer-term investment, they changed their strategy. One participant, in “exploration mode” would set task redundancy between 3 and 10 until they identified the best workers, and then had little or no redundancy within the pool of vetted workers. Another participant that opted for a hierarchical review strategy [91], where vetted workers would review or spot-checked entry-level workers, reported 1-2 workers interacting with any task on average.

6.5 Recruiting

We asked participants about their recruitment techniques for crowdsourcing. Our findings largely broke down along these dimensions, expanded below.

Five participants hosted their own crowd work platforms, five utilized an external provider's platform, and two did both.

In-house vs. external platform. Before we started the survey of industry users, we were already aware of some companies using in-house platforms, i.e., having a crowdsourcing platform built in-house, where only tasks from that company are issued, and answered by workers possibly either working within the auspices of the company, or from an external (trusted) outsourcing company. These crowdsourced workers are trusted workers, and work on the companies tasks from 9–5, much similar to a full-time job. So we decided to explore how many participants use their own in-house platforms versus external ones, operated by companies like Amazon (Mechanical Turk) or CrowdFlower. Of 13 responses, five participants hosted their own crowd work platforms, five utilized an external provider's platform (limited to Mechanical Turk and CrowdFlower), two did a mix of both, and one provided an unclear response.

Generally, we found that the larger the organization, or the more complex the problem, the more likely the participant was to use an internal platform for task distribution. For simpler tasks, and especially for microtasks, participants trusted external provider's platforms for task completion. Companies using crowdsourcing at scale often preferred to have their own platform, possibly to have fine-grained control, hiring, retention, and firing of workers and to deal with sensitive information (as we will see below).

Companies using crowdsourcing at scale often preferred to have their own platform, possibly to have fine-grained control of hiring and firing of workers and to deal with sensitive information.

One vs. many sources of crowd workers. Next, we wanted to see if the participants we surveyed used multiple platforms for soliciting crowd work, or used only one, and if they did use multiple platforms, how did they divide work across these platforms. Of the 13 responses we received, eight participants sourced their workers from multiple different crowds (e.g., Upwork and Mechanical Turk), four went to a single source, and one response was

unclear. Some participants had a notion of overall quality or specialties of particular crowds. One participant claimed that they believed vendors/out-sourced contractors (via say, an in-house platform) to be more trustworthy than Mechanical Turk. An in-house platform also enabled processing of that was sensitive in nature, but we also noted some historical effects, where an organization that started with in-house workers might stick with them for task and content types that other participants used external contractors for. Another participant went to Mechanical Turk for the greatest pool of workers with useful filters (e.g., geographies or quality) for targeting particular types of workers. Finally, one participant used a mix of Upwork, CrowdFlower, and temporary contractors depending on the type of task they were completing.

This question also resulted in a number of interesting one-off responses. One participant, who had one of the highest task throughputs we encountered, did not limit their recruitment on Mechanical Turk to workers of a particular quality rating or geography, but narrowed down the prospects with qualification tests and internal grades. Other participants utilized a mix of worker filters for geographies and quality ratings, but also made use of task-specific qualifications. A participant that had utilized Mechanical Turk for several years and established relationships with some workers set up a mailing list to advertise new tasks.

Two participants utilized a feature of CrowdFlower (described in detail in Section 9.2.1) that allows requesters to make use of the CrowdFlower task distribution platform but provide their own employees or workers. This feature points to a future for systems-building for crowd-powered data processing: some organizations, even if they are particular about their use of in-house employees, find benefit in using pre-built systems to distribute work and estimate work quality.

6.6 Crowd worker tenure

The most common response for maximum worker tenure was 3 years, with medians between 1 and 2.5 years. At the low end, participants interacted with 50-100 workers per week, and at the high end, participants interacted with high hundreds to low thousands of workers in a week.

We prompted participants to expand on how many crowd workers interacted with their workflows each week, and what the median and maximum tenures of their workers were. In the eleven responses, two groups arose.

Black box/unsure. The first group used services like CrowdFlower or Mechanical Turk, or even in-house platforms operated by outsourcing firms, abstracting away their direct interaction and recruiting of crowd workers. Many, but not all, of these respondents explained that they didn't know how many workers they interacted with in a week, or how long a relationship they had with those workers. Tenure and quality is still important on these systems, but these notions are abstracted from the end-user. So while marketplaces still optimize for tenure and quality, the users of these intermediary marketplaces largely trust marketplaces' abstractions.

Long-term/self-aware. The second group sought out contracts with workers on platforms like Upwork, or revisited their highest-performing workers on platforms like Mechanical Turk. As a result, they were more aware of their relationship length and number of workers. The length of the worker relationship in these systems was striking and largely correlated with the lifespan of the organization: the most common maximum worker tenure response was 3 years, with medians between 1 and 2.5. At the low end, these participants interacted with 50-100 workers per week, and at the high end, participants interacted with high 100s to low 1000s a week. These numbers did not seem to have a relationship with the participant persona: the scale and value of the problem tended to indicate the scale of the investment in the crowd.

When participants relied on microtasks with worker relationships and quality managed or abstracted away by marketplaces like CrowdFlower, they tended to have less of a sense of the depth of their relationship with crowd workers.

For some participants, we saw both forms of responses. When participants relied on microtasks with worker quality managed by marketplaces like CrowdFlower, they tended to have less of a sense of the depth of their relationship with crowd workers. When they managed longer-term investments in contractors (such as on Upwork), they tended to know how many hundreds of workers they had interacted with, often for many years per worker.

7

Survey of Industry Users: Use cases and Prior Approaches

In this chapter, we focus on the value proposition offered by crowdsourcing, especially for large-scale data processing. We begin by capturing the various use cases (Section 7.1), then cover the prior approaches that these participants used before the advent of crowdsourcing (Section 7.2), and then describe the benefits of crowdsourcing as explained by the participants (Section 7.3).

7.1 Use Cases

As part of the industry survey, we asked participants to identify some common crowd-powered data processing use-cases. The goal of identifying use cases is to contrast what is actually used in practice with the algorithms that academics have focused on developing crowd-powered variants for in Chapter 3.

We provided participants a list and asked them to select options from this list, and also allowed them to provide their own. For each of the use-cases identified, we asked them to expand on details using free text.

Of the provided use cases, all but one of the use cases were selected by at least one participant. Summary statistics on common use cases can be found in Table 7.1. Note that the use-cases we list in that table do not have a direct

| Use case | # Participants | Examples |
|--|----------------|---|
| Classification and categorization (all except spam detection/content moderation) | 12 | User classification, business vertical classification, sentiment analysis on product reviews, song genre classification, structured data extraction, website categorization |
| Spam detection/content moderation | 5 | Web/email/comment spam, adult/offensive content, illicit search engine optimization schemes, copyright violations. |
| Entity resolution/matching | 6 | Identifying whether two business listings are the same, or identifying businesses, people, places, celebrities, or movies in unstructured text |
| Ranking and relevance | 5 (unprompted) | Image/video-based multimedia ranking to surface high-quality content to users, or search relevance in areas such as web or domain name search. |
| Data cleaning/normalization | 5 | Canonicalizing/normalizing data to a style guide, finding authoritative sources to vet extracted facts, verifying previously extracted facts to ensure accuracy, identifying new facts about existing entities. |
| Data extraction | 5 | Digitizing paper forms, converting unstructured data embedded in photos, PDFs, Word documents, HTML documents, and Flash animations to structured data, and lead generation. |
| Text generation | 5 | Researching a company or product and writing a blurb about it, summarizing news articles, or rewriting existing content. |

Table 7.1: Crowd use cases, number of participants, and examples that our participants reported. Note that all options were enumerated for participants in a checklist except for *ranking and relevance*, which participants reported without prompt. While the last five use cases were all cited by five participants, the participants for each use cases did not always overlap.

one-to-one relationship with algorithms that we described in Chapter 3. This is by design: instead of using technical terms that are familiar to researchers in crowdsourced data processing, we opted instead of simpler terms that would be easily understood by developers and data scientists. For instance, instead of using the term *filtering* (often used in database parlance), we opted instead for *classification* (more common in machine learning and data science). We will add footnotes wherever appropriate to shed light on the mapping throughout this chapter.

As can be seen in the table, almost all participants made use of crowds for some form of classification and data cleaning/normalization. While we did not include it as an option in our use case list (an omission on our part), many participants self-reported crowd-powered ranking/rating as a common use case. The one option we provided that no participant selected was schema mapping, and while we know of efforts to crowdsource such work in indus-

try¹, we were unable to get feedback from companies that use crowds for schema mapping. In the sections to follow, we dive into the details of various use cases.

We begin with unique and notable use cases (Section 7.1.1), and then cover each of the more traditional use cases—corresponding to rows of Table 7.1—in turn.

7.1.1 Unique and notable use cases

While the details of common crowd-powered data processing use cases are illuminating, we also found a few long-tail gold nuggets. These creative and unique uses are exemplars of the breadth of capabilities that crowds enable. We now highlight some of the notable cases.

Continuous Queries and Pulse Checks. One of the most exciting scenarios we heard of was from a small multi-case user. Because of a long-term trusted relationship with about a hundred crowd workers on Mechanical Turk, the participant is able to pre-pay crowd workers to monitor and track changing facts in the world. For example, the participant might prepay a worker around \$20 to monitor a set of companies and alert the participant when any key employees change. For every correct update, the worker can subtract \$1 from their running tab, and as the balance becomes low, the worker can request another allocation. Whereas most crowdsourced data processing tasks have a relatively short request-response microtask format, the participant described something closer to a continuous query [29] in streaming database systems. In addition to suggesting an interesting direction for crowd-powered databases research, it also highlights an endpoint on the diverse spectrum of crowd worker-requester relationships. Typical relationships on Mechanical Turk are fleeting in that requesters don't establish longer-lasting relationship with workers, and especially not over the course of a single task. Here, the relationship's long-term nature allows a requester to trust workers to work on a prepaid basis and maintain complete and accurate datasets over time.

For example, the participant might prepay a worker around \$20 to monitor a set of companies and alert the participant when any

¹e.g., Data Tamr: <http://www.tamr.com/>

key employees change. For every update, the worker can subtract \$1 from their running tab, and as the balance becomes low, the worker can request another allocation.

The prepaid crowdsourcing example underscores an interesting use of crowds for push-based continuous queries. In addition to the use case above, another participant also reported sending out a survey to crowd workers periodically as a “pulse check” to verify the strength of various signals (e.g., determining how relevant a topic is to current events). While these pull-based queries don’t change, the crowd’s responses to them changes over time.

Translation and Surveys. While the previous examples focused on interesting query types, two other interesting uses of crowds focused on properties of specific crowd workers. One participant reported doing crowd-powered language translation, which requires an understanding of the languages a given worker is capable of translating. In another use-case, two participants reported sending surveys out to the crowd to collect subjective ratings and opinions. Whereas the most common use cases of crowd work seek some universally recognized truthful response across multiple workers, eliciting subjective opinions from a crowd presents new and interesting challenges. There has been some initial work from the PL research community on making it easy to design surveys [178].

Design Iteration and Programming. Crowd work tends to be presented as an inexpensive mechanism to perform repeatable piecemeal knowledge work without much skill, and this was representative of the majority of use cases participants reported. Two use cases pushed against this caricature, however. One participant tested design iterations of new user experiences and interfaces on their crowd, suggesting a more creative and subjective avenue for getting feedback at scale, while also requiring the crowd to be reasonably design-minded. Another participant climbed up the knowledge work ladder from their typical categorization tasks to also hire programmers from the Upwork platform. While hiring freelance programmers existed before the advent of crowdsourcing, this use case toed the line: the participant asked multiple programmers to solve the same problem, keeping the solution they judged to be the best amongst several. This use case lies at the junction of traditional freelancing, where a single knowledge worker delivers some expert contribu-

tion, and traditional crowd work, in which multiple workers are often asked to provide solutions to a problem.

7.1.2 Classification and categorization

Almost all of the organizations we spoke with performed some kind of crowd-powered classification (e.g., categorizing a business type or identifying spam content), albeit for a very broad set of purposes.

The first use case that we asked the participants about was classification and categorization. Classification and categorization refer to verifying if each item belongs to one out of a set of classes. Thus, based on Chapter 3, classification would correspond to filtering (Section 3.3.2), while categorization could correspond to rating (Section 3.3.2) — if it involves a few classes, or could correspond to categorization (Section 3.3.4) — if the item is to be categorized into a taxonomy. Note that we separate out spam detection and content moderation out into a separate use case even though both of those could be labeled as classification; this is because we expected spam detection and content moderation to require special attention from the participants due to the sensitive nature of the material. We describe our findings on spam detection and content moderation subsequently.

Almost all (12 of the 13) of the participant organizations we spoke with performed some kind of crowd-powered classification (e.g., categorizing a business type), albeit for a very broad set of purposes. We list them next:

- Inferring some new attribute of a user (e.g., their gender or job) from a social media profile,
- Business vertical classification (e.g., is this business a dry cleaner or a restaurant?),
- Sentiment analysis on unstructured product reviews (e.g., positive vs. negative reviews),
- Identifying the genre of a song,

- Structured data extraction from unstructured text (e.g., identifying the price of a menu item), and
- Web-page or product categorization into one of around 3000 categories.

In exploring the use cases, we came upon two dimensions by which we could break down these tasks:

Training vs. verification. A large number of the participants we spoke with utilized crowd-powered classification to train or retrain a classifier. One large organization estimated that 75% of their crowd-powered classification tasks are for generating training data that will power machine learning algorithms. A smaller number of organizations used crowd-powered classification to vet a decision made by an algorithm or another human being, for example, reviewing whether an algorithm properly identified the type of a business.

Simple, limited-class classification vs. media understanding. Most organizations used crowdsourcing to classify text, with a small number of classes (e.g., business type, or sentiment class). Text is easy to generate features for, since the features naturally correspond to the words in the document, and are easy to automatically identify. Further, training a classifier is also easier in this case because the number of classes is bounded.

Two of the larger organizations used crowds for more challenging media understanding tasks, including image, video, audio clip or unstructured text, where the output domain is large. Such tasks included image classification, machine reading comprehension, speech data understanding, or caption generation. Based on our findings, such tasks seem to be limited to large organizations that have the ability and resources to tackle these more challenging problems. As the cost of computation and storage goes down, and as these problems become more tractable via the development of machine learning algorithms that can partially address them, we expect that more organizations will turn to media understanding and large domain classification problems.

7.1.3 Spam detection/content moderation

While essentially a form of classification, we kept spam detection and content moderation in its own category of use case for several reasons. First, given the sensitive nature of the content, it is notable that organizations trust

crowd workers with it, and in fact, some participants left this work to internal contractors. Second, making these nuanced moderation decisions requires detailed instructions and examples, suggesting a level of sophistication that might surpass more general classification decisions. For instance, while it may be alright if we mistakenly label a tweet as negative instead of positive, it may be catastrophic if we label a not-spam email as spam.

Five of the 13 participants utilize crowds for spam detection and content moderation. The moderated content included web spam, email spam, comment spam, adult content, offensive content, illicit search engine optimization schemes, and copyright violations. While most of the ones in this list are not surprising to us, we were not aware that companies used crowds for the last two in the above list.

The five participants that employed crowds for such content moderation were some of the larger organizations that we had surveyed. Their size may indicate that moderation is more crucial when dealing with scale that makes it necessary to moderate content. The fact that these participants made use of crowd workers implies that making such decisions requires more nuance than fully automated algorithms might be able to provide.

Furthermore, since the nature of undesirable content or spam may change or evolve over time, these companies view it as a “moving target,” for which training data needs to be constantly obtained. This is very much unlike the traditional wisdom of getting some training data, building a good machine learning model, and letting it operate without any changes.

Since the nature of undesirable content or spam may change or evolve over time, these companies view it as a “moving target,” for which training data needs to be constantly obtained.

7.1.4 Entity resolution or matching

We now move onto entity resolution or matching. As we described in Section 3.3.3 of Chapter 3, entity resolution refers to the task of detecting which pairs of entities are identical. This could be used as a precursor to a step where the duplicate entities are removed. As it turns out, six of the 13 participants used crowdsourcing for entity resolution. Some common use cases were:

- *Pairwise resolution.* Given business listings from two different sources (e.g., “Joe’s Pizza on 123 Cambridge St. in Cambridge MA” and “Joe’s Restaurant on Cambridge St. in Cambridge”) and determining whether they described the same business. The same task is also applied to people, places, celebrities, entities, and movies.
- *Matching mentions to entities.* Matching detected entities in text to known entities (e.g., given a news article, and an underlined individual, match this individual to a social networking profile).

These tasks sometimes stand alone, but are often utilized to train or tune entity resolution algorithms.

7.1.5 Ranking and relevance

We next consider ranking or relevance; this corresponds to either rating, as described in Section 3.3.2, if the eventual output is a rating of each item independent of other items, or corresponds to ranking, as described in Section 3.3.1, if the eventual output is a ranking of all items.

Ranking was the most popular data processing use case that we had not explicitly called out in our survey.

Ranking was the most popular data processing use case that we had not explicitly called out in our survey—this was an omission on our part. Around five use cases arose in participants’ self-report. Ranking was generally used at larger organizations that wanted to tune relevance algorithms with micro-task feedback from crowd workers. Image/video-based multimedia ranking was one application area, and search relevance in areas such as web or domain name search also arose as a popular application. Groups responsible for products such as social news feeds also explained that they utilized paid crowd work to identify high-quality articles to surface to users.

The specific mechanism or crowd worker input interface was not always discussed. However, we saw more evidence of rating-based interfaces, where a workers rate a single item on a pre-set scale (e.g., 1=low relevance, 7=high relevance), than of comparison-based ones, where workers might be asked

to rank several exemplar media elements (e.g., articles, web pages) against one-another. The design of interfaces for eliciting rankings from a crowd has witnessed some attention from academia [132]; from our survey, we found that participants tended to pick simpler rating-based interfaces that have been shown to have lower cost and accuracy over more expensive and possibly more accurate comparison-based interfaces.

7.1.6 Data cleaning, verification, and normalization

In the work on crowd-powered database systems (Chapter 4), one of the motivating applications was to clean or normalize data already present in the database or gathered from the crowds. We therefore considered asking our participants if they used data cleaning and normalization. Five participants send data cleaning and normalization tasks to crowd workers. These tasks included:

- Canonicalizing and normalizing data to a style guide (e.g., changing capitalization or abbreviations),
- Finding authoritative sources to vet extracted facts from unstructured data,
- Verifying that a business has provided valid information about itself by looking up its presence online and verifying its existence through phone calls or postcards,
- Verifying previously extracted facts to ensure accuracy, and
- Identifying new facts about existing entities (e.g., what was Abraham Lincoln’s date of birth?).

Data cleaning and normalization, along with data extraction, appeared to comprise some of the most fundamental and business-critical applications of crowdsourcing for the companies that utilized it.

While not the most popular tasks across the participants, data cleaning and normalization, along with data extraction (next), appear to comprise some

of the most fundamental and business-critical applications of crowdsourcing for companies that utilize it.

7.1.7 Data extraction

Another motivating application for crowd-powered database systems is data extraction; i.e., extracting and providing information from other sources.

Five of 13 participants reported crowd-powered data extraction tasks. While this use case has heavy overlap with many of the previous use cases, we call it out because in two of the five organizations, data extraction was their primary or only application area. Some of the reported uses of crowd-powered data extraction included:

- Digitizing paper forms,
- Extracting facts about various entities (e.g., people, businesses) from webpages,
- Lead generation by identifying key contacts at various companies (e.g., identifying the CEO of IBM), and
- Extracting structured data from images, PDFs, Word documents, HTML documents, and Flash animations.

7.1.8 Text generation

Five of the 13 participants perform some form of text generation with crowd workers. Text generation applications are somewhat more subjective and creative than other use cases, and included:

- Researching a company and writing a blurb about it,
- Writing product descriptions for new products,
- Summarizing news articles, and
- Rewriting content so that it achieves better search engine optimization.

| Prior solution | # Participants | Notes |
|---------------------------------------|----------------|--|
| Didn't solve the problem | 4 | One participant claimed "Most of these problems we didn't solve at all" while enumerating tasks such as lead generation, data extraction, classification, and text generation. |
| Manual labeling by in-house employees | 4 | In some cases, developers and researchers would label enough data to train an algorithm just well enough |
| External contractors | 4 | Example: one organization, whose primary task involves high-scale data entry, previously solved a now-crowdsourced task by contracting the work out to typists. |
| Too young to know otherwise | 2 | A participant explained, "Fortunately, we always had crowdsourcing as long as we have been running our company." |

Table 7.2: Various ways in which organizations solved problems prior to turning to crowdsourcing. Some organizations reported multiple prior solutions.

7.2 Prior approaches

To help gauge the effect of crowd-powered data processing systems on organizations, we prompted participants with a free-response question "How did you solve these problems before crowdsourcing?" In their responses, a spectrum of solutions emerged, which we now summarize. In Table 7.2, we aggregate the number of responses that fell into each category. Note that some organizations did not respond to this question, while others solved different problems via different means, and thus their answers reported more than once.

Too young to know otherwise. Two companies we spoke with were started after the growth in popularity of crowdsourcing. One participant explained: "Fortunately, we always had crowdsourcing as long as we have been running our company. I can't imagine any affordable way to have done many of these things without crowdsourcing. Attempting to solve these problems would have simply been out of bounds." As we see below, more established companies arrived at solutions before the advent of paid crowdsourcing, but the sentiment of this response underscore the ease with which crowdsourcing helps in data processing tasks, and how much of an essential ingredient it is to modern-day data processing.

Didn't solve the problem, sometimes working around it. Four organizations walked the line between not having solved the problems and solving

related problems in automated ways. One participant claimed “Most of these problems we didn’t solve at all,” while enumerating tasks such as lead generation, data extraction, classification, and text generation. Other participants, for classification tasks, made do without human-provided labels for certain applications, one of them opting for “automated mechanisms with less scale and quality.” The response of this group of participants, along with that of the “too young” category, underscore the variety of tasks that were previously difficult or impossible to perform at the price points and low fixed costs that crowdsourcing enabled.

Manual labeling by in-house employees. Four of the organizations, who were amongst the larger ones we interviewed, solved certain human-in-the-loop problems with the help of full-time in-house employees. In some cases, developers and researchers would label enough data to train an algorithm just well enough. In other cases where larger scales was required, organizations formed teams of full-time in-house employees whose job was to provide human input for processing large datasets.

External contractors. Four organizations received human input from contract workers. One organization, whose primary task involves high-scale data entry, previously solved a now-crowdsourced task by contracting the work out to typists. The other organizations sought different groups of contractors to solve their varied tasks.

Note that the last two categories of prior solutions start to approach the in-house platforms, currently adopted by many of the large organizations. Thus, even before the advent of internet-enabled crowdsourcing marketplaces as we know it, many of these companies were already crowdsourcing via external contractors.

We found two anecdotes to be particularly telling of the dynamics of the spectrum above. The first comes from an organization that predates crowd work. Having solved their problem with contract workers, each task design required multi-page guidelines and a multi-layer reviewer hierarchy. As they learned which workers were most trustworthy, the organization gave those workers access to the most challenging problems, often with less guidelines. Given how close this workflow is to a crowdsourced one, the organization found benefit in the more flexible recruiting capabilities that crowdsourcing allowed as crowd work platforms availed themselves.

The second anecdote highlights the benefit of in-person contract workers over crowd workers in certain scenarios. One participant was incorporated after the birth of several crowdsourcing marketplaces, and almost exclusively performs its human-in-the-loop data processing tasks with crowd workers. Still, when prototyping particularly challenging interfaces and tasks, the organization brings in contract workers that engage with developers on-site for months at a time, reducing the communication that remote and often distant crowd work opportunities introduce. The on-site contractors help the organization more quickly iterate on guidelines, improve interfaces and catch bugs than the crowdsourced marketplaces allow.

7.3 Benefits of crowdsourcing

In a second free-response question, we asked participants to sell us on why one should use crowdsourcing at all, with a prompt of “In your words, what are the benefits of crowdsourcing? How do you sell it within your organization?” We now describe the common responses, which are summarized in Table 7.3.

“It’s the only way to do a bunch of things at an affordable cost. So it’s often a question of whether we want to solve certain problems at all. Once we opt to solve those problems, crowdsourcing isn’t optional.”

Flexibility to scale up/down. Seven organizations highlighted the flexibility of workforce size as a key benefit of crowd work. One participant explained that “we can ramp up needs ad-hoc without having to plan ahead and commit to specific uses.”

Price. Seven participants mentioned price as a consideration for crowdsourced labor. Because marketplaces allow for contracts with crowd workers all over the world, it is easier to find workers at price points that make various tasks feasible. As one participant explained, “It’s the only way to do a bunch of things at an affordable cost. So it’s often a question of whether we want to solve certain problems at all. Once we opt to solve those problems, crowdsourcing isn’t optional.”

| Benefit | # Participants | Notes |
|---|----------------|---|
| Flexibility to scale up/down | 7 | “we can ramp up needs ad-hoc without having to plan ahead and commit to specific uses.” |
| Price | 7 | “It’s the only way to do a bunch of things at an affordable cost. So it’s often a question of whether we want to solve certain problems at all. Once we opt to solve those problems, crowdsourcing isn’t optional.” |
| Enabling previously difficult or impossible tasks | 6 | Six organizations identified tasks that they simply didn’t solve until they were able to benefit from crowdsourced human input. |
| More/varied data | 5 | “our internal full-time employees would not be willing to provide millions of labels the way our crowd workers do.” |
| Free up time | 2 | “use crowdsourcing to free up our own time...or to scale it out.” |
| Quality higher than algorithms | 2 | One organization gave examples of photo moderation and OCR, which, when solved using purely algorithmic solutions, did not provide high enough quality to utilize on their datasets. |

Table 7.3: Various benefits of crowdsourcing that participants cited. Participants could provide more than one benefit.

Enabling previously difficult or impossible tasks. In an echo of their responses to how they solved various tasks before crowdsourcing, six organizations identified tasks that they simply didn’t solve until they were able to benefit from crowdsourced input.

More/varied data. Five participants identified the number of readily available workers on some platforms as a benefit when diversity of opinion or persistence at a tedious task mattered, with one participant explaining that “our internal full-time employees would not be willing to provide millions of labels the way our crowd workers do.” Another high-volume participant viewed this benefit through the lens of the high throughput of responses their application required. One organization that provided survey-like tasks to workers mentioned that although their organization is quite large, certain tasks require a level of expertise (e.g., surgeons or cardiologists) that the company does not always have ready access to.

Free up time. Two participants, both of whom are members of small teams within larger organizations, identified crowdsourcing as saving them time as a force multiplier. One participant explained that given that their teams couldn’t

grow in size by traditional means, they “use crowdsourcing to free up our own time...or to scale it out.”

Quality higher than algorithms. Two organizations identified crowd responses as being of higher quality than some algorithms. One organization gave examples of photo moderation and OCR, which, when solved using purely algorithmic solutions, did not provide high enough quality to utilize on their datasets.

We close this section with two competing and telling quotes. In the first, a participant explains what they see as the value of crowd work, highlighting many of the areas above: “Conceptually the big benefit is flexibility. We can ramp up needs ad-hoc without having to plan ahead and commit to specific uses. Practically, the benefit is training data and evaluation. Crowdsourcing helps us get new products/features out the door faster, both by giving us more/different data to learn from and by helping us [figure out] where we stand in terms of accuracy.”

The second quote offers a word of caution. While crowdsourcing affords organizations fast turnarounds on a variety of data processing problems, it does not come without an investment: “Don’t underestimate the difficulty [of] asking questions and iterative design.”

8

Survey of Industry Users: Task Quality, Worker Incentives, and Workflow Decomposition

In this chapter, we describe how crowdsourcing workflows are developed, designed, and tested. In particular, we consider quality management in Section 8.1, incentivization of workers in Section 8.2, and task decomposition in Section 8.3.

8.1 Ensuring Quality

Crowd workers could provide low-quality or incorrect responses for many reasons: poorly-written or unclear instructions, lack of necessary skills, the inherent hardness or subjectivity of tasks, malice, laziness, or fatigue.

With quality control measures, typically reliant on some form of redundancy (i.e., having multiple workers attempt the same task), crowd-powered workflow developers can ensure that the eventual results are of high quality. Along the way, many of these quality control or management schemes also identify skill-sets or accuracies of workers.

In this section, we study participants' approaches to these problems.

8.1.1 Techniques for evaluating task quality

The crowdsourcing literature is rich with algorithmic approaches to determining the individual and aggregate accuracies of workers and tasks (Sec-

| Techniques | # Participants |
|---|----------------|
| Simple aggregation techniques (e.g., majority vote) | 9 |
| Crowd workers review other workers | 8 |
| Expectation Maximization (EM) | 7 |
| Maximum Likelihood | 7 |

Table 8.1: Approaches that participants reported utilizing to algorithmically evaluate a task’s result quality as multiple workers provide responses.

tion 2.3.2). We surveyed participants on the styles of quality control approaches they utilize when requesting that multiple crowd workers redundantly provide responses to a task. Participants reported using multiple techniques. We summarize these responses in Table 8.1.

The most popular family of approaches for ensuring task quality involve simple aggregation measures, such as reporting the response that received the majority vote. Nine participants reported utilizing such techniques, likely due to their implementation simplicity and efficacy at eliminating poor worker responses.

An equal number of participants (7) reported using either expectation maximization (EM)—or maximum likelihood-style techniques. While these techniques often go hand in hand, they are different. In EM-style approaches [66], we repeatedly converge on worker quality estimates and best answer estimates by iteratively fixing one and optimizing the other, until our estimates no longer improve. These approaches do not assume the worker accuracy or quality is known beforehand. Maximum likelihood-style approaches start with the assumption that worker quality values are known upfront, and infer the highest probability response per task given those values. Worker quality estimates may be obtained by having workers perform test tasks for which correct answers are known before they start attempting tasks for which answers are not known.

All of the approaches described so far involve an algorithm inferring an accurate response given several workers’ redundant responses. Eight participants reported using a different approach, in which another set of workers is provided with previous worker responses, and is asked to identify or verify a correct response. The mechanism by which the second set of workers provides a correct response varies: workers may directly correct prior workers’

work, or they might vote on the response or responses that are most accurate. While this approach is not reliant on complex algorithmic analysis, it balances that algorithmic simplicity with more task and interface design for crowd-powered verification, as well as the additional cost of compensating the second set of workers that perform verification and review. This not unlike the approaches adopted by Turkit [126] and Soylent [36].

Eight participants reported using a different approach, in which another set of workers is provided with previous worker responses, and is asked to identify or verify a correct response.

Several participants noted that while complex approaches tend to serve them well, it is also the case that complexity is not always required. If one establishes longer-term relationships with workers, trust and improved communication can often render quality estimation and management less necessary.

8.1.2 Evaluating worker output quality

As we saw in the last section, several participants found that evaluating a worker's overall work quality or accuracy is often a useful step in ensuring high aggregate task quality.

All of the participants measured and maintained estimates of worker quality, but differed on the approaches employed, and the ephemerality of the measurement. Smaller teams tended to use less of these approaches than larger ones, but all teams employed at least one, and most teams employed several. When using a platform such as CrowdFlower, worker output quality is provided automatically by the platform (this is presumably also true if a company has its own internal platform), though even in these situations, participants tended to maintain their own independent estimates of worker's output quality. We summarize common approaches to determining worker quality in Table 8.2.

The two most popular approaches to determining a worker's output quality were gold standard questions and disagreement-based schemes. Each approach was employed by all but one participant, with a different participant avoiding each technique respectively. Gold standard questions are questions

| Technique | # Participants |
|---|----------------|
| Disagreement-based schemes | 12 |
| ▷ Expectation Maximization (EM) | 6 |
| Gold standard questions | 12 |
| ▷ Quizzes/qualification | 8 |
| Review/verification by other workers | 9 |
| Task completion speed | 8 |
| External measures (e.g., education, background) | 1 |

Table 8.2: Approaches to determining a worker’s quality given their responses. Participants might report using multiple approaches.

with known answers that are distributed throughout a worker’s workload and can be used to either signal that a worker is completing tasks correctly or educate or warn the worker if they make a mistake. Disagreement-based schemes generate a worker output quality metric based on their overall agreement with other workers who answered the same question. For instance, one simple disagreement based scheme would penalize a worker with 0 every time the worker disagreed with the majority response, and would reward the worker with a +1 every time the worker agreed: the quality of the worker would simply be the sum of these scores divided by the number of questions attempted. Yet another more complicated disagreement scheme would be to use the Expectation-Maximization (EM) estimate of worker output quality as described in Section 8.1.1.

The two most popular approaches for determining a worker’s output quality were gold standard questions and disagreement-based schemes. Each approach was employed by all but one participant, with a different participant avoiding each technique respectively.

Beyond gold standard questions and disagreement-based schemes, there were other approaches that were still popular. About two thirds of participants reported using the output of other workers’ reviews or verification to estimate worker output quality. About two thirds of participants reported employing qualification tests to quiz workers prior to the start of their actual work, though some participants were concerned with the validity of the per-

formance of a worker on an explicit test. This is because workers could easily pay attention and do well on the test and then proceed to not pay attention and do poorly on actual tasks. Note that these participants also answered positively for the gold standard approach, since qualification tests are essentially gold standard questions, but used at the start of the work. About half of participants reported employing expectation-maximization schemes to simultaneously estimate worker output quality alongside aggregate task quality. Once again, these participants also answered positively for the disagreement-based schemes approach.

In addition to worrying about worker output quality, about two thirds of participants were also concerned with another worker-related metric: speed. Most crowdsourced workflows implicitly or explicitly compensate workers for the time they spend on a task, and measuring the efficiency with which a worker completes a task is important to maintaining a reasonable cost per task. It is also possible for workers complete tasks too quickly, which might signal a work quality issue.

While we explicitly asked about it, only one of the participants reported using external measures such as education or background to identify high-quality workers, opting to allow a worker's performance to speak for itself.

Several participants reported other quality estimation techniques. One largely single-case user with a complex workflow used the aggregate degree of agreement with computer vision techniques to determine how well workers performed on vision tasks. Several participants also reported manually vetting the quality of workers' output well after an initial hiring and bootstrapping phase would have completed. Finally, while we explicitly asked about it, only one of the participants reported using external measures such as education or background to identify high-quality workers, opting to allow performance (as opposed to background, ethnicity, or demographics) to speak for itself.

8.1.3 Frequency of worker performance evaluation

As crowd workers perform tasks over a period of time for a given requester, their performance on the type of task provided by the requester may change,

both for the better (e.g., gaining experience and learning along the way) or for the worse (e.g., having difficulty concentrating through monotony). We asked participants to explain how frequently they evaluate worker performance. The majority of respondents (8) re-evaluate worker quality on every answer, continually updating their view of the worker's overall performance. A considerable group (5) only periodically updates their view of a worker's performance.

The majority of respondents (8) re-evaluate worker quality on every answer . . . A considerable group (5) only periodically updates their view of a worker's performance.

As it turns out, it is not the case that continual evaluation is necessarily the better approach. Re-evaluating worker performance too frequently might result in adverse knee-jerk reactions to natural noise in a system, for example. One participant identified this balance, and explained that they adopt different strategies in different situations. In situations where overall task quality is a concern and the participant hasn't previously vetted the workers, the participant explained that they rely on Expectation Maximization (EM)-style techniques, writing that "I don't really evaluate quality of other workers (except implicitly, on the rare occasions I use an EM algorithm), as I only care about task-level quality in the cases when I don't use my custom pool." The custom pool this participant refers to is a set of workers that have previously proven themselves, and the participant goes to this crowd for quick, reliable work, without any repeated performance evaluation. The participant treats this group differently, explaining that "For my custom pool of workers, I'll do an upfront check when creating the pool, and periodically (every quarter or half-year or so, ...) check that they're still going strong."

8.1.4 Benefiting from more advanced algorithms

We asked participants how they thought they might benefit from more advanced quality algorithms (such as those in Chapter 3) to reduce their cost. Post-hoc, we realized we had asked a leading question that might have skewed responses toward claiming perceived benefit, so we do not report

them in aggregate. The question did, however, elicit several lively free responses against more advanced algorithmic approaches, which we now report.

One of the largest participants by weekly volume responded that because of their volume, evaluating worker quality would exclude too many inputs, and that simple techniques to ensure aggregate task quality was all they could use to sustain volume: “Maybe by giving worker with excellent track record higher weight, there could be some savings. However, we did not see there would be big enough savings for us to justify the effort. Perhaps a readily available algorithm will change our mind. Our operation is high volume/low cost per task. Excellent workers only work on a small portion of the total.”

One participant explained that algorithms aren’t as effective on qualitative tasks. They expanded that “Most answers are richly qualitative or based on consensus. Manual review and agreement are safeguards enough.”

Finally, a participant highlighted the benefit of vetting trusted pools of workers ahead of time: “[The] answers I get tend to be accurate enough already (even if I place only a single worker on the task).”

8.1.5 Communicating feedback

A learning opportunity arises each time a crowd worker makes an error on a task. We asked participants if and how they communicate feedback to workers. Of all participants, 8 claimed to provide any sort of feedback to workers, while 4 provided none. Generally, we found that larger teams and teams that manage the internal platform tended to have more mechanisms for providing feedback than the smaller teams.

We also found reasonable diversity in the forms of feedback participants provided to crowd workers. One participant found great value in immediate per-task feedback. Having experimented with around two dozen incentive mechanisms, they found that immediate feedback was the most important and effective one. In particular, this participant found that providing incorrect workers the correct answer immediately, and affirming correct workers immediately, gave crowd workers a sense that the participant cares about their input. The participant explained that “Even if [we] don’t know perfect answer, [we] tell the worker what the machine decided [so far based on other answers or a prior]. The worker can provide a rebuttal if they disagree. This

gives instantaneous feedback, but user can still challenge it.”

“Even if [we] don’t know perfect answer, [we] tell the worker what the machine decided [so far based on other answers or a prior]. The worker can provide a rebuttal if they disagree.”

Another participant was more concerned about higher-level feedback to develop their crowd workers’ skills. The participant expanded that “[T]his is an area where we wish we could give feedback sooner, and to classify high-level mistakes. If a reviewer repeatedly changes an entry-level worker’s tasks, we’d like to tell the worker ‘it looks like you consistently fail on, say, capitalization, and here is a document you can read to improve your work’.” Several participants echoed this desire to actively improve their feedback, specifically if they can detect that the worker is consistently making a certain type of mistake.

8.2 Incentives

An important component in building crowd-powered systems and workflows is the incentive mechanisms in place for crowd workers. Incentives can impact response speed, accuracy, and in general, the effort put into providing the answer. Our participant base is biased heavily toward users of paid crowd work, and so many of the incentives that surfaced involved some form of monetary compensation, but other interesting mechanisms surfaced as well.

8.2.1 Popular Incentive mechanisms

We asked participants to describe the incentive mechanisms they utilize with their crowd workers. We prompted the participants to describe both monetary and non-monetary incentives. While monetary compensation was by far more popular, participants also described other mechanisms. We summarize these findings in Table 8.3.

At a high level, our participants primarily rely on per-task payments along with bonuses, while some participants use hourly payments (primarily for workers who are trusted), non-monetary incentives like leaderboards and gamification, as well as promotions along a hierarchy.

| Mechanism | # Participants | Notes |
|---------------------------------------|----------------|--|
| Per-task payment | 11 | Example: pay 5 cents for completing each task. More complex per-task payment schemes exist: one participant estimates the difficulty of a task given the number of button presses, mouse movements, wait time, and mental preparation, to determine how much to compensate a worker. |
| Bonus-based payments | 10 | Crowd workers are incentivized to reach certain higher-level goals (e.g., speed or quality) across several tasks and are compensated with a bonus for their efforts. |
| Hourly payments | 6 | Example: pay \$10 per hour of work. |
| Gamification | 4 | Example: Awarding points and badges for accomplishments, like completing 1000 tasks without error. |
| Leaderboard | 4 | Example: Our top ten fastest workers this week were: ... |
| Hierarchy of statuses with promotions | 3 | Example: help manage the crowd if you do well enough. |

Table 8.3: The various incentive mechanisms used to encourage worker performance. Participants could report more than one mechanism.

Our participants primarily rely on per-task payments along with bonuses, while some participants use hourly payments (primarily for workers who are trusted), non-monetary incentives like leaderboards and gamification, as well as promotions along a hierarchy.

Given the heavy microtask bias of our participants, the most popular monetary incentive pay mechanism was a per-task payment, which 11 of our participants used. While these payments are typically the same for each task, one participant made use of keystroke-level models (KLM-GOMS [49]), a model that estimates the difficulty of a task given the number of button presses, mouse movements, wait time, and mental preparation, to determine how much to compensate a worker. The next most popular payment model (10 participants) was a bonus-based one, where crowd workers are incentivized to reach certain higher-level goals (e.g., speed or quality) across several tasks and are compensated with a bonus for their efforts.

Finally, 6 participants compensated some or all of their crowd workers on an hourly basis rather than a per-task basis. One participant explained that they start all workers off with a per-task payment, and shift only trusted workers to hourly rates. They noted that certain workers' throughput degrades when switching to hourly payments, and so this participant only finds hourly work meaningful if they have worked with that worker long enough to appre-

ciate their work quality and integrity, and if a task type arises wherein it is difficult to estimate the task requirements and difficulty up front.

Fewer participants made use of non-monetary incentive mechanisms. Four participants made use of various forms of gamification [185]. Four participants maintained some form of leaderboard to publicly recognize the highest-performing crowd workers (possibly in terms of quality or amount of work done).

Finally, while most participants claimed a flat organizational model, three utilized a hierarchical model that allowed them to use promotions to higher statuses or more interesting positions as an incentive mechanism. We find this last model particularly interesting, because it indicates a move toward more traditional employment practices.

8.2.2 Worker classes and differing incentives

We next asked participants whether they have different classes of workers, and if so, what those classes were. About half of the participants responded that they did have different classes of workers, while the other half did not. Larger organizations with more use cases for their crowd work tended to differentiate workers, while smaller organizations did not.

Larger organizations with more use cases for their crowd work tended to differentiate workers, while smaller organizations did not.

Two forms of worker classes arose: skill-based and trust-based. In the skill-based system, workers with particular expertise (e.g., the ability to speak a particular language) were placed in a worker class that could process relevant tasks requiring that expertise. In the trust-based system, manual or automated processes determined the quality of a worker's responses over time, and placed workers with desirable qualities into more trusted classes, which were compensated more for their achievement or allowed to perform more challenging or interesting tasks. One participant created a hierarchical review-based system [118, 91], in which more trusted workers would review and spot-check the work of workers who had yet to gain trust in the system.

Participants placed significant value on incentivizing trusted workers to continue to work with them. One participant explained that finding the best workers was less important than finding reasonable performers: “In my experience, the difference between good workers and unknown workers is much more important/impactful than the difference between good workers and great workers or workers with different specific skills.”

“...the difference between good workers and unknown workers is much more important/impactful than the difference between good workers and great workers or workers with different specific skills.”

When we asked participants how the incentive mechanisms for different classes of workers differed, many responded that aside from pay rate or type of task, there was not much investment in differing incentive mechanisms. Workers generally did not have much transparency into different incentive schemes. One participant explained that there was “[no] transparency, except for the fact that they can expect to receive a steady stream of work from us should they perform well.” Some participants wanted to provide more transparency into the process to workers, with one participant explaining that “workers know our schemes, but often complain that they don’t know when they will be promoted...We send them weekly emails to say how they rank, but this can’t predict when we’ll have an opening for a [more lucrative position].”

8.3 Task design

From microtasks to complex work, verification to redundancy, there are many designs to crowd-powered data processing workflows (Chapter 4). We asked participants about the types of workflows they design for their crowd workers, and summarize them below.

8.3.1 Crowd-Machine integration

One promise of crowd work is that it helps bridge the gap between the capabilities of machine learning algorithms and the abilities of crowds to process

| Integration | # Participants | Notes |
|-------------|----------------|---|
| Train | 12 | Crowd workers provide training data for automated classifiers, and at a certain point the classifiers process all subsequent tasks. |
| Vet | 11 | Crowd workers verify or evaluate the output of multiple algorithms. |
| Uncertain | 8 | Only when an algorithm is uncertain of a decision does a workflow reach out to crowd workers for a decision. |
| Standalone | 7 | Crowd workers complete all of the work in a data processing workflow. |
| Active | 5 | Algorithms identify which data to send to crowd workers for further training in an active learning loop. |

Table 8.4: The various ways in which crowd workers and machine learning algorithms are integrated. Participants could report more than one approach.

unstructured information. We wanted to understand how practitioners bridge this gap: are they relying more on crowds or more on machine learning? How are these two options used in parallel? We asked workers to identify which forms of crowd-machine integration they employ, and summarize responses in Table 8.4.

The most popular approaches were *Train* (12 participants train some automated classifiers) and *Vet* (11 participants assess the quality of their models with crowd worker input). While 8 participants only send *Uncertain* algorithmic decisions to the crowd for quality purposes, 7 use crowds in a *Standalone* fashion with no subsequent modeling or learning. Finally, 5 participants employ closed-loop *Active* learning, in which the crowd worker responses to uncertain algorithmic decisions are used to retrain future models.

One pattern to note is that, of the crowd-machine hybrid models, the popularity of the three approaches decreases as the complexity of the approach increases. *Train* is the simplest and most popular model, *Uncertain* is more complex but used by the majority of participants, and *Active* is the most complex and least popular. None of the participants we classified as largely single-case user personas employ active learning, and the approach is typically more popular with the multi-case users or internal providers. One participant explained that for one project, they transitioned through the three models in order of complexity as the maturity of the team and needs of the project grew. This suggests that if a researcher or practitioner built an accessible and reusable framework that simplified active learning, they might have an audience that could benefit from these techniques, as potential users don't have the resources to build the tools themselves.

8.3.2 Use of frameworks

In Chapter 4, we described several tools for declarative and imperative specification of crowd work, with the aim of reducing developer load, and seamless optimization [126, 144, 80, 116, 133, 22] for common crowdsourcing operations. We asked participants about their use of such tools, frameworks, libraries, and third-party APIs. We summarize these in Tables 8.5 and 8.6.

Surprisingly, no participants utilized third-party frameworks to simplify their crowd-powered data processing workflows.

Surprisingly, no participants utilized third-party frameworks to simplify their crowd-powered data processing workflows. Six participants claimed to use internally generated frameworks for simplifying their task design. Eleven reported utilizing a low-level API for accessing a service for posting, retrieving, and paying for tasks. Six of those reported implementing against the Mechanical Turk API, three against the CrowdFlower API, and two against the oDesk API.

It appears that participants only utilized APIs in as much as they provided low-level payment and task creation functionality. For the crowd-powered systems-building community, these findings can be interpreted in two ways. First, it suggests that the community might need to better communicate its contributions and deliver them as usable open source artifacts if they intend to have impact on industry users. Second, it might suggest that teaming up with industry users to learn what high-level constructs they have baked into their frameworks might result in inspiration for more principled contributions from the systems builders.

8.3.3 Task decomposition

One area of active research in crowd-powered workflow design is in task decomposition, i.e., decomposing a large job into smaller tasks [116, 91, 119, 114]. We asked participants whether their workflows feature mostly microtasks (e.g., simple yes/no questions that are asked redundantly of several crowd workers) vs. macrotasks (e.g., complex work that might take several minutes or hours to complete, and is hard to reconcile across redundant workers).

| Tool | # Participants | Marketplace integration | # Participants |
|-------------------------|----------------|-------------------------|----------------|
| Marketplace integration | 11 | Mechanical Turk | 6 |
| Internal framework | 6 | CrowdFlower | 3 |
| Third-party framework | 0 | Upwork | 2 |

Table 8.5: Tools used by participants to simplify their crowd-powered workflow development. Some participants reported utilizing multiple tools.

Table 8.6: The API integrations with various marketplace providers that participants reported implementing. Some participants utilized multiple platforms.

Of 11 respondents, five claimed to entirely utilize microtasks, one claimed to use primarily microtasks, and five claimed to use both models about equally frequently.

Of 11 respondents, five claimed to entirely utilize microtasks, one claimed to use primarily microtasks, and five claimed to use both models about equally frequently. Two participants explained that their largest spending rate on crowd work was on macrotasks. One participant proposed that, in their experience, decomposing large jobs into redundant microtasks provided better results. These responses suggest that while crowd-powered workflows that rely on workers completing complex tasks are feasible, task decomposition is a area of research that is based on real business needs.

8.3.4 Multistep workflows

As in more traditional industrial workflows, complex crowd-powered workflows sometimes require more than one step with human involvement (perhaps in different capacities). If we think about a *crowdsourcing step* as a single user interface coupled with some algorithmic computation (e.g., a crowd-powered filter operation, or an active learning loop), it is interesting to explore how often real workflows employ multiple steps. For example, the Find-Fix-Verify [36] design pattern involves three sets of crowd workers to focus on the individual finding, fixing, and verification steps.

Many participants reported that none of their workflows have more than crowdsourcing step.

We asked our participants what fraction of their workflows involve more than one crowdsourcing step. Many participants reported that none of their workflows see this. Of eleven respondents, two participants explained that half of their workflows involve more than one step, and four explained that they rarely employ more than one crowdsourcing step. One participant was particularly pessimistic about multistep workflows, explaining “In my experience, if you need multiple steps of crowdsourcing, it’s almost always more productive to go back and do a bit more automation upfront.” It is interesting to note that while industrial crowdsourcing systems are complex (they require teams and budgets of reasonable sizes), the complexity doesn’t stem from the design and maintenance of multi-step workflows.

These findings can be interpreted in two ways. On one hand, a good amount of research efforts have gone into complex multi-step workflow designs for crowdsourcing, and it is unclear that practitioners needs these tools at the moment. On the other hand, we are still in the early days of crowd-powered workflow development, and as more research bears the fruit of high-level creative and knowledge work [159], the need for multi-step and more involved workflows might become more apparent.

8.3.5 Design patterns

A common set of crowd design patterns and primitives is arising in the academic research literature. We asked participants about their use of three popular patterns as well as any other they would like to share. The three we asked about were iterative refinement [126], find-fix-verify [36], and do-verify [21, 91]. Iterative refinement allows workers to build on other workers’ output in sequence. Find-fix-verify asks three groups of workers to identify issues, propose fixes to those issues, and identify the best eventual outcome. Do-verify relies on a hierarchy of trusted workers that review, correct, and vet work that entry-level workers complete.

While only four participants claimed to use at least one of these patterns [iterative refinement, find-fix-verify, or do-verify], all three patterns appeared equally popular.

While only four participants claimed to use at least one of these patterns, all three patterns appeared equally popular. Another participant explained that they combine several stages from a collection of the three workflows above, but did not provide additional detail. Like our findings on complex workflows, it is unclear whether the low adoption of these patterns is a signal that these crowd-oriented design patterns are not terribly useful in practice, or whether they will see more use as the kind of crowd work that practitioners explore necessitates these more complex patterns.

8.3.6 Design iterations

In as much as building crowd-powered workflows is an engineering task, it is also a design task. We asked participants how much iteration went into task design, and received varied answers. Almost all participants actively iterate on their designs. The two that claimed to not do design iterations are some of the largest single-workflow users, and as a result of early iteration, have reached a level of output quality that they are pleased with and aren't incentivized to change.

The most common response we received was around small, incremental updates to designs. One larger participant explained that “[we] do build some initial prototypes and then get initial feedback. If the question is vague or guidelines are not enough or if the question is too difficult in some cases then we iterate. It helps a lot, and our workers like that.” Another participant echoed a similar viewpoint, that “Most teams don't get everything right the first time, but generally only need to make incremental changes to make further improvements.” Most participants actively reached out to workers for feedback, which helped drive their future task design and development.

At the other end of the spectrum, there were participants that spent a lot of time on task redesign. Only one participant (an internal crowdsourcing provider) explicitly called out A/B testing, or randomized testing in which different workers are presented with the same task in different ways to see which design is more effective. This participant also referred to task design as an art, and displayed a hands-on mentality toward design.

8.3.7 Crowd worker communication, interaction, and collaboration

Entire communities of researchers, like the Computer-Supported Cooperative Work [7] community, have been formed around human-human interaction techniques in traditional work environments. When we asked participants about their use of such tools and techniques to facilitate interaction between crowd workers, few reported back any such infrastructure.

One participant set up mailing lists and chatrooms for their crowd workers, and engaged on message boards such as TurkerNation [16] and Cloud Me Baby [4]. Another participant, in addition to email and chatrooms, allowed entry-level workers and reviewers to leave notes for one-another on individual tasks to facilitate learning experiences. Three other participants reported some form of interaction tools, but did not specify details. Aside from facilitating interaction, no participant designed tasks with active crowd worker collaboration in mind.

It is clear from the existence of message boards such as TurkerNation that crowd workers benefit from advice, training, and knowledge-sharing, but it is also clear that support for such interaction has not been built in by many practitioners. Designing tools and frameworks to facilitate such interaction might result in fruitful research going forward.

9

Survey of Marketplace Providers of Crowdsourcing

In the previous chapter, we described the results of surveys and interviews we conducted with industry users of crowd work. We now turn to another group: the providers of public marketplaces where crowd workers and these industry users can find one-another. We used a similar methodology as described in Section 5.2 to identify and interview four crowdsourcing marketplace providers. The questions we asked the providers can be found in Appendix B, with a summary of the overall question types in Table 9.1.

Whereas the industry user survey allowed us to speak with people at various levels throughout their respective organizations, in our marketplace survey we spoke exclusively with company leadership. At the four marketplaces, we received responses from two CEOs, one Senior Director, and one VP of Product. This bias toward leadership had the benefit that we could get good high level descriptions from each participant, with the drawback that some of the descriptions of implementation details were not as deep. Where possible, we reconstructed such details by exploring each platform ourselves.

We will first describe the four marketplaces we interviewed, summarize some high-level observations, and then compare and contrast the marketplaces in detail across the various questions we asked. We cover statistics about marketplaces in Section 9.2, implementation details: task creation, task

| Section | # Questions | Paraphrased Example Questions |
|-----------------------|-------------|---|
| Crowd characteristics | 6 | — What is the approximate age/gender/geographic/education distribution of your crowd? — What fraction of your crowd uses your platform as their primary job? |
| Implementations | 9 | — What fraction of the tasks in your marketplace are microtasks versus macrotasks? — How do you match workers to tasks? (topics include requesters picking workers, workers picking tasks, automatically match workers to tasks, etc.) |
| Statistics | 6 | — What is the distribution of spending per requester? — Do you ever observe an imbalance between the number of workers and number of tasks at any point? |
| Quality assurance | 4 | — What mechanisms do you use to infer worker trustworthiness? (topics include worker education level, engagement on platform, accuracy, etc.) |

Table 9.1: A summary of the types of questions we asked participants either through a survey they filled out on their own time or through phone interviews. The example questions provided are paraphrased descriptions. Detailed questions can be found in Appendix B.

types, and matching in Section 9.3, and then cover quality control in Section 9.4.

9.1 Executive Summary

Marketplaces vary widely in their mission, their target requesters, their target workers, their demographics, and the type and format of the work available on each platform. We’ve picked four representative ones that operate relatively differently from one another, offering requesters varying choices of task design, quality assurance mechanisms, and degree of self-service in the process.

We now provide high-level takeaways from our interactions with participants. We provide more details in subsequent sections. The four marketplace providers we interviewed are summarized in Table 9.2.

The ecosystem is quite fluid. In the time that we conducted interviews with participants, one company (MobileWorks) changed its mission from being a marketplace provider to providing a particular set of services (lead generation); two companies that were rivals (Elance and oDesk) merged into a single company (first called Elance-oDesk, and rebranded to Upwork).

Marketplace providers comprise half of the space. In the industry surveys, we found that industry users have a roughly 50%-50% split between utilizing

| Marketplace | Total Crowd Workers | Type of Marketplace | Notes |
|-------------|-----------------------|------------------------------|---|
| CrowdFlower | Millions | Microtasking | Requesters can design and post microtasks, and tune parameters for the platform to manage a quality-cost tradeoff through trusted workers and redundancy. |
| Upwork | Millions | Macrotasking via Freelancers | A market on which requesters and contractors can find one-another with a focus on payment and reputation rather than the mechanics of task delivery. |
| MobileWorks | Hundreds to thousands | Macrotasking | Requesters create high-level tasks without microtask decomposition while paying workers fair hourly wages. |
| Samasource | Thousands | Microtasking | A mission-focused nonprofit that provides requesters with projects management and consultation on task design while connecting them with workers from marginalized populations of mostly women and young individuals. |

Table 9.2: The marketplaces that participated in our survey, along with their approximate size in crowd workers, and notes on each marketplace’s key points.

marketplaces for crowd work and hosting their own internal platforms for crowd work, some of which are staffed through tightly controlled outsourcing firms (Section 6.5). Our conversations with marketplace providers can thus only describe options utilized by about half of our industry users.

The requester experience varies wildly by marketplace. On each marketplace, the expectations of which aspects of workflow management requesters handle and which aspects the marketplace is responsible for are quite different. For example, whereas Upwork offers facilities for discovering, establishing, and paying for contracts (an agreement between a requester and worker), CrowdFlower abstracts these concepts away and automates a lot of the requester-worker relationship.

Worker demographics are quite different. On CrowdFlower and MobileWorks, the United States is listed as the top country of residence, whereas on Upwork (according to oDesk), the top two countries are India and Pakistan and on Samasource, the top countries are India, Kenya, and Uganda. Gender distributions differ by platform, with Samasource explicitly biasing toward women in certain countries, CrowdFlower reporting 26% women in

aggregate, and MobileWorks reporting 50% women. Finally, the educational distribution is different, with CrowdFlower reporting 36% of workers without a college degree, Upwork claiming workers are typically “College graduates and above,” MobileWorks explaining that all workers have or are pursuing college degrees, and Samasource explicitly focusing on populations with no college degree or previous job.

Marketplaces have a worker-work imbalance. Every marketplace except Upwork said that they encountered situations with “the number of available workers being greater than the number of available tasks, and hence workers have to wait.” CrowdFlower added that while this statement was true for its entry-level workforce, its experienced tier of workers saw more demand than they could fulfill.

The marketplaces differ significantly in throughput. Given their different missions and requester experiences, there are several order-of-magnitude differences in the number of workers or tasks completed per day on each platform. Given that some platforms (e.g., Samasource) focus on economic and social opportunity, others (e.g., CrowdFlower) focus on self-serve user interfaces for task management, and others yet (e.g., Upwork, MobileWorks) focus on complex tasks, it is not meaningful to compare the systems based on raw numbers, and it is also unreasonable to recommend a “best” platform.

Quality control approaches vary significantly. Given the differing work formats that each platform offers (Section 9.3.1), and the differing availability of training staff co-located with workers, the approaches to quality control on each platform differ (Section 9.4). While CrowdFlower offers the most automated quality control measures, Upwork provides publicly available post-task feedback and ratings, MobileWorks managers offer review of workers’ tasks, and Samasource offers in-person training based on spot-check reviews.

9.2 Marketplace Details

We now dive into each of the four marketplaces, providing details on aspects like company mission, implementation of platform, demographics, and size. An astute reader might note that Amazon’s Mechanical Turk is not one of the marketplaces we include in this survey. We reached out to Amazon to participate, but did not hear back.

9.2.1 CrowdFlower

CrowdFlower was one of the first crowdsourcing marketplaces. CrowdFlower focuses primarily on providing industry users like data scientists access to a large group of crowd workers that complete microtasks. It provides APIs and user interfaces for requesters to design and populate tasks for crowd workers to complete. Requesters generally leave quality control to CrowdFlower after providing gold standard examples for training and vetting workers, and by setting quality and cost thresholds within which CrowdFlower automates task redundancy. Requesters can either provide their own set of workers and utilize CrowdFlower’s interface and quality control mechanisms, or source workers from approximately 130 CrowdFlower channel partners¹ that compensate workers with items ranging from regular currency (e.g., ClixSense [3]), cryptocurrencies (e.g., BitcoinReserve [1]), and reward points that can be redeemed at various locations (e.g, Swagbucks [15]).

CrowdFlower is relatively open with its crowd composition, publishing results of surveys with its crowd [207]. Nearly a third of its crowd workers, called “contributors,” live in the USA (18% of contributors) and India (12%), with other popular locations including the UK (6%) and Indonesia, Canada, the Philippines, and Pakistan (4% each). The largest group of contributors report having a bachelor’s degree (25%), with 20% reporting a high school diploma/GED, 16% receiving some college with no degree, 11% with a Master’s degree, and, at the smallest end of the spectrum 2% reporting a Doctorate. 39% of contributors claim to be males between the ages of 24-34, and overall, 72% report male and 26% report female as their gender. While 4% of the workforce reported being younger than 18 at the time of CrowdFlower’s survey, 50% reported being younger than 30, and 3% reported being older than 64. 46% report a household income less than \$10,000, and 3% report a household income greater than \$150,000. 50% of workers report working online because “it is a great way to spend free time and get some cash.”

In the month we interviewed CrowdFlower, their daily active crowd workers ranged from 6,500 to 17,000. Until that point, CrowdFlower had seen 5 million contributors log in and participate in the platform in some way or another. Contributors are organized into levels, with a large pool of level

¹We determined this by logging into CrowdFlower and looking at a configurable “Channel Table” in a task’s configuration

1 crowd workers for requesters that need tasks completed quickly. As crowd workers prove themselves on the platform, they can be promoted into trusted levels 2 and 3, which provide more lucrative or interesting tasks, and CrowdFlower reports around 10,000 active contributors are available at this level.

Until the point when we interviewed them, CrowdFlower had seen 5 million contributors log in and participate in the platform.

9.2.2 Upwork

Upwork² is a marketplace where requesters looking for subject matter experts can hire crowd workers as “freelancers,” and have them work on a “contract.” This model is more similar to the “freelancing” firms of the early 2000’s rather than the Mechanical Turk and CrowdFlower-like microtasking marketplaces of the late 2000s. Elance and oDesk merged to form Upwork, and Elance was established in 1999 while oDesk was established in 2003. In this model, a team can supplement their existing skillset with a freelancer, or a project manager can source experts in many fields, leaving Upwork to handle payments, time tracking, and identifying good leads.

At the end of 2014, Upwork report [17] a combined freelancer base of greater than 8,000,000 in more than 180 countries, with “\$750M worth of work done in 2013.” While the oDesk website lists a little more than 80 categories of freelancers available, the most promoted categories on its front page are web design (34,711 freelancers), copy editing (21,132), web development (62,984), search engine optimization (20,947), mobile application development (21,088), data entry (76,216), virtual personal assistance (27,119), and software development (56,849). Based on these categories, it is easy to see that this freelancer or worker pool is focused more on the “higher end” tasks that require significant expertise. In searching for a freelancer, a requester can see their desired hourly wage, their hours worked on the platform, a portfolio of previous projects, and reviews from previous contracts.

²Elance and oDesk merged in late 2013 and eventually rebranded to Upwork, and while we interviewed oDesk employees, the two marketplaces are very similar in nature. When reporting aggregate statistics, we include counts from both markets when possible.

Through the survey, Upwork reported that their workforce was between the ages of 18 and 50, but did not provide more granularity. The educational achievements of the freelancers was described as “College grads and above.” The top 10 freelancer countries reported are India, Philippines, USA, Ukraine, Pakistan, Russia, Bangladesh, China, Canada, UK.

9.2.3 MobileWorks

MobileWorks was a startup that focused³ on providing requesters with solutions to problems that are larger than microtasks. While requesters still had programmatic access to MobileWorks, they could request higher-level tasks than microtasks (e.g., “create a presentation based on this outline”), and implemented a review-based hierarchy where trusted managers vetted crowd worker contributions. Part of the mission of the company is to provide fair wages to workers in both the developed and developing world, and this is reflected in both the company branding, charges, and high-level tasks they describe.

While the company did not provide detailed statistics, its crowd workforce’s age ranged between 18 and 60. 50% of the crowd claims to be female, whereas 47% responded male. All of the MobileWorks crowd workers have a college degree or are in the process of completing one. The primary countries in which the company has crowd workers are the US, India, Philippines, Kenya, Serbia, and Jamaica. Many workers report using the MobileWorks platform as their primary job, but the company does not have precise numbers on this.

9.2.4 Samasource

Samasource is a non-profit social enterprise with a vision “to connect the one billion people living in poverty around the world to work using the power of technology” [13]. To work toward this vision, it connects corporate customers in need of microtask labor with agencies in the developing world that source crowd workers. Agencies provide resources such as training, a location to do work, and computers on which to do the work. Some training and feedback

³in 2014, the company became LeadGenius, which provides lead generation services with the help of the crowd.

is supported by Samasource's employees and systems, but the primary role of ensuring that workers complete tasks well is in the hands of trainers and managers at individual agencies.

As part of Samasource's mission involves pairing marginalized populations of women and young individuals that are unemployed or underemployed, most of the Samasource workers are between the ages of 18 and 30, with a few between the ages of 30 and 40. Samasource has partner agencies supervising crowdsourcing in Haiti, Ghana, Uganda, Kenya, and India, with the largest concentrations in India, Kenya, and Uganda. These agencies offer *affiliate delivery centers*, which are physical locations that workers go to that offer all of the required facilities (e.g., managers, trainers, computers, stable electricity) to do crowd work. The gender distribution varies by geography: in India, Samasource works with a close partner that only works with women, whereas in Africa Samasource works with relief centers that also source many men.

As per Samasource's standards, crowd workers must be English speakers, but other criteria aim to assure that the workers are in some way "in need." A lot of the workers have never had a job before and do not have a college degree, and while training programs vary by geography, a training period of approximately 2 weeks includes technology and workplace etiquette training. While we don't have statistics on this, Samasource assumes most workers focus on tasks it provides as their primary employment.

A recent impact report [12] explains that Samasource has interacted with 6277 workers by late 2014, and with 878 active workers in the second quarter of 2014. Samasource builds worker churn into its goals: with 92% of workers reported to be underemployed or unemployed before working with Samasource, 89% pursue additional employment or education after Samasource. As a result, one can not compare the active or aggregate number of workers interacting with Samasource to the other marketplace providers.

9.3 Implementations

As the providers each serve different missions, the implementation details of each marketplace are quite different. We now compare and contrast each marketplace's design decisions and discuss the implications for requesters

| Marketplace | Task Complexity | Notes |
|-------------|-----------------|---|
| CrowdFlower | Microtasks | Requesters submit tasks programmatically or via spreadsheet upload, with the platform sending microtasks to multiple workers |
| Upwork | Macrotasks | Requesters manage relationships and contract creation for themselves |
| MobileWorks | Macrotasks | The platform accepts a wide variety of tasks described at a high level that managers vet and ensure quality on |
| Samasource | Microtasks | Requesters submit tasks programmatically or via spreadsheet upload, with Samasource employees helping manage and design the process |

Table 9.3: The degree of complexity (microtask vs macrotask) of tasks on each platform.

building workflows on each platform.

We describe the complexity of tasks in Section 9.3.1, task creation in Section 9.3.2, matching workers to tasks in Section 9.3.3, and dealing with mismatches between numbers of workers and tasks in Section 9.3.4.

9.3.1 Task Complexity

The four platforms we spoke with have different approaches to the size of a task a crowd worker must complete, which we describe in Table 9.3. CrowdFlower and Samasource provide workers with *microtasks*, or well-defined simple questions (e.g., yes/no, categorization, simple free-response). This form of task is often useful for training classifiers and many other standard data processing operations that we described in Chapter 3. Upwork and MobileWorks, on the other hand, focus on *macrotask* labor. In Upwork’s case, requesters bring freelancers on to perform high-level tasks like designing websites or writing free-form text, that do not fit the simple well-defined small question template. MobileWorks, while still providing programmatic access to create tasks, also considers all tasks to be macrotasks: requesters are given relative freedom to describe a high-level task, and MobileWorks has systems in place to ensure the quality of the eventual output.

9.3.2 Self-serve vs. Managed Task Creation

We asked the platform providers a series of questions to understand how requesters could access their platforms. We summarize the results in in Table 9.4. Specifically, we wanted to know what sort of programmatic access

| Marketplace | Service Type | Notes |
|-------------|-------------------------------|---|
| CrowdFlower | Self-serve w/ managed quality | CrowdFlower reports that all users start with the platform's user interface to generate their tasks and test their workflows, but then about half utilize the API to generate tasks once they have ironed out the details. |
| Upwork | Self-serve | Because tasks and contracts are built around higher-level macrotasks, requesters on Upwork are not offered specific authoring or task management tools. |
| MobileWorks | Hybrid | Requesters are provided with programmatic access to the platform. End-to-end management of task design was made easier through the explicit role of managers on MobileWorks, who could help with both vetting crowd workers, and creating and experimenting with new workflows for completing work. |
| Samasource | Managed | Most frequently, requesters rely on Samasource to manage the end-to-end process, including task and interface design, as well as crowd worker training and vetting. |

Table 9.4: The degree to which the crowd workflow design and development process is self-serve or managed on each platform.

they allowed requesters to generate new tasks and compensate workers, and what methods requesters used to manage their workflows.

CrowdFlower provides a user interface in which requesters can design their tasks in a markup/template language that makes it simple to embed task-specific information in web forms that codify the questions a crowd worker must answer. It then allows them to upload spreadsheets with data they wish to have processed (e.g., images to be classified) as well as an API to create these tasks. CrowdFlower reports that all users start with the platform's user interface to generate their tasks and test their workflows, but then about half utilize the API to generate tasks once they have ironed out the details.

Samasource does not provide a user interface for task design, but allows requesters to either upload spreadsheets or create new tasks programmatically. Most frequently, requesters rely on them to manage the end-to-end process, including task and interface design, as well as crowd worker training and vetting. This is most similar to traditional business process outsourcing, in which the outsourcing firm performs project management and design work on behalf of the client.

MobileWorks, while providing more self-serve functionality for creating and managing tasks than Samasource, ultimately found that most requesters opted for end-to-end management of task design. This was made easier through the explicit role of managers in the platform, who could help

with both vetting crowd workers, and creating and experimenting with new workflows for completing work.

Upwork offers the richest APIs of the four platforms for updating contracts, extracting worker statistics, and managing payments. Because tasks and contracts are built around higher-level macrotasks, requesters are not offered specific authoring or task management tools. However, enterprise clients are offered an in-the-works TaskManager tool that facilitates task design, management, and reporting.

9.3.3 Matching Workers to Tasks

Given the vast work available on each marketplace, it is interesting to ask how workers are matched to work. On CrowdFlower, MobileWorks, and Samasource, workers decide which tasks to work on. Thus, these marketplaces are “worker-centric” in that (assuming they meet the qualifications), the worker ultimately decides which tasks they would like to work on. This is also true of Mechanical Turk, for example. These platforms offer requesters some filtering ability, like filtering on worker qualifications (e.g., whether a worker has experience in labeling images). CrowdFlower and Samasource offer more generic filters, such as filtering on worker’s prior quality ratings or their country of origin. Samasource also allows requesters to create whitelists of workers with which they have had good experiences, and blacklists of workers to avoid working with in the future.

On CrowdFlower, MobileWorks and Samasource, workers ultimately decide which tasks to work on... on Upwork, requesters ultimately decide which workers to hire.

Upwork, because of the more contractual nature of the work, acts as a matchmaker at the contract level. Requesters, whose profiles and ratings are available, can post tasks and receive offers from freelancers whose profiles, previously completed contracts, and ratings are available. It is ultimately up to requesters which freelancers to start contracts with. Thus, Upwork is “requester-centric” in that requesters make the ultimate decision (even though workers can indicate their eagerness to work on tasks).

While the matchmaking process differs by platform, all platforms indicated a desire for better algorithmic matching tools, whether they improve search results or make recommendations for requesters and workers alike. This matching problem may prove to be fruitful ground for future research.

9.3.4 Work-Worker Mismatches

CrowdFlower explained that in their largest pool of less vetted workers, there are always more workers available than tasks to complete. In their more curated pools of workers that had proven themselves on the platform, the amount of work was greater than the available crowd worker supply.

Marketplaces have a unique view of both the demand and the supply of labor. In some cases, they can segment this information further by specific types of tasks to identify holes in either supply or demand for varied areas of expertise. We asked participants about their understanding of this mismatch. MobileWorks and Samasource explained that it was almost always the case that the number of workers available to complete tasks was larger than the amount of work available, leaving workers waiting for tasks. CrowdFlower provided a nuanced version of this story: in their largest pool of less vetted workers, there are always more workers available than tasks to complete. In their more curated pools of workers that had proven themselves on the platform, the amount of work was greater than the available crowd worker supply. This could indicate many things, such as:

- There is a higher need for skilled work (and therefore skilled workers) in crowdsourcing platforms.
- The skilled workers are not as present in crowdsourcing because they prefer traditional work, or quickly transition out of crowdsourcing before they reach the skilled worker levels (this could mean that there is significant churn in the worker pool).
- Part-time crowd workers are happy staying in the less skilled levels (due to the simplicity of the tasks in that level), even though they could transition to the skilled levels.

| Marketplace | Task Redundancy | Signals of Quality |
|-------------|---|--|
| CrowdFlower | 2-8 workers/task | CrowdFlower's focus on redundant responses to microtasks allows them to rely on traditional gold standard and algorithmic mechanisms for simultaneously determining both work and worker quality. Requesters pick between different methods of redundantly asking questions (e.g., simple N-way redundancy, or stop when algorithm is confident) |
| Upwork | None | Combines information such as education/background/demographics, public and private feedback at the end of a contract, and performance on task-specific qualification tests to determine freelancer quality |
| MobileWorks | Spotchecks: Digitally paired workers & managers | MobileWorks relies heavily on the strength of a manager's review, and aggregates these reviews to create a global notion of worker quality |
| Samasource | Spotchecks: Collocated workers & managers | Since workers and managers are physically collocated, traditional face-to-face training and feedback mechanisms suffice for their quality thresholds |

Table 9.5: The degree of redundancy of tasks, and the general approach to determining work and worker quality on each platform.

Further investigation is needed to reveal which of these reasons, or others, could be responsible for this mismatch.

Furthermore, requesters tended to create tasks at particular times of day and days of week, which CrowdFlower signals to crowd workers so they can plan for ideal times to complete tasks. Finally, CrowdFlower notes that the supply/demand mismatch varies by task type, indicating that even amongst microtasks, some types of labor are more valued than others.

9.4 Quality Control

The quality control mechanisms of each platform relate to and depend on the types of tasks the workers are doing on that platform. In traditional microtask-based labor, the inputs and outputs to various tasks are well-defined enough that one can reconcile multiple workers' responses after redundantly assigning multiple workers to each task. As the complexity of tasks increases, platforms opt for more traditional spot-checked reviews to ensure training and quality. We summarize the various platforms' approaches in Table 9.5. We describe, in turn, task redundancy in Section 9.4.1, signals for inferring task quality in Section 9.4.2, and providing feedback in Section 9.4.3.

9.4.1 Task Redundancy

CrowdFlower, whose focus is on well-defined inputs and outputs to micro-tasks, achieves task quality by offering tasks redundantly to workers. By default, the system asks three workers to complete each microtask and infers the most likely answer based on the majority response and workers' previous track records of responding correctly. It also provides more complex redundancy measures, such as varying the redundancy of each task from 2-8 crowd workers, stopping when it reaches 95% confidence in a response based on the level of agreement of responses. Samasource, while it also offers microtask labor, avoids redundancy-based schemes, leaving quality control to tools like spotchecks of worker responses and continuous training by affiliate delivery centers.

The macrotask-oriented providers also avoid redundancy. On Upwork, the tasks are high-level enough that there would be no easy way to reconcile redundant responses, and quality control is left to requester/freelancer ratings and feedback at the end of a contract. To support less rigid task definitions, MobileWorks pairs crowd workers with managers who review worker output, vet responses, and offer additional training or instruction based on worker accomplishments.

9.4.2 Signals of Worker Quality

We asked the platforms which mechanisms and signals they utilized to infer the overall quality of a worker's output. The responses varied widely.

Samasource takes an organic approach, leaving the task to managers in delivery centers: since workers and managers are physically collocated, traditional face-to-face training and feedback mechanisms suffice for their quality thresholds. Upwork utilize information such as education/background/demographics, public and private feedback at the end of a contract, and performance on task-specific qualification tests to determine freelancer quality and appropriately recommend them for future work.

While CrowdFlower and MobileWorks take different approaches, they collect and rely on the largest amount of features to characterize crowd worker quality. CrowdFlower's focus on redundant responses to microtasks allows them to rely on traditional gold standard and algorithmic mechanisms

for simultaneously determining both work and worker quality. MobileWorks relies more heavily on the strength of a manager's review than on redundantly assigning tasks, and aggregates these reviews to create a global notion of worker quality. Both systems also rely on worker engagement on the platform, including factors such as relative task completion time of different workers, the number of tasks crowd workers recently completed, and workers' overall tenure in the system. Because of its scale, CrowdFlower also looks for workers with shared IPs to identify collusion.

One last signal that both CrowdFlower and Samasource identifies is "gold standard" tasks. This approach, generally targeted at microtasks, requires requesters to provide some example microtask along with their answers, which the platforms then integrate into a worker's workflow. If workers consistently provide an incorrect response to these gold standard tasks, the platform can determine that either the worker needs further training, the worker is acting maliciously, or, commonly, the task is underspecified or confusing.

Generally, all of the respondents that utilized signals avoided complex inference techniques for determining overall worker quality. They rely more heavily on a set of simple summary statistics instead of complex approaches. CrowdFlower in particular explained that they are concerned with increasingly complex models, as these models might offer up more challenging-to-understand adversarial models, and are also more difficult to explain to workers.

9.4.3 Providing Workers with Feedback

In addition to ensuring the quality of a worker's work output and matching the worker with appropriate work in the future, the other benefit of quality assurance is that it can serve as a mechanism for offering workers feedback on their progress and on which areas of expertise they need to strengthen. While all of the participants provide some form of feedback, the format differs. MobileWorks and CrowdFlower offer workers some aggregate measures of their performance, but also provide feedback on every single task, letting crowd workers know when they made particular mistakes. On MobileWorks, such feedback comes in the form of a manager's review and rating of a task, whereas on CrowdFlower, it comes from a contributor's agreement with other workers that answered a particular question. Samasource offers feedback,

generally by way of a manager in a delivery center. Feedback comes to freelancers on Upwork in the form of free-form and ratings in public and private forms after a contract has ended.

One area that respondents highlighted as challenging is transparency. In cases where a crowd worker can be promoted or qualified to do more challenging or interesting work, how should the platform let them know that they are on track? What is the best way to explain how a worker should focus to reach the next level? These questions become more challenging as the platforms scale and such feedback is provided in some automated form.

10

Conclusion

Crowdsourced data processing has made its way into most large technology companies, and has powered the innovations behind several successful startups. At the same time, over the past few years, there has been a lot of excitement in the academic community (including the development of a new conference) on identifying creative and efficient uses of crowd work, and on fully leveraging the potential benefits of crowdsourced data processing.

Unfortunately, the industrial and academic pursuits of crowd-powered data processing systems have largely grown in parallel, preventing academics from getting inspiration from applications, and industry users from adopting the finest academic contributions. With this book, we have sought to bridge the two communities, both by summarizing the state of the art in academia, and by speaking with industrial users and marketplace providers of crowd work, to identify their approaches, philosophies, and concerns.

Several academic communities have contributed to the early days of crowdsourcing research, including the databases, economics, human-computer interaction, machine learning/artificial intelligence, psychology, and theory communities. The communities provided several inspirational examples for future applications of crowd work and human computation. Additionally, the communities established some fundamental building blocks

of crowdsourced workflows, including systems for building the workflows, algorithms for ensuring quality worker output, techniques to motivate good work, novel interfaces for completing crowd work, demographic studies of the crowd, and, importantly, an exploration of the ethics of crowd work.

Through our surveys of industry users and marketplace providers of crowd work, we have found that not only do technology companies make heavy use of crowd work for data processing, but that they have also invested several tens of full-time engineers and designers and millions of dollars into these workflows. Industry users of crowd work appear to be benefiting from the work in diverse application areas, and are looking to expand the parts of their businesses that crowd work can power. While crowdsourced data processing has found a home in industry, it is still very much in its infancy. The workflows our industry participants described are relatively simple, and most of the inference algorithms industry users have applied to ensure high quality work are simple compared to contributions from the academic community.

In the other direction, industry has also explored aspects of the design space left virtually untouched by the academic community. Most notably, we find that crowd worker tenures on projects are often measured in years and that a common approach to ensuring quality is establishing longer-term relationships with workers. The community could benefit from more studies in crowd worker tenure and longevity.

We view these gaps between industry and academia as opportunities. Industrial applications of crowd work have a long way to grow, and many novel applications and efficiencies that the academic community has contributed are yet unexplored. Academic researchers can see more adoption of their novel algorithms and approaches by releasing more of their contributions as usable software artifacts that industry can integrate. Future academic contributions can come from observing how industrial users apply crowd work.

As crowd work becomes more prevalent, one final area of research is arguably most important to its longevity: the development of sustainable, just, and ethical labor models through which it can grow. This is one area that we see academics, practitioners, and policymakers coming together to build a better future for crowd work and crowd workers. While the issue is currently on the fringes of most discussions about crowdsourcing, it must increasingly be front and center if crowdsourcing is to succeed in the long run.

Crowdsourced data management has arrived, and is being adopted widely in many novel application areas. In the decades to come, we hope to see more exciting work and collaborations in the space between academia and industry. With a solid foundation established by both communities, we are excited to see where the future takes us.

Acknowledgements

We are grateful to the cast of characters that made this book possible. All of the novel content and learnings in this book are due to the anonymous industry and marketplace participants that set aside a few hours each to answer our detailed survey questions. Without your years of experience distilled down into these conversations, this book would be tremendously dull and boring. We are also ever grateful to the crowd workers that make this all possible.

The content assistance and feedback we received throughout the process were also so helpful. Thank you to Joe Hellerstein, who invited us to start the endeavor with James Finlay, our editor, and to James, Mike Cafarella, and the anonymous reviewers that gave our early drafts a critical eye. We were so lucky to have feedback on our survey questions, tables, and figures from Meredith Blumenstock, who ended up recreating a few of them from scratch.

We are thankful to our institutions for giving us the freedom to pursue this book. The University of Illinois, GoDaddy, and Unlimited Labs were all accommodating and supportive throughout the process. A lot of our early content and experiences were also shaped by our thesis research, and we are ever-indebted to our advisors and collaborators while at MIT and Stanford for helping form our thoughts on the world of crowd-powered data processing.

Finally, we are so thankful to our parents and spouses for being our sources of inspiration and support throughout this process. Typing “thank you” on a keyboard does not begin to highlight how grateful we are to Dipti and Meredith.

Appendices

A

Industry Users Survey

Part 1 of 6: Crowd Use Cases

Which of the following use cases of crowds apply to the tasks your team is solving? (place an X inside the [] for all options that apply):

- classification
- entity resolution/matching
- schema mapping
- spam detection
- content moderation
- text generation
- data cleaning/normalization
- data extraction
- other(s):

For any of the use cases you've checked off above, please provide a one- or two-line description of your use case

What other use cases of crowdsourcing do you know about within your company beyond your team?

How did you solve these problems before crowdsourcing?

In your words, what are the benefits of crowdsourcing? How do you sell it within your organization?

Part 2 of 6: Crowd Management

How many people on your team / company work on crowdsourcing?

Do you use explicit crowd work (e.g., paid workers, volunteer workers), or implicit crowd work (e.g., email spam tagging by users or search keywords entered by users) or both?

If you use explicit crowd work, please answer the following questions:

- A. How do you recruit crowd workers -- which platforms, if any? Do you have in-house crowd workers or do you use external crowds? In either case, explain why?
- B. How many tasks per week do your crowd workers complete? How many crowd workers do you work with, in a week? What is the median/max length of a relationship you've had with a crowd worker?
- C. If you use explicit, paid crowd work, roughly how much money do you spend per week on compensating crowd workers?

Part 3 of 6: Quality of Work and Workers

How do you evaluate worker quality? (place an X inside the [] for all options that apply)

- gold standard tasks interspersed between real tasks
- performance on quizzes/tests/qualification tasks before real tasks are asked
- disagreement measures (e.g., how often does a worker disagree with the majority)
- expectation-maximization-style procedures where you determine worker quality and work quality simultaneously

- task completion speed
- reviews/ verification , where another worker or expert determines the correctness of the work output
- external measures, such as education, background
- other(s):

How often do you calculate worker quality? (place an X inside the for all options that apply)

- every task they answer
- periodically , say after a week, or after completing X tasks
- never, once we evaluate them up-front
- other(s):

Given your evaluations of workers, how do you evaluate the overall quality of a task? (place an X inside the for all options that apply)

- expectation-maximization-style procedures where you determine worker quality and work quality simultaneously
- given worker qualities learned upfront, use techniques like maximum-likelihood to infer the highest probability answer per task
- simple aggregation measures that ignore worker quality, e.g., majority vote
- reviews/ verification , where another worker or expert determines the correctness of the work output
- other(s):

Would/do you benefit from optimization algorithms to reduce the cost or improve accuracy of answers? If yes, which algorithms do you employ; if no, why not?

Do you provide feedback to workers (on how well they are doing) per-task or overall? If so, how?

Part 4 of 6: Incentives/Payment Mechanisms

What incentive mechanisms do you use? (place an X inside the for all options that apply)

- hourly payment
- per-task payment
- bonuses
- gamification

- leaderboards
- promotion to a higher position
- other(s):

For any of the mechanisms you've checked off above, please provide a one- or two-line description of your mechanism.

Do you have different classes of crowd workers? What are the classes? Are they hierarchical (e.g., managers vs. employees), or are they based on different skill-sets (e.g., workers good at classification vs. workers good at writing textual passages)?

Are your incentive mechanisms different for different crowd workers? How much transparency do workers have into your payment/promotion schemes? How many roles has a crowd worker served in with you?

Part 5 of 6: Task Design/Decomposition Questions

How are crowds integrated into your internal workflows? (place an X inside the for all options that apply):

- train: crowds provide training data for classifiers, that then process all subsequent tasks
- uncertain: when an algorithm is uncertain about a response, it asks the crowd
- active: algorithms determine where to get training data from crowds (using active learning)
- stand-alone: crowds answer all questions
- test: crowds are used to verify or evaluate the output of possibly competing algorithms
- other(s):

For any of the integrations you've checked off above, please provide a one- or two-line description of your integration, and how often it is used.

What frameworks (e.g., TurKit, Clockwork Raven), APIs (e.g., MTurk API), or libraries (e.g., Get-Another-Label) do you use in building your crowd workflows?

How much iteration or A/B testing goes into your task design?

Do you use primarily microtasks or macrotasks?

What fraction of your crowdsourced workflows have more than one crowdsourcing step involved? You can answer as a fraction (e.g., 3 of the 7 workflows we've built involve a human in more than one step), or with a more qualitative measure (e.g., None, Some, Most, All).

Have you ever employed workflows of the following form? (place an X inside the [] for all options that apply):

- iterative refinement (Turkit-style)
- find-fix-verify (Soylent-style)
- do-verify (hierarchies of more trusted workers vetting other workers)
- other(s):

If you checked "other" above, please explain

Do your team/company have some form of collaboration or interaction between workers working on tasks? If so, please explain:

Part 6 of 6: Challenges

On a scale from 1 (not challenging at all) to 7 (extremely challenging), how do you rate each of the following challenges?

- recruiting
- crowd training and initial vetting
- gold standard task creation
- task decomposition and designing workflows
- user interface design
- incentive design
- payment mechanisms
- gamification
- identifying correct responses
- identifying high/low quality workers
- providing workers with feedback on their prior work
- integrating crowds with other algorithms
- eliminating cognitive bias in worker responses
- addressing differences in cultural knowledge amongst workers
- other: please explain

How long did this survey take you?

Thank you so much for your time and insights!!

B

Marketplace Providers Survey

Part 1 of 5: Crowd characteristics

For all of these questions, provide as precise an answer as you can, but if you only have rough descriptions, that's fine as well.

What is the approximate age distribution of your crowd? (Please write a rough percentage within the [] for each type that you see in your marketplace; if you don't know the exact numbers, place an X inside the [] for all options that apply)

- [] 0-18
- [] 18-30
- [] 30-40
- [] 40-50
- [] 50-60
- [] 60+

What is the approximate gender distribution of your crowd?

What is the approximate educational achievement distribution of your crowd?

What is the approximate per-country distribution or geographical distribution of your crowd?

What fraction of your crowd uses your platform as their primary job?

Across all crowd marketplaces, how many daily/weekly/monthly active workers are there? If you have a source for this estimate, it would be helpful for us.

Part 2 of 5: Implementations

What fraction of the tasks in your marketplace are microtasks (i.e., a small number of simple questions, typically a few minutes) versus macrotasks (i.e., involved work/answers, up to a few hours). A rough estimate is OK (write a rough percentage within the [] for each type that you see in your marketplace; if you don't know the exact numbers, place an X inside the [] for all options that apply):

- Almost all microtasks
- Mostly microtasks
- Roughly even
- Mostly macrotasks
- Almost all macrotasks

If your company primarily deals with either microtasks or macrotasks, is there a particular reason or set of reasons why your company is focusing on that market? Please explain.

Please provide some insights into what fraction of the microtasks fall under the following types (write a rough percentage within the [] for each type that you see in your marketplace; if you don't know the exact numbers, place an X inside the [] for all options that apply):

- classification
- entity resolution/matching
- schema mapping
- spam detection
- content moderation
- text generation
- data cleaning/normalization
- data extraction
- translation, transcription
- other(s):

What is the distribution of # assignments/redundancy set by the requester of each task?

Do you provide an API or other programmatic access to the following information? (place an X inside the [] for all options that apply)

- retrieving worker information/statistics
- for worker payment
- for posting tasks and retrieving responses

How do requesters use your platform? For each of the following, write a number inside the [] between 1 (not at all) and 7 (very often) for how often requesters use the following modalities:

- [] using a user interface you provide to requesters (if any)
- [] using an API you provide with their own custom interfaces (if any)
- [] having you completely handle the management of tasks and workers

How do you match workers to tasks? (place an X inside the [] for all options that apply)

- [] requesters decide which workers to hire
- [] workers decide which tasks to work on; if so, do you further use the following constraints to limit which tasks workers can work on? (place an X inside the [] for all options that apply)
 - [] generic qualifications (e.g., worker rating, country of origin)
 - [] task-specific qualifications (e.g., workers must have image tagging mastery)
 - [] blacklist (i.e., requesters bar some workers from working on tasks, possibly because they did poorly in the past)
- [] whitelist (e.g., requesters invite a set of trusted workers from previous tasks to the current one)
- [] other: please explain
- [] automatically match workers to tasks; if so, do you use the following signals to infer the best matching? (place an X inside the [] for all options that apply)
 - [] requester specifications of generic worker and task properties (e.g., qualifications, constraints, task type)
- [] automatically inferred task type (e.g., automatically identifying that a task is an image tagging task)
- [] automatically inferred task type-specific worker abilities (e.g., abilities inferred from the past performance of workers on image tagging tasks and/or the educational background of workers)
- [] worker-reported task type-specific abilities (e.g., "I am great at image tagging!")
- [] other: please explain
- [] other: please explain

Do you see a benefit for a programming toolkit or workflow management tool that would allow your requesters to easily create common workflows and reason about worker responses? Why or why not?

Does such a tool already exist, and if not, what limitations do you see in current workflow management tools?

Part 3 of 5: Statistics

For all of these questions, provide as precise an answer as you can, but if you only have rough descriptions, that's fine as well.

What is the distribution of # unique request types (e.g., "label image 1" and "label image 2" are a single request type) per requester?

What is the distribution of # total tasks (e.g., "label image 1" and "label image 2" are two tasks) per requester?

What is the distribution of # workers per requester?

What is the distribution of spending per requester?

Do you ever observe an imbalance between the number of workers and number of tasks at any point?

Yes

If yes, do you see: (place an X inside the for all options that apply)

The number of available workers being greater than the number of available tasks, and hence workers have to wait

The number of tasks being greater than the number of available workers, and hence tasks take a very long time to complete

No

If you observe an imbalance, what does the imbalance depend on? (place an X inside the for all options that apply)

time of day, month or year

pricing level (e.g., for some task prices, there is an imbalance between number of workers and tasks)

task type (e.g., for image tagging tasks, there is an imbalance between the number of workers and tasks)

Part 4 of 5: Quality assurance

If you infer trustworthiness of workers, what mechanisms do you use to infer trustworthiness scores? (place an X inside the for all options that apply)

education, background, demographics

- level of engagement on the platform:
 - time spent on average
 - tasks completed
 - time since registering for the service
 - acceptance scores from requesters
 - task completion speed
- accuracy, computed using the following measures:
 - gold-standard tasks interspersed between real tasks
 - performance on quizzes/tests/qualification tasks before real tasks are asked
 - disagreement measures (e.g., how often does a worker disagree with the majority)
 - expectation maximization style procedures to simultaneously determine worker quality and work quality simultaneously
 - reviews/ verification , where another worker or expert determines the correctness of the work

If you don't use complex schemes for inferring accuracy or trust, given the research on quality assurance schemes, is there a reason why you chose to not use them? Please explain.

Given your trustworthiness scores of workers, how do you evaluate the overall quality of a task? (place an X inside the for all options that apply)

- expectation-maximization-style procedures where you determine worker quality and work quality simultaneously
- given worker qualities learned upfront, use techniques like maximum-likelihood to infer the highest probability answer per task
- simple aggregation measures that ignore worker quality, e.g., majority vote
- reviews/ verification , where another worker or expert determines the correctness of the work output
- other(s): please explain

Do you provide feedback to workers (positive and/or negative feedback) per-task or overall? If so, how?

Part 5 of 5: Challenges

What do you view as requesters' challenges? On a scale from 1 (not challenging at all) to 7 (extremely challenging), how do you rate each of the

following challenges?

- recruiting
- crowd training and initial vetting
- gold standard task creation
- task decomposition and designing workflows
- user interface design
- incentive design
- payment mechanisms
- gamification
- identifying correct responses
- identifying high/low quality workers
- providing workers with feedback on their prior work
- integrating crowds with other algorithms
- eliminating cognitive bias in worker responses
- addressing differences in cultural knowledge amongst workers
- other: please explain

Where do you see yours and other marketplaces evolving to in the future? On a scale from 1 (not likely at all) to 7 (highly likely), how do you rate the following scenarios?

- increasing levels of associations of “ identities ” with workers, making them more responsible for their work
- moving towards macro instead of micro tasks
- moving towards more complex tasks (programming, virtual assistance) rather than simple ones (labeling, etc.)
- increasing levels of skillsets associated with workers, possibly with qualification tests or hierarchies of workers (“managers” vs. “ employees”)
- other: please give a two line description

Is there anything else you would like to add?

Approximately how long did this survey take you to complete?

Approximately what percentage of the survey did you need assistance from one or more colleagues in order to complete?

Thank you so much for your time and insights!!!!

References

- [1] BitcoinReserve (Retrieved 16 March 2015). <http://bitcoinreserve.org/>.
- [2] ClickWorker (Retrieved 22 July 2013). <http://clickworker.com>.
- [3] ClixSense (Retrieved 16 March 2015). <http://www.clixsense.com/>.
- [4] Cloud Me Baby (Retrieved 22 July 2013). <http://www.cloudmebaby.com>.
- [5] CrowdFlower Content Moderation Platform (Retrieved 14 August 2013). <http://crowdfLOWER.com/type-content-moderation>.
- [6] CrowdFlower (Retrieved 22 July 2013). <http://crowdfLOWER.com>.
- [7] CSCW (Retrieved 22 July 2013). <http://cscw.acm.org>.
- [8] Duolingo Inc. (Retrieved 14 August 2013). <http://www.duolingo.com>.
- [9] Google, Inc. (Retrieved 14 August 2013). <http://www.google.com>.
- [10] Mechanical Turk (Retrieved 22 July 2013). <http://www.mturk.com>.
- [11] Microsoft Bing (Retrieved 14 August 2013). <http://www.bing.com>.
- [12] Samasource Impact Report (Retrieved 16 March 2015). <http://www.samasource.org/impact/>.
- [13] Samasource Jobs (Retrieved 16 March 2015). <http://www.samasource.org/people/#jobs>.
- [14] Samasource (Retrieved 22 July 2013). <http://samasource.com>.
- [15] Swagbucks (Retrieved 16 March 2015). <http://www.swagbucks.com/>.
- [16] Turker Nation (Retrieved 22 July 2013). <http://www.turkernation.com>.

- [17] Upwork (Retrieved August 4 2015). <https://www.upwork.com/>.
- [18] Yahoo! Inc. (Retrieved 14 August 2013). <http://www.yahoo.com>.
- [19] The Apache Pig project, April 2012. <http://pig.apache.org/>.
- [20] Captricity website, April 2012. <http://captricity.com/>.
- [21] Mobileworks website, April 2012. <http://www.mobileworks.com/>.
- [22] Salman Ahmad, Alexis Battle, Zahan Malkani, and Sepandar D. Kamvar. The jabberwocky programming environment for structured social computing. In *Proceedings of the Annual ACM Symposium on User Interface Software and Technology*, pages 53–64, 2011.
- [23] Miklós Ajtai, Vitaly Feldman, Avinatan Hassidim, and Jelani Nelson. Sorting and selection with imprecise comparisons. In *Proceedings of the International Colloquium on Automata, Languages and Programming*, pages 37–48, 2009.
- [24] Omar Alonso. Implementing crowdsourcing-based relevance experimentation: an industrial perspective. *Information Retrieval*, 16(2):101–120, 2013.
- [25] Omar Alonso, Catherine C. Marshall, and Marc Najork. Debugging a crowd-sourced task with low inter-rater agreement. In *Proceedings of the 15th ACM/IEEE-CS Joint Conference on Digital Libraries*, pages 101–110, New York, NY, USA, 2015. ACM.
- [26] Omar Alonso, Daniel E. Rose, and Benjamin Stewart. Crowdsourcing for relevance evaluation. *SIGIR Forum*, 42(2):9–15, 2008.
- [27] Yael Amsterdamer, Yael Grossman, Tova Milo, and Pierre Senellart. Crowd mining. In *SIGMOD Conference*, pages 241–252, 2013.
- [28] Arvind Arasu, Michaela Götz, and Raghav Kaushik. On active learning of record matching packages. In *SIGMOD Conference*, pages 783–794, 2010.
- [29] Shivnath Babu and Jennifer Widom. Continuous queries over data streams. *ACM Sigmod Record*, 30(3):109–120, 2001.
- [30] Eytan Bakshy, Jake M. Hofman, Winter A. Mason, and Duncan J. Watts. Everyone’s an influencer: quantifying influence on twitter. In *Proceedings of the ACM International Conference on Web Search and Data Mining*, pages 65–74, 2011.
- [31] Maria-Florina Balcan, Alina Beygelzimer, and John Langford. Agnostic active learning. *Journal of Computer and System Sciences*, 75(1):78–89, 2009.
- [32] Daniel W. Barowy, Charlie Curtsinger, Emery D. Berger, and Andrew McGregor. Automan: a platform for integrating human-based and digital computation. In *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 639–654, 2012.

- [33] Jeff Barr and Luis-Felipe Cabrera. AI gets a brain. *ACM Queue*, 4(4):24–29, 2006.
- [34] Kedar Bellare, Suresh Iyengar, Aditya G. Parameswaran, and Vibhor Rastogi. Active sampling for entity matching. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1131–1139, 2012. Online: <http://doi.acm.org/10.1145/2339530.2339707>.
- [35] Michael S. Bernstein, Joel Brandt, Robert C. Miller, and David R. Karger. Crowds in two seconds: enabling realtime crowd-powered interfaces. In *Proceedings of the Annual ACM Symposium on User Interface Software and Technology*, pages 33–42, 2011.
- [36] Michael S. Bernstein, Greg Little, Robert C. Miller, Björn Hartmann, Mark S. Ackerman, David R. Karger, David Crowell, and Katrina Panovich. Soylent: a word processor with a crowd inside. In *Proceedings of the Annual ACM Symposium on User Interface Software and Technology*, pages 313–322, 2010.
- [37] Alina Beygelzimer, Sanjoy Dasgupta, and John Langford. Importance weighted active learning. In *Proceedings of the International Conference on Machine Learning*, page 7, 2009.
- [38] Alina Beygelzimer, Daniel Hsu, John Langford, and Tong Zhang. Agnostic active learning without constraints. In *Proceedings of the Annual Conference on Neural Information Processing Systems*, pages 199–207, 2010.
- [39] Jeffrey P. Bigham, Chandrika Jayant, Hanjie Ji, Greg Little, Andrew Miller, Robert C. Miller, Robin Miller, Aubrey Tatarowicz, Brandyn White, Samuel White, and Tom Yeh. Vizwiz: nearly real-time answers to visual questions. In *Proceedings of the Annual ACM Symposium on User Interface Software and Technology*, pages 333–342, 2010.
- [40] Philip Bohannon, Srujana Merugu, Cong Yu, Vipul Agarwal, Pedro DeRose, Arun Shankar Iyer, Ankur Jain, Vinay Kakade, Mridul Muralidharan, Raghu Ramakrishnan, and Warren Shen. Purple sox extraction management system. *SIGMOD Record*, 37(4):21–27, 2008.
- [41] Alessandro Bozzon, Marco Brambilla, and Stefano Ceri. Answering search queries with crowdsearcher. In *Proceedings of the International World Wide Web Conference*, pages 1009–1018, 2012.
- [42] Alessandro Bozzon, Marco Brambilla, Stefano Ceri, and Andrea Mauri. Reactive crowdsourcing. In *Proceedings of the International World Wide Web Conference*, pages 153–164, 2013.

- [43] Alessandro Bozzon, Marco Brambilla, Stefano Ceri, Matteo Silvestri, and Giuliano Vesci. Choosing the right crowd: expert finding in social networks. In *Proceedings of the International Conference on Extending Database Technology*, pages 637–648, 2013.
- [44] Jonathan Bragg, Mausam, and Daniel S. Weld. Crowdsourcing multi-label classification for taxonomy creation. In *Proceedings of the 1st AAAI Conference on Human Computation and Crowdsourcing*, 2013.
- [45] Steve Branson, Catherine Wah, Boris Babenko, Florian Schroff, Peter Welinder, Pietro Perona, and Serge Belongie. Visual recognition with humans in the loop. In *European Conference on Computer Vision*, Heraklion, Crete, Sept. 2010.
- [46] Nicolas Bruno. Minimizing database repros using language grammars. In *Proceedings of the International Conference on Extending Database Technology*, pages 382–393, 2010.
- [47] David R. Butenhof. *Programming with POSIX Threads*. Addison-Wesley Professional, 1997.
- [48] Caleb Chen Cao, Jieying She, Yongxin Tong, and Lei Chen. Whom to ask? jury selection for decision making tasks on micro-blog services. *Proceedings of the VLDB Endowment*, 5(11):1495–1506, 2012.
- [49] Stuart K Card, Thomas P Moran, and Allen Newell. *The psychology of human-computer interaction*. Lawrence Erlbaum Associates, 1983.
- [50] Ruggiero Cavallo and Shaili Jain. Efficient crowdsourcing contests. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems*, pages 677–686, 2012.
- [51] Xiaoyong Chai, Ba-Quy Vuong, AnHai Doan, and Jeffrey F. Naughton. Efficiently incorporating user feedback into information extraction and integration programs. In *SIGMOD Conference*, pages 87–100, 2009.
- [52] Dana Chandler and John Joseph Horton. Labor allocation in paid crowdsourcing: Experimental evidence on positioning, nudges and prices. In *Human Computation*, 2011.
- [53] Kuan-Ta Chen, Chen-Chi Wu, Yu-Chun Chang, and Chin-Laung Lei. A crowdsourcable qoe evaluation framework for multimedia content. In *ACM Multimedia*, pages 491–500, 2009.
- [54] Xi Chen, Qihang Lin, and Dengyong Zhou. Optimistic knowledge gradient policy for optimal budget allocation in crowdsourcing. In *Proceedings of the 30th International Conference on Machine Learning*, pages 64–72, 2013.

- [55] Lydia B. Chilton, Greg Little, Darren Edge, Daniel S. Weld, and James A. Landay. Cascade: crowdsourcing taxonomy creation. In *ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 1999–2008, 2013.
- [56] David A. Cohn, Les E. Atlas, and Richard E. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.
- [57] Seth Cooper, Firas Khatib, Adrien Treuille, Janos Barbero, Jeehyung Lee, Michael Beenen, Andrew Leaver-Fay, David Baker, Zoran Popović, and Foldit players. Predicting protein structures with a multiplayer online game. *Nature*, 466(7307):756–760, 2010.
- [58] Peng Dai, Christopher H. Lin, Mausam, and Daniel S. Weld. Pomdp-based control of workflows for crowdsourcing. *Artif. Intell.*, 202:52–85, 2013.
- [59] Nilesh Dalvi, Anirban Dasgupta, Ravi Kumar, and Vibhor Rastogi. Aggregating crowdsourced binary ratings. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 285–294. International World Wide Web Conferences Steering Committee, 2013.
- [60] Nilesh Dalvi, Ravi Kumar, Bo Pang, Raghu Ramakrishnan, Andrew Tomkins, Philip Bohannon, Sathiya Keerthi, and Srujana Merugu. A web of concepts. In *Symposium on Principles of Database Systems*, pages 1–12, 2009.
- [61] Nilesh Dalvi, Aditya G. Parameswaran, and Vibhor Rastogi. Minimizing uncertainty in pipelines. In *Proceedings of the Annual Conference on Neural Information Processing Systems*, pages 2951–2959, 2012.
- [62] Nilesh Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. *Very Large Data Base Journal (VLDBJ)*, 16(4):523–544, 2007.
- [63] Sanjoy Dasgupta, Daniel Hsu, and Claire Monteleoni. A general agnostic active learning algorithm. In *Proceedings of the Annual Conference on Neural Information Processing Systems*, 2007.
- [64] Sanjoy Dasgupta and John Langford. Tutorial summary: Active learning. In *Proceedings of the International Conference on Machine Learning*, page 178, 2009.
- [65] Susan B. Davidson, Sanjeev Khanna, Tova Milo, and Sudeepa Roy. Using the crowd for top-k and group-by queries. In *Proceedings of the International Conference on Database Theory*, pages 225–236, 2013.
- [66] A. P. Dawid and A. M. Skene. Maximum likelihood estimation of observer error-rates using the em algorithm. *Applied Statistics*, 28(1):20–28, 1979.
- [67] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, January 2008.

- [68] Gianluca Demartini, Djellel Eddine Difallah, and Philippe Cudré-Mauroux. ZenCrowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In *WWW '12: Proceedings of the 21st International Conference on World Wide Web*. ACM, April 2012.
- [69] Gianluca Demartini, Beth Trushkowsky, Tim Kraska, and Michael J. Franklin. CrowdQ: Crowdsourced query understanding. In *Proceedings of the Conference on Innovative Data Systems Research*, 2013.
- [70] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.
- [71] Amol Deshpande and Samuel Madden. Mauvedb: supporting model-based user views in database systems. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, pages 73–84, New York, NY, USA, 2006. ACM.
- [72] Daniel Deutch, Ohad Greenshpan, Boris Kostenko, and Tova Milo. Using markov chain monte carlo to play trivia. In *International Conference on Data Engineering*, pages 1308–1311, 2011.
- [73] AnHai Doan, Michael J. Franklin, Donald Kossmann, and Tim Kraska. Crowdsourcing Applications and Platforms: A Data Management Perspective. *Proceedings of the VLDB Endowment*, 4(12):1508–1509, 2011.
- [74] AnHai Doan, Raghu Ramakrishnan, and Alon Y. Halevy. Crowdsourcing systems on the world-wide web. *Communications of the ACM*, 54(4):86–96, 2011.
- [75] Pinar Donmez, Jaime G. Carbonell, and Jeff G. Schneider. Efficiently learning the accuracy of labeling sources for selective sampling. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 259–268, 2009.
- [76] John R. Douceur. The Sybil Attack. In *Proceedings of the International Workshop of Peer-to-Peer Systems*, pages 251–260, 2002.
- [77] Steven Dow, Anand Pramod Kulkarni, Scott R. Klemmer, and Björn Hartmann. Shepherding the crowd yields better work. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pages 1013–1022, 2012.
- [78] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate Record Detection: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, 2007.

- [79] Amber Feng, Michael J. Franklin, Donald Kossmann, Tim Kraska, Samuel Madden, Sukriti Ramesh, Andrew Wang, and Reynold Xin. CrowdDB: Query Processing with the VLDB Crowd. *Proceedings of the VLDB Endowment*, 4(12):1387–1390, 2011.
- [80] Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. CrowdDB: answering queries with crowdsourcing. In *SIGMOD Conference*, pages 61–72, 2011.
- [81] Yihan Gao and Aditya G. Parameswaran. Finish them!: Pricing algorithms for human computation. *Proceedings of the VLDB Endowment*, 7(14):1965–1976, 2014.
- [82] Arpita Ghosh. Game theory and incentives in human computation systems. In *Handbook of Human Computation*. Springer, 2013.
- [83] Arpita Ghosh, Satyen Kale, and Preston McAfee. Who Moderates the Moderators? Crowdsourcing Abuse Detection in User-Generated Content. In *Proceedings of the ACM Conference on Economics and Computation*, pages 167–176, 2011.
- [84] Arpita Ghosh and R. Preston McAfee. Crowdsourcing with endogenous entry. In *Proceedings of the International World Wide Web Conference*, pages 999–1008, 2012.
- [85] Chaitanya Gokhale, Sanjib Das, AnHai Doan, Jeffrey F. Naughton, Narasimhan Rampalli, Jude Shavlik, and Xiaojin Zhu. Corleone: Hands-Off Crowdsourcing for Entity Matching. *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, March 2014.
- [86] Ryan Gomes, Peter Welinder, Andreas Krause, and Pietro Perona. Crowd-clustering. In *Proceedings of the Annual Conference on Neural Information Processing Systems*, pages 558–566, 2011.
- [87] Pankaj Gulhane, Amit Madaan, Rupesh R. Mehta, Jeyashankher Ramamirtham, Rajeev Rastogi, Sandeepkumar Satpal, Srinivasan H. Sengamedu, Ashwin Tengli, and Charu Tiwari. Web-scale information extraction with vertex. In *International Conference on Data Engineering*, pages 1209–1220, 2011.
- [88] Pankaj Gulhane, Rajeev Rastogi, Srinivasan H. Sengamedu, and Ashwin Tengli. Exploiting content redundancy for web information extraction. *Proceedings of the VLDB Endowment*, 3(1):578–587, 2010.
- [89] Stephen Guo, Aditya G. Parameswaran, and Hector Garcia-Molina. So who won?: dynamic max discovery with the crowd. In *SIGMOD Conference*, pages 385–396, 2012. Online: <http://doi.acm.org/10.1145/2213836.2213880>.

- [90] Maya R. Gupta and Yihua Chen. Theory and use of the em algorithm. *Foundations and Trends in Signal Processing*, 4(3):223–296, 2010.
- [91] Daniel Haas, Jason Ansel, Lydia Gu, and Adam Marcus. Argonaut: Macro-task Crowdsourcing for Complex Data Processing. *Proceedings of the VLDB Endowment*, 8(12):1642–1653, 2015.
- [92] Steve Hanneke. A bound on the label complexity of agnostic active learning. In *Proceedings of the International Conference on Machine Learning*, pages 353–360, 2007.
- [93] Björn Hartmann and Panagiotis G. Ipeirotis. What’s the right price? pricing tasks for finishing on time. 2011.
- [94] Hannes Heikinheimo and Antti Ukkonen. The Crowd-Median Algorithm. In *Proceedings of the 1st AAAI Conference on Human Computation and Crowdsourcing*, 2013.
- [95] Paul Heymann and Hector Garcia-Molina. Turkalytics: analytics for human computation. In *Proceedings of the 20th international conference on World wide web*, pages 477–486, 2011.
- [96] John Joseph Horton and Lydia B. Chilton. The labor economics of paid crowdsourcing. In *ACM Conference on Electronic Commerce*, pages 209–218, 2010.
- [97] Eric Huang, Haoqi Zhang, David C. Parkes, Krzysztof Z. Gajos, and Yiling Chen. Toward automatic task design: a progress report. In *Proceedings of the Conference on Human Computation and Crowdsourcing*, New York, NY, USA, 2010.
- [98] Panagiotis G. Ipeirotis. Analyzing the amazon mechanical turk marketplace. *XRDS: Crossroads*, 17:16–21, December 2010.
- [99] Panagiotis G. Ipeirotis. Demographics of mechanical turk. Technical report, March 2010.
- [100] Panagiotis G. Ipeirotis, Foster Provost, and Jing Wang. Quality management on amazon mechanical turk. In *Proceedings of the Conference on Human Computation and Crowdsourcing*, New York, NY, USA, 2010.
- [101] Shaili Jain, Yiling Chen, and David C. Parkes. Designing incentives for online question and answer forums. In *ACM Conference on Electronic Commerce*, pages 129–138, 2009.
- [102] Shaili Jain and David C. Parkes. A Game-Theoretic Analysis of Games with a Purpose. In *Proceedings of the Conference on Web and Internet Economics*, pages 342–350, 2008.

- [103] Shaili Jain and David C. Parkes. The role of game theory in human computation systems. In *Proceedings of the Conference on Human Computation and Crowdsourcing*, pages 58–61, 2009.
- [104] Shawn R. Jeffery, Michael J. Franklin, and Alon Y. Halevy. Pay-as-you-go user feedback for dataspace systems. In *SIGMOD Conference*, pages 847–860, 2008.
- [105] Shawn R. Jeffery, Liwen Sun, Matt DeLand, Nick Pendar, Rick Barber, and Andrew Galdi. Arnold: Declarative crowd-machine data integration. In *Proceedings of the Conference on Innovative Data Systems Research*, 2013.
- [106] Manas Joglekar, Hector Garcia-Molina, and Aditya Parameswaran. Evaluating the Crowd with Confidence. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2013. Online: <http://dl.acm.org/citation.cfm?id=2487575.2487595>.
- [107] Manas Joglekar, Hector Garcia-Molina, and Aditya Parameswaran. Evaluating the crowd with confidence. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 686–694. ACM, 2013.
- [108] Manas Joglekar, Hector Garcia-Molina, and Aditya Parameswaran. Comprehensive and Reliable Crowd Assessment Algorithms. In *International Conference on Data Engineering*, 2015.
- [109] Ece Kamar, Severin Hacker, and Eric Horvitz. Combining human and machine intelligence in large-scale crowdsourcing. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems*, pages 467–474, 2012.
- [110] Ece Kamar and Eric Horvitz. Incentives for truthful reporting in crowdsourcing. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1329–1330, 2012.
- [111] Nikos Karampatziakis and John Langford. Online importance weight aware updates. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 392–399, 2011.
- [112] David R. Karger, Sewoong Oh, and Devavrat Shah. Iterative learning for reliable crowdsourcing systems. In *Proceedings of the Annual Conference on Neural Information Processing Systems*, pages 1953–1961, 2011.
- [113] David R. Karger, Sewoong Oh, and Devavrat Shah. Efficient crowdsourcing for multi-class labeling. In *Special Interest Group on Measurement and Evaluation (SIGMETRICS)*, pages 81–92, 2013.

- [114] Juho Kim, Haoqi Zhang, Paul André, Lydia B. Chilton, Anant Bhardwaj, David Karger, Steven P. Dow, and Robert C. Miller. Cobi: Community-Informed Conference Scheduling. In *1st AAAI Conference on Human Computation and Crowdsourcing*, 2013.
- [115] Aniket Kittur, Ed H. Chi, and Bongwon Suh. Crowdsourcing user studies with Mechanical Turk. In *the ACM Conference on Human Factors in Computing Systems*, pages 453–456, 2008.
- [116] Aniket Kittur, Boris Smus, and Robert Kraut. CrowdForge: crowdsourcing complex work. In *The ACM Conference on Human Factors in Computing Systems, Extended Abstracts*, pages 1801–1806, 2011.
- [117] Shailesh Kochhar, Stefano Mazzocchi, and Praveen Paritosh. The anatomy of a large-scale human computation engine. In *Proceedings of the Conference on Human Computation and Crowdsourcing*, New York, NY, USA, 2010.
- [118] Anand Kulkarni, Philipp Gutheim, Prayag Narula, David Rolnitzky, Tapan S. Parikh, and Björn Hartmann. MobileWorks: Designing for Quality in a Managed Crowdsourcing Architecture. *IEEE Internet Computing*, 16(5):28–35, 2012.
- [119] Anand Pramod Kulkarni, Matthew Can, and Björn Hartmann. Collaboratively crowdsourcing workflows with turkomatic. *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pages 1003–1012, 2012.
- [120] Ray Kurzweil. *The Singularity Is Near: When Humans Transcend Biology*. Penguin (Non-Classics), 2006.
- [121] Walter S. Lasecki, Kyle I. Murray, Samuel White, Robert C. Miller, and Jeffrey P. Bigham. Real-time crowd control of existing interfaces. In *Proceedings of the Annual ACM Symposium on User Interface Software and Technology*, pages 23–32, 2011.
- [122] Edith Law and Luis von Ahn. *Human Computation*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2011.
- [123] David D. Lewis and William A. Gale. A sequential algorithm for training text classifiers. In *SIGIR Conference*, pages 3–12, 1994.
- [124] Christopher H. Lin, Mausam, and Daniel S. Weld. Crowdsourcing control: Moving beyond multiple choice. In *UAI*, pages 491–500, 2012.
- [125] Christopher H. Lin, Mausam, and Daniel S. Weld. Dynamically switching between synergistic workflows for crowdsourcing. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, 2012.

- [126] Greg Little, Lydia B. Chilton, Max Goldman, and Robert C. Miller. Turkkit: tools for iterative tasks on mechanical turk. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Workshop on Human Computation*, pages 29–30, 2009.
- [127] Qiang Liu, Jian Peng, and Alexander Ihler. Variational inference for crowdsourcing. In *Proceedings of the Annual Conference on Neural Information Processing Systems*, pages 701–709, 2012.
- [128] Xuan Liu, Meiyu Lu, Beng Chin Ooi, Yanyan Shen, Sai Wu, and Meihui Zhang. CDAS: A Crowdsourcing Data Analytics System. *Proceedings of the VLDB Endowment*, 5(10):1040–1051, 2012.
- [129] Iliia Lotosh, Tova Milo, and Slava Novgorodov. CrowdPlanr: Planning made easy with crowd. In *International Conference on Data Engineering*, pages 1344–1347, 2013.
- [130] Adam Marcus, David R. Karger, Samuel Madden, Rob Miller, and Sewoong Oh. Counting with the crowd. *Proceedings of the VLDB Endowment*, 6(2):109–120, 2012.
- [131] Adam Marcus, Eugene Wu, David R. Karger, Samuel Madden, and Robert C. Miller. Demonstration of quirk: a query processor for humanoperators. In *SIGMOD Conference*, pages 1315–1318, 2011.
- [132] Adam Marcus, Eugene Wu, David R. Karger, Samuel Madden, and Robert C. Miller. Human-powered sorts and joins. *Proceedings of the VLDB Endowment*, 5(1):13–24, 2011.
- [133] Adam Marcus, Eugene Wu, Samuel Madden, and Robert C. Miller. Crowdsourced databases: Query processing with people. In *Proceedings of the Conference on Innovative Data Systems Research*, pages 211–214, 2011.
- [134] Winter A. Mason and Duncan J. Watts. Financial incentives and the “performance of crowds”. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Workshop on Human Computation*, pages 77–85, 2009.
- [135] Geoffrey J. McLachlan and Thriyambakam Krishnan. *The EM Algorithm and Extensions (Wiley Series in Probability and Statistics)*. Wiley-Interscience, 2 edition, March 2008.
- [136] Paul Mineiro. Cost-Sensitive Binary Classification and Active Learning, 2012. <http://www.machinedlearnings.com/2012/01/cost-sensitive-binary-classification.html>.

- [137] Marti Motoyama, Kirill Levchenko, Chris Kanich, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. Re: Captchas-understanding captcha-solving services in an economic context. In *USENIX Security Symposium*, pages 435–462, 2010.
- [138] Barzan Mozafari, Purnamrita Sarkar, Michael J. Franklin, Michael I. Jordan, and Samuel Madden. Active learning for crowd-sourced databases. *arXiv preprint arXiv:1209.3686*, 2012.
- [139] Jon Noronha, Eric Hysen, Haoqi Zhang, and Krzysztof Z. Gajos. Platemate: crowdsourcing nutritional analysis from food photographs. In *Proceedings of the Annual ACM Symposium on User Interface Software and Technology*, pages 1–12, 2011.
- [140] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, 2008.
- [141] Aditya Parameswaran, Stephen Boyd, Hector Garcia-Molina, Ashish Gupta, Neoklis Polyzotis, and Jennifer Widom. Optimal crowd-powered rating and filtering algorithms. Technical report, Stanford University, 2013.
- [142] Aditya G. Parameswaran, Nilesh Dalvi, Hector Garcia-Molina, and Rajeev Rastogi. Optimal schemes for robust web extraction. *Proceedings of the VLDB Endowment*, 4(11):980–991, 2011.
- [143] Aditya G. Parameswaran, Hector Garcia-Molina, Hyunjung Park, Neoklis Polyzotis, Aditya Ramesh, and Jennifer Widom. Crowdscreen: algorithms for filtering data with humans. In *SIGMOD Conference*, pages 361–372, 2012. Online: <http://doi.acm.org/10.1145/2213836.2213878>.
- [144] Aditya G. Parameswaran, Hyunjung Park, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. Deco: declarative crowdsourcing. In *Proceedings of the ACM International Conference on Information and Knowledge Management*, pages 1203–1212, 2012.
- [145] Aditya G. Parameswaran and Neoklis Polyzotis. Answering Queries using Humans, Algorithms and Databases. In *Proceedings of the Conference on Innovative Data Systems Research*, pages 160–166, 2011.
- [146] Aditya G. Parameswaran, Anish Das Sarma, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. Human-assisted graph search: it’s okay to ask questions. *Proceedings of the VLDB Endowment*, 4(5):267–278, 2011.

- [147] Aditya G. Parameswaran, Ming Han Teh, Hector Garcia-Molina, and Jennifer Widom. DataSift: An Expressive and Accurate Crowd-Powered Search Toolkit. In *Proceedings of the 1st AAAI Conference on Human Computation and Crowdsourcing*, 2013.
- [148] Aditya G. Parameswaran, Ming Han Teh, Hector Garcia-Molina, and Jennifer Widom. DataSift: a crowd-powered search toolkit. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, pages 885–888, 2014.
- [149] Hyunjung Park, Richard Pang, Aditya G. Parameswaran, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. Deco: A system for declarative crowdsourcing. *Proceedings of the VLDB Endowment*, 5(12):1990–1993, 2012.
- [150] Hyunjung Park, Aditya Parameswaran, and Jennifer Widom. Query processing over crowdsourced data. Technical report, Stanford University, September 2012.
- [151] Hyunjung Park and Jennifer Widom. CrowdFill: collecting structured data from the crowd. *SIGMOD Conference*, pages 577–588, 2014.
- [152] Vassilis Polychronopoulos, Luca de Alfaro, James Davis, Hector Garcia-Molina, and Neoklis Polyzotis. Human-powered top-k lists. In *Proceedings of the 16th International Workshop on the Web and Databases*, pages 25–30, 2013.
- [153] Alexander J. Quinn and Benjamin B. Bederson. Human computation: a survey and taxonomy of a growing field. In *The ACM Conference on Human Factors in Computing Systems*, pages 1403–1412, 2011.
- [154] M. Jordan Raddick, Georgia Brace, Pamela L. Gay, Chris J. Lintott, Phil Murray, Kevin Schawinski, Alexander S. Szalay, and Jan Vandenberg. Galaxy zoo: Exploring the motivations of citizen science volunteers. September 2009. <http://arxiv.org/abs/0909.2925>.
- [155] A. Ramesh, A. Parameswaran, H. Garcia-Molina, and N. Polyzotis. Identifying reliable workers swiftly. Technical report, Stanford University, September 2012.
- [156] Vikas C. Raykar and Shipeng Yu. Eliminating spammers and ranking annotators for crowdsourced labeling tasks. *Journal of Machine Learning Research*, 13:491–518, 2012.

- [157] Vikas C. Raykar, Shipeng Yu, Linda H. Zhao, Anna K. Jerebko, Charles Florin, Gerardo Hermosillo Valadez, Luca Bogoni, and Linda Moy. Supervised learning from multiple experts: whom to trust when everyone lies a bit. In *Proceedings of the International Conference on Machine Learning*, page 112, 2009.
- [158] T. Rekatsinas, A. Deshpande, and A. Parameswaran. CrowdGather: Entity Extraction over Structured Domains. *ArXiv e-prints*, February 2015.
- [159] Daniela Retelny, Sébastien Robaszkiewicz, Alexandra To, Walter S. Lasecki, Jay Patel, Negar Rahmati, Tulsee Doshi, Melissa Valentine, and Michael S. Bernstein. Expert crowdsourcing with flash teams. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, pages 75–85, New York, NY, USA, 2014. ACM.
- [160] Flavio P. Ribeiro, Dinei A. F. Florêncio, and Vitor H. Nascimento. Crowdsourcing subjective image quality evaluation. In *Proceedings of the International Conference on Image Processing*, pages 3097–3100, 2011.
- [161] Joel Ross, Lilly C. Irani, M. Six Silberman, Andrew Zaldivar, and Bill Tomlinson. Who are the crowdworkers?: shifting demographics in mechanical turk. In *The ACM Conference on Human Factors in Computing Systems, Extended Abstracts*. ACM, 2010.
- [162] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.
- [163] Jeffrey M. Rzeszutarski and Aniket Kittur. Instrumenting the crowd: using implicit behavioral measures to predict task performance. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*. ACM Request Permissions, October 2011.
- [164] Jeffrey M. Rzeszutarski and Aniket Kittur. Crowdscape: interactively visualizing user behavior and output. In *Proceedings of the Annual ACM Symposium on User Interface Software and Technology*, pages 55–62, 2012.
- [165] Niloufar Salehi, Lilly C. Irani, and Michael S. Bernstein. We Are Dynamo: Overcoming Stalling and Friction in Collective Action for Crowd Workers. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 1621–1630. ACM, 2015.
- [166] Akash Das Sarma, Ayush Jain, Arnab Nandi, Aditya G. Parameswaran, and Jennifer Widom. Surpassing humans and computers with JELLYBEAN: crowd-vision-hybrid counting algorithms. Technical report, 2015.
- [167] Anish Das Sarma, Aditya Parameswaran, Hector Garcia-Molina, and Alon Halevy. Crowd-powered find algorithms. In *International Conference on Data Engineering*, 2014.

- [168] Lauren Schmidt. Crowdsourcing for human subjects research. *Proceedings of CrowdConf*, 2010.
- [169] Burr Settles. *Active Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.
- [170] H. Sebastian Seung, Manfred Opper, and Haim Sompolinsky. Query by committee. In *Proceedings of the Conference On Learning Theory*, pages 287–294, 1992.
- [171] Victor S. Sheng, Foster J. Provost, and Panagiotis G. Ipeirotis. Get another label? improving data quality and data mining using multiple, noisy labelers. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 614–622, 2008.
- [172] Rion Snow, Brendan O’Connor, Daniel Jurafsky, and Andrew Y. Ng. Cheap and Fast - But is it Good? Evaluating Non-Expert Annotations for Natural Language Tasks. In *Proceedings of the Conference on Empirical Methods on Natural Language Processing*, pages 254–263, 2008.
- [173] Michael Stonebraker, Daniel Bruckner, Ihab F Ilyas, George Beskales, Mitch Cherniack, Stanley B. Zdonik, Alexander Pagan, and Shan Xu. Data Curation at Scale: The Data Tamer System. *Proceedings of the Conference on Innovative Data Systems Research*, 2013.
- [174] Siddharth Suri, Daniel G. Goldstein, and Winter A. Mason. Honesty in an online labor market. In *Human Computation*, 2011.
- [175] Omer Tamuz, Ce Liu, Serge Belongie, Ohad Shamir, and Adam Kalai. Adaptively learning the crowd kernel. In *Proceedings of the International Conference on Machine Learning*, pages 673–680, 2011.
- [176] The Economist. The data deluge, February 2010.
- [177] Michael Toomim, Travis Kriplean, Claus Pörtner, and James A. Landay. Utility of human-computer interactions: toward a science of preference measurement. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pages 2275–2284, 2011.
- [178] Emma Tosch and Emery D. Berger. Surveyman: programming and automatically debugging surveys. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, pages 197–211, 2014.
- [179] Beth Trushkowsky, Tim Kraska, Michael J. Franklin, and Purnamrita Sarkar. Crowdsourced enumeration queries. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 673–684, 2013.
- [180] Petros Venetis and Hector Garcia-Molina. Dynamic max algorithms in crowdsourcing environments. Technical report, Stanford University, August 2012.

- [181] Petros Venetis, Hector Garcia-Molina, Kerui Huang, and Neoklis Polyzotis. Max algorithms in crowdsourcing environments. In *Proceedings of the 21st International World Wide Web Conference 2012*, pages 989–998, 2012.
- [182] Luis von Ahn. *Human computation*. PhD thesis, Pittsburgh, PA, USA, 2005.
- [183] Luis von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. CAPTCHA: Using Hard AI Problems for Security. In *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 294–311, 2003.
- [184] Luis von Ahn and Laura Dabbish. Labeling images with a computer game. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pages 319–326, 2004.
- [185] Luis von Ahn and Laura Dabbish. Designing games with a purpose. *Communications of the ACM*, 51(8):58–67, 2008.
- [186] Luis von Ahn, Ruoran Liu, and Manuel Blum. Peekaboom: a game for locating objects in images. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pages 55–64, 2006.
- [187] B. Walczak and D.L. Massart. Dealing with missing data: Part II. *Chemometrics and Intelligent Laboratory Systems*, 58(1):29 – 42, 2001.
- [188] Daisy Zhe Wang, Eirinaios Michelakis, Minos Garofalakis, and Joseph M. Hellerstein. BayesStore: managing large, uncertain data repositories with probabilistic graphical models. *Proceedings of the VLDB Endowment*, 1(1):340–351, 2008.
- [189] Jiannan Wang, Tim Kraska, Michael J. Franklin, and Jianhua Feng. Crowder: Crowdsourcing Entity Resolution. *Proceedings of the VLDB Endowment*, 2012.
- [190] Qinqin Wang, Patrick Cavanagh, and Marc Green. Familiarity and pop-out in visual search. *Perception & Psychophysics*, 56(5):495–500, 1994.
- [191] P. Welinder and P. Perona. Online crowdsourcing: rating annotators and obtaining cost-effective labels. In *CVPR Conference*, 2010.
- [192] Steven Euijong Whang, Peter Lofgren, and Hector Garcia-Molina. Question selection for crowd entity resolution. In *Proceedings of the VLDB Endowment*. VLDB Endowment, April 2013.
- [193] Jacob Whitehill, Paul Ruvolo, Tingfan Wu, Jacob Bergsma, and Javier R. Movellan. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *Proceedings of the Annual Conference on Neural Information Processing Systems*, pages 2035–2043. 2009.

- [194] Jennifer Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *Proceedings of the Conference on Innovative Data Systems Research*, pages 262–276, 2005.
- [195] Wikipedia. Citizen science — wikipedia, the free encyclopedia, 2013. [Online; accessed 22-July-2013].
- [196] Wikipedia. Unstructured data — Wikipedia, the free encyclopedia, 2013. [Online; accessed 6-July-2013].
- [197] Jeremy M. Wolfe and H. Pashler (Editor). *Attention*, chapter Visual Search, pages 13–73. University College London Press, 1998.
- [198] Bianca Zadrozny, John Langford, and Naoki Abe. Cost-sensitive learning by cost-proportionate example weighting. In *Proceedings of the 3rd IEEE International Conference on Data Mining*, Washington, DC, USA, 2003.
- [199] Omar Zaidan and Chris Callison-Burch. Feasibility of human-in-the-loop minimum error rate training. In *Proceedings of the Conference on Empirical Methods on Natural Language Processing*, pages 52–61, 2009.
- [200] Haoqi Zhang, Edith Law, Rob Miller, Krzysztof Gajos, David C. Parkes, and Eric Horvitz. Human computation tasks with global constraints. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pages 217–226, 2012.
- [201] Yuchen Zhang, Xi Chen, Dengyong Zhou, and Michael I. Jordan. Spectral Methods meet EM: A Provably Optimal Algorithm for Crowdsourcing. *arXiv preprint arXiv:1406.3824*, 2014.
- [202] Dengyong Zhou, Sumit Basu, Yi Mao, and John C. Platt. Learning from the wisdom of crowds by minimax entropy. In *Advances in Neural Information Processing Systems*, pages 2195–2203, 2012.
- [203] Dengyong Zhou, Qiang Liu, John C. Platt, and Christopher Meek. Aggregating Ordinal Labels from Crowds by Minimax Conditional Entropy. In *Proceedings of the 31st International Conference on Machine Learning*, June 2014.
- [204] Honglei Zhuang, Aditya G. Parameswaran, Dan Roth, and Jiawei Han. De-biasing crowdsourced batches. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, pages 1593–1602, 2015.
- [205] Paul C. Zikopoulos, Chris Eaton, Dirk deRoos, Thomas Deutsch, and George Lapis. *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*. McGraw-Hill Osborne Media, 1st edition, 2011.

- [206] Matthew Zook, Mark Graham, Taylor Shelton, and Sean Gorman. Volunteered geographic information and crowdsourcing disaster relief: a case study of the haitian earthquake. *World Medical & Health Policy*, 2(2):7–33, 2010.
- [207] Nathan Zukoff. Demographics of the Largest On-demand Workforce (Retrieved 16 March 2015), 2014. <http://www.crowdflower.com/blog/2014/01/demographics-of-the-largest-on-demand-workforce>.