Collecting Data With the Crowd

Beth Trushkowsky

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2014-208
http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-208.html

December 4, 2014

Collecting Data With the Crowd

by

Katherine E. Trushkowsky

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Michael J. Franklin, Chair
Professor Armando Fox
Professor Bjoern Hartmann
Professor Tapan Parikh

Fall 2014

Collecting Data With the Crowd

Abstract

Collecting Data With the Crowd

by

Katherine E. Trushkowsky

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Michael J. Franklin, Chair

Hybrid human/machine database and search systems promise to greatly expand the usefulness of query processing by incorporating human knowledge and experience via "crowdsourcing" into existing systems for data gathering and other tasks. Of course, such systems raise many implementation questions. For example, how can we reason about query result quality in a hybrid system? How can we best combine the benefits of machine computation and human computation? In this thesis we describe how we attacked these challenges by developing statistical tools that enable users and systems developers to reason about query completeness in hybrid database systems, as well as combining human and automated processing in search engines. We present evaluations of these techniques using experiments run on a popular crowdsourcing platform, Amazon's Mechanical Turk.

To the people who believed in me

Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my gratitude to the many people who supported me throughout my graduate school career, and without whom this thesis would not have been possible.

My advisors, Michael Franklin and Armando Fox, provided both academic and emotional support over the years, for which I am very grateful. Mike encouraged me to aim for deep and fundamental insight in my work, and his guidance was instrumental in tackling the question of reasoning about the open world in the context of human and machine computation. He also taught me that experiments that do not go as expected can be recast as exciting new challenges to explore. Over the years, Armando has been adept at asking the perfect probing questions to direct my work towards more fundamental and influential contributions. Our conversations were an invaluable asset in developing my research.

Of course, the work presented in this thesis was not accomplished alone. In addition to my advisors, I am very fortunate to have worked with my collaborators Tim Kraska and Purnamrita Sarkar in developing the ideas in this thesis. I would also like to thank the many other members of the AMP Lab who provided feedback and great conversations about this work. I additionally received valuable feedback from my thesis committee members Bjoern Hartmann and Tapan Parikh, including discussions about user interface design and other facets of working with the crowd that helped shape the direction of this thesis.

My friends and colleagues in the AMP Lab (and the RAD Lab) as well as the Berkeley Database group have made my experience in graduate school enjoyable and exciting. I am fortunate to have worked with so many compassionate and intelligent individuals. Kuang Chen, Neil Conway, Tyson Condie, amongst others, made studying for the preliminary exam an almost pleasurable experience. My many lab mates and fellow graduate students made day-to-day life and our biannual lab retreats fun and memorable. My experience in the AMP Lab was enjoyable also thanks to the amazing administrative efforts from Kattt Atchley, Sean McMahon, Boban Zarkovich, and Jon Kuroda.

I am very thankful for the people who believed in me and were there for me over the years. Michael Armbrust was a pleasure to work with as well as an inspirational and remarkable person. I am fortunate to have Kristal Curtis as a friend, whose thoughtful advice and conversations were a great resource for me. I enjoyed many intellectual and insightful discussions with Andy Konwinski, which provided inspiration and bolstered my confidence; I am extremely grateful to have such a compassionate person in my life. I want to thank Nicholas Dashman for his encouragement, support, and patience throughout my graduate school career. I would also like to thank my family for their love and faith in me.

My path both to and during graduate school was eased by the great mentorship and support provided by Jeffrey Forbes, Dan Garcia, and Sheila Humphreys. Jeff, my undergraduate advisor at Duke, initially got me involved in research and gave me invaluable guidance as well as the freedom to direct my own exploration. I am thankful that I had the opportunity during my time at Berkeley to participate in various diversity and outreach programs organized and facilitated by Dan and Sheila.

I am very grateful for the support I have received from both the National Science Foundation as well as the industry sponsors for the AMP Lab. In particular, I would like to acknowledge the support from a NSF graduate fellowship, a Google-UNCF scholarship, NSF CISE Expeditions Award CCF-1139158, LBNL Award 7076018, and DARPA XData Award FA8750-12-2-0331, and gifts from Amazon Web Services, Google, SAP, The Thomas and Stacey Siebel Foundation, Adobe, Apple, Inc., Bosch, C3Energy, Cisco, Cloudera, EMC, Ericsson, Facebook, GameOnTalis, Guavus, HP, Huawei, Intel, Microsoft, NetApp, Pivotal, Splunk, Virdata, VMware, and Yahoo!.

Chapter 1

Introduction

1.1 The Power of People

People can easily connect to one another on the internet, a property that has given rise to a number of applications in which people share their knowledge and experience with one another to tackle a wide array of tasks. Examples of such tasks include debugging software programs, analyzing satellite images, and collecting data—the focus of this thesis. The success of these applications demonstrates that people are both willing and capable of collaborating in this online setting. An example of this collaboration, well-known amongst software programmers, is the question-and-answer (Q&A) forum Stack Overflow[1], where people submit programming related questions to the community. A study shows that most questions actually receive helpful answers (over 90%), and those answers become available in a relatively short period of time (11-minute median) [2]. Another general Q&A website is Quora[2], which leverages a directed social graph to help direct users to meaningful content [3]. Of course, Wikipedia is another canonical example of collaborative efforts to collect and curate a vast body of knowledge using the "wisdom of the crowd".

Small units of effort aggregated from groups of people can be combined to help solve complex problems—problems that otherwise could be too difficult or tedious to overcome. Existing projects leverage human perception by using a crowd of people to analyze huge amounts of data such as images, e.g., classifying shapes of galaxies in Galaxy Zoo [4] or looking for clues in satellite images in the search for Jim Gray [5]. In the FoldIt game, players solve computationally difficult protein folding puzzles; the aim of the project is to expedite the scientific discovery process [6]. The ESP game from Luis Von Ahn, et. al. [7] is another example of a "game with a purpose"; competing players are shown images and try to guess keywords their opponents are typing; the more keywords they agree on, the more points they receive. The outcome of gameplay is a set of accurate, meaningful labels for images—data that is useful for many applications, including computer vision algorithms and image search.

Applications that involve people in the computational process is at the heart of research in

[1] www.stackoverflow.com

[2] www.quora.com

human computation and *crowdsourcing*. Von Ahn's Ph.D. dissertation, which describes "how to use human processing power in order to solve large-scale open problems in computer science" [8], caused rapid growth in modern interest in human computation research since its publication in 2005, as measured by appearance of the related terms "human computation" and "crowdsourcing" in computer science literature in 2005-2010, growing from fewer than ten to over 300 in that period [1][3]. A workshop dedicated to the topic of human computation began in 2009, which then became a full conference in 2013; participants hail from a wide range of computer science and human-centered disciplines[4].

1.2 Humans As a Resource: Opportunities and Challenges

This research has been accelerated by the availability of services for easily recruiting workers for human computation tasks. In particular, platforms such as Amazon's Mechanical Turk crowdsourcing marketplace (AMT) offer access to human workers via programmatic interfaces (APIs). These APIs provide an intriguing new opportunity, namely, to create hybrid human/computer systems. Such systems, could, to quote J.C.R. Licklider's famous 1960 prediction for man-computer symbiosis, "...process data in a way not approached by the information-handling machines we know today" [9]. As exemplified by the projects mentioned above, human knowledge, experience, and perception can be instrumental in solving complex tasks. With the use of crowdsourcing APIs, we can combine human computation with machine computation to create hybrid systems that can tackle problems the machine alone cannot.

Crowdsourcing marketplaces like AMT provide access to human workers as an elastic, pay-per-use resource: users in need of human computation post small units of work, called micro-tasks, which workers can complete in exchange for a small reward, typically a few cents. It may be tempting to view this crowd of workers as another type of elastic resource, such as infrastructure-as-a-service [10]. However, a group of human beings is very different from a cluster of compute nodes. For instance, in an early experiment using AMT, we asked the crowd to gather a list of all professors at UC Berkeley. The first task I posted simply asked a worker to provide the entire list, formatted as comma separated values; due to the time-consuming nature of this task, I did not expect any worker to attempt it. Much to my surprise, after many hours of work, several workers emailed me reporting on their progress and asking if there were particular sources they should be using to complete the task. I must say, not once have I gotten a frantic email from an EC2 instance[5].

Leveraging the power of human computation also means coping with human behaviors and characteristics. Unlike machines, people can experience fatigue or boredom as they are working on a task. Crowd workers may act maliciously, or even just unintelligently—behaviors that can impact the quality of their work. They may have very different backgrounds, experiences, and biases that impact the way they approach and respond to tasks. Developers of hybrid human/machine systems need to both acknowledge and manage these human behaviors in their algorithm design.

[3] See Figure 2 in [1]

[4] www.humancomputation.com

[5] A virtual machine in Amazon's Elastic Compute Cloud (EC2). See http://aws.amazon.com/ec2

1.3 Query Processing With the Crowd

Members of the database systems community have begun to explore the potential of hybrid human/machine systems for database query processing [11, 12, 13]. In these systems, human workers can perform query operations such as subjective comparisons, fuzzy matching for predicates and joins, entity resolution, etc. In an operator-based relational query engine, crowd processing can be encapsulated into operators that can be used along with traditional machine-based operators in query plans. A hybrid system can take advantage of existing database technology, like query optimization and data independence, while additionally leveraging human computation to answer queries the machine alone would have difficulty with.

An important class of queries are those that allow a user to collect a particular set of entities, or items. For example, the set of graduate students at UC Berkeley, or all the local dry cleaners open on Sundays. People are particularly adept at data gathering tasks because they have experience with finding and interpreting data on the web, as well as the ability to leverage their own knowledge about the world. The diverse backgrounds and experiences found in the crowd particularly benefit data collection because different individuals may be able to contribute answers that others may not have even known about.

1.4 Thesis Overview: Challenges and Contributions

In this thesis, I investigate crowd-based data collection in the context of the hybrid human/machine query processing systems mentioned above. For data collection, also called set enumeration queries throughout the thesis, the user issues a query to generate a set of result items.

Enlisting the help of the crowd to answer these types of queries raises many challenges, some of which are the focus of this thesis. These include:

Reasoning about query results In traditional relational database systems, the correctness of a query result is based on a fundamental assumption that states that the database has the complete set of information at query time; this notion is known the *closed-world assumption*. In a hybrid system, the crowd can be enlisted to gather more results as the query is running—violating this closed-world assumption. The addition of people into the query execution process forces us into the *open world*, raising an important question: how can we reason about query results in this context?

Cost versus quality One way to incorporate human computation is via paid crowdsourcing: individual workers are asked to complete a task in exchange for payment[6]. In the case of data collection queries, gathering more query results with the help of the crowd incurs more cost. An important aspect of the quality of a data collection query result is how complete it is; gathering more data can increase result quality in addition to cost. Thus it is necessary to reason about the

[6]Chapter 2 elaborates on types of crowdsourcing

cost versus quality tradeoff when developing crowd-based set enumeration strategies. The user interface (UI) presented to crowd workers can influence the cost and quality of data collection. For example, guiding workers to supply a diverse set of answers can both reduce payment for redundancy as well as yield a more complete result.

Combining humans and machines There is potential to further reduce crowdsourcing cost by taking advantage of automated processing when possible. Existing systems like search engines can be helpful for finding results in data collection tasks, and querying many of these resources takes little time and does not incur a cost to the user. The challenge is determining when additional human computation is required to provide a satisfactory result for the user, and how that human involvement should be leveraged to achieve a higher quality result at reasonable cost.

Human behaviors In addition to the power of human knowledge and perception, people also bring to the table characteristics and behaviors that can impact how they approach and complete tasks. As mentioned above, diverse backgrounds and experiences are advantageous for data gathering queries. People also may have different levels of cleverness and resourcefulness finding data. However, crowd workers may also have differing levels of commitment and quality: some people may contribute a lot versus a little, some may be adept at the task while others struggle. Tools developed for leveraging human computation may need to acknowledge this spectrum of the crowd's abilities.

This thesis describes how I, along with my collaborators [14, 15, 16], addressed these challenges. First, we adapt biostatistical techniques used for species estimation [17] to estimate query result set size as responses stream in from crowd workers. Using this estimate of result set size, we can understand query progress—providing the means to reason about the quality of the query result by measuring how complete it is. In our approach, we compensate for the particular behaviors people exhibit as they supply responses; we find the greatest detriment to estimator accuracy is the presence of overzealous workers who provide many more answers than others. We develop a technique to ameliorate the impact of these workers on the estimation algorithm, achieving a more accurate prediction of query result size.

With paid crowdsourcing, the cost of paying people to help with a data collection query is an important consideration. A cost versus quality tradeoff naturally emerges: spending more money for crowd work more yields more data, i.e., a more complete query result. In this thesis, we describe how to cut crowdsourcing costs in several ways. First, we present an approach to detect if the results for a data collection query are available on a single webpage by observing when workers provide responses in the same order; detection of this behavior enables the use of an alternate user interface (UI) that workers use to collect data, such as an application to simply scrape the webpage, for gathering data at lower cost. For cases in which scraping a web page will not suffice, we show that we can still reduce the cost of data collection queries with a different UI that intelligently reduces answer redundancy, and thereby overall cost, while retaining the statistical information necessary for the result set size estimation described above.

Finally, for some data collection queries we can reduce cost by leveraging existing automated processing from search engines, which incurs no cost to the user. Search engines can have difficulty processing certain parts of a query, which is where the addition of human computation is beneficial. However, there is no *a priori* knowledge of which aspects of a given query are better suited for human or machine processing. We develop an approach that first probes the automated system with multiple reformulations of the original query to generate sets of candidate results; we then select a subset of these candidates to send for crowd processing using an adaptive algorithm based on multi-armed-bandit techniques. We demonstrate that our technique yields more correct results given a budget for crowdsourcing than several baseline algorithms for an image search application.

In summary, the contributions of this thesis are:

- We develop statistical tools to estimate query progress and reason about result quality for data collection queries in the open world.
- We design and evaluate different user interfaces for gathering data with the crowd to reduce query cost.
- We develop an approach to combine human and machine processing for data collection queries using search engines.
- We examine the effectiveness of our techniques via experiments using a popular crowdsourcing platform, Amazon's Mechanical Turk (AMT).

1.5 Thesis Organization

The thesis is organized as follows. Chapter 2 contains background on human computation and crowdsourcing, including a discussion of the dimensions that differentiate crowdsourcing platforms, services, and approaches. We also provide a detailed description of Amazon's Mechanical Turk (AMT), the platform used for the experimental results described in this thesis. We give an overview of techniques and algorithms for managing quality in human computation tasks, as well as examples of existing hybrid human/machine projects. Chapter 3 focuses on hybrid database systems; we give an overview of traditional relational database systems before describing the architectures and human computation extensions for existing human/database systems.

In Chapter 4, we investigate data collection queries in which crowd workers help gather, or enumerate, a set of items one at a time in response to a user query. With the crowd-provided data, natural questions arise about the quality of the set; namely, how do we know when the set is complete? Should we pay to get more answers? We develop a way to reason about result quality by adapting statistical techniques to estimate the progress of such an enumeration query.

Chapter 5 describes alternate worker task interfaces for these types of queries. We explore reducing the crowdsourcing cost to enumerate a set of items by restricting answer redundancy using an interface called "Negative Suggest". We also describe an algorithm for detecting when workers may be consulting lists of items on the web, potentially indicating when the set could be gathered using a data scraping tool.

Chapter 6 looks at combining the efforts of human workers with existing search engines for data gathering tasks. A user may specify a data collection query with a set of constraints, or predicates, that each item in the set must satisfy. Some of these predicates may be well-suited for the search engine, while others are better left for human computation. However, given an *ad hoc* user query, we do not know *a priori* which predicates will be troublesome for the search engine. We develop an algorithm based on multi-armed-bandit (MAB) techniques to combine humans and machines to respond to these queries.

Finally, in Chapter 7 we describe a plan for future work towards the development of an optimizer for hybrid human/machine query processing systems, and in Chapter 8 we conclude the thesis.

Chapter 2

Background: Human Computation

In order to understand the advantages and challenges of collecting data with the help of human workers, it is important to first understand the basics of crowdsourcing, as well as how hybrid human/machine systems incorporate the crowd into existing software systems. This chapter provides overall background; chapter-specific discussions of related work will also appear throughout the thesis.

In the first few sections, I define and provide an overview of human computation. This includes a discussion of the different types of crowdsourcing and available platforms. I describe in detail the specifics of Amazon's Mechanical Turk (AMT), a popular crowdsourcing service for paid human computation tasks and the platform used for the experimental results in this thesis.

The latter part of this chapter delves into the use of human computation. I discuss ways that humans have been incorporated into software applications, enhancing the capability of machine computation with human knowledge, perception, and experience. However, while humans have the potential to help solve complex problems, there are also challenges in managing the crowd. I describe common techniques and algorithms for *quality control*, an important consideration in crowdsourcing applications.

2.1 Dimensions of Human Computation and Crowdsourcing

Many definitions of *human computation* and *crowdsourcing* exist in the literature. In the most general sense, these terms refer to the notion of using people to accomplish tasks, typically tasks that are complex. Leveraging human knowledge and skills in this manner is not a new concept. For example, the creation of the Oxford English Dictionary in the late nineteenth century has been described as an early crowdsourcing project: over decades, hundreds of thousands of people contributed definitions of every word in the English language to form this dictionary [18]. The Mathematical Tables Project, which began in 1938, employed 450 unskilled people to calculate tables for mathematical functions such as exponential functions and logarithms [19].

More recently, a myriad of human computation projects and systems have been developed to tackle a wide array of challenges. Several existing surveys have looked into different ways to

characterize this space, including [20, 1, 21]. In this section, I briefly describe several aspects of these taxonomies, focusing on the dimensions that are most relevant to understanding the type of crowdsourcing used for the work in this thesis. These dimensions touch on *how* humans provide work, *why* they provide work, and what is the *goal* or output of the human computation.

2.1.1 Implicit vs. Explicit Crowdsourcing

People can *explicitly* provide data in response to a given question or task or contribute data *implicitly* as a byproduct of their behavior. This distinction is called the *nature of collaboration* in [21] and the *participatory culture* in [20]. With implicit crowdsourcing, data is gathered by making observations of how people interact with their environment or use particular tools and applications. For example, monitoring the queries issued to search engines can provide insight into global events and trends; researchers at Google have shown that mining search terms can be very effective at predicting flu outbreaks[1]. The ubiquity of mobile devices creates ripe opportunity for participatory sensing[22], in which people can implicitly contribute observations regarding their local environment through various sensors such as cameras and accelerometers. This data can then be aggregated and used to benefit the crowd itself. Traffic monitoring and congestion prediction is possible using data from the Global Positioning System (GPS)[2]. Another example is the Carat project [23], which helps diagnose battery issues on cellular phones by aggregating usage statistics from phones of people who have installed the application.

Implicit crowdsourcing yields data as a byproduct of human behaviors and sensing devices. In contrast, with explicit crowdsourcing, people directly and consciously address a question or task. The size and complexity of the task can vary. Examples of small tasks include asking people to verify an address or provide keyword labels for images, while more involved tasks include designing logos[3] or programming[4]. A task could ask workers directly to contribute observational data, similar in nature to the data garnered through implicit crowdsourcing. For example, the eBird citizen science project solicits local species observations from bird watchers in order to gain a global perspective of bird distribution [24]. Explicit crowdsourcing also includes collaborative composition and editing activities, Wikipedia being the canonical example.

Naturally, data collection can occur using either type of collaboration. In this thesis I leverage explicit crowdsourcing, i.e., asking people to participate in the data gathering process.

2.1.2 Motivation/Incentives

A separate notion from how people contribute computation is their motivation for doing so. Quinn and Bederson [1] outline the following incentive categories: pay, enjoyment, altruism, reputation, and implicit. A straightforward way to compensate people for an arbitrary task is to offer a reward upon completion, e.g., money or virtual goods. General crowdsourcing marketplaces such as AMT

[1] Google flu trends http://www.google.org/flutrends

[2] traffic.berkeley.edu

[3] e.g., 99designs.com

[4] e.g., TopCoder.com

use this model. However, reward-based crowdsourcing can have unintended effects such as incentivizing people to put forth minimal effort or trying to game the task in order to get more rewards more quickly; I discuss quality control issues and techniques in more detail in Section 2.3.

Instead of a reward, "games with a purpose" use enjoyment or entertainment to solicit human computation. For example, to generate relevant keyword labels for images, the ESP game[7] presents an image to two opponents and challenges them to guess what the other is typing; if the players type the same string, points are awarded—and, as a sub-effect, that string is verified as relevant keyword. One of the opponents may even be computer algorithm, and thus the human player is verifying the output of the algorithm. Other games include the protein-folding puzzle game FoldIt[6] and the language learning tool DuoLingo[5] that translates text on the web. People can also participate because they want to contribute to a cause (altruism), such as the search for Jim Gray mentioned in the Introduction, or because they desire the prestige associated with their contributions (reputation).

Some applications require the completion of some task or an agreement to send observational data in order for the user to gain access to a service, fitting into Quinn and Bederson's "implicit" motivation category. For example, using the "My location" feature with Google maps on a mobile device to identify the user's current location allows the maps application to use the device's GPS information for aggregate statistics. Many web services verify that a user is indeed human by requiring him/her to transcribe text shown in an image; the reCAPTCHA[6] project capitalizes on this task by having the user additionally transcribe text that an OCR algorithm was unable to deciper.

2.1.3 Task Complexity and Goals

Crowdsourcing applications differ in the role that workers play; tasks can range in size, complexity, and difficulty. Doan et al. [21] describe various roles of human users and the degree of manual effort. For example, they use the term *slaves* to denote workers who are used as a resource in a divide-and-conquer fashion for tasks such as looking for clues in many satellite images. A role with more effort required of the worker would be a *content provider*, e.g., someone who creates and revises articles on Wikipedia. In [20], Doan et al. describe a similar characterization of human effort from the perspective of the task rather than the worker. They outline a task granularity spectrum, ranging from small *micro-tasks* such as image labeling, to *complex tasks* such as building a website.

Related to the discussion of *what* workers are doing is the question of what expertise or background knowledge is required of workers to complete a given task. Quinn and Bederson [1] describe this as the *human skill* dimension. At one end of the spectrum, a crowdsourcing task may only require fundamental human skills that all people possess, such as identifying well-known objects in images. Other tasks may require particular skills or knowledge, such as fluency in two languages for a translation task.

[5] www.duolingo.com

[6] www.google.com/recaptcha

In this thesis, individual workers complete small units of work for data gathering tasks that do not require special expertise, however I touch on the challenge of domain knowledge for data collection in Chapter 3.

2.1.4 Quinn and Bederson Classification of Human Computation

In addition to their taxonomy of human computation systems, of which I discussed several dimensions above, Quinn and Bederson [1] also provide a classification of human computation versus related ideas such as crowdsourcing. For reference, I reproduce this classification in Figure 2.1.

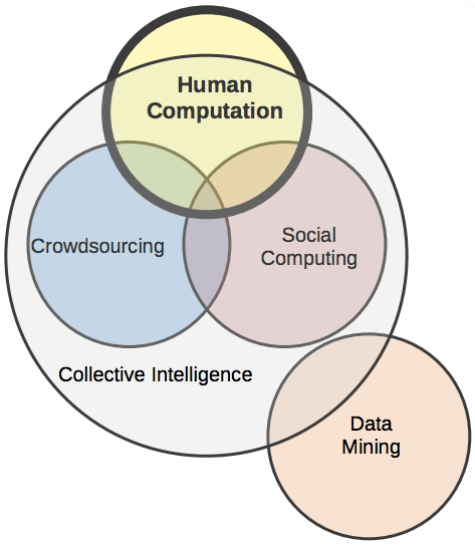


Figure 2.1: Quinn and Bederson's classification of human computations systems (Figure 1 in [1])

.

Notably, they distinguish between the concepts human computation, crowdsourcing, social computing, and collective intelligence. The authors argue that human computation must be directed by a computational process; thus a collaborative online encyclopedia is not human computation because its activities are dictated by the human participants. Instead, these *social computing* applications simply use technology to mediate communication. The term crowdsourcing is reserved for instances when a particular job or role typically done by an individual or algorithm is replaced by the work of a large group of people.

In this thesis, I use the terms human computation and crowdsourcing interchangably. Quinn and Bederson describe the intersection of these areas in Figure 2.1 as "applications that could reasonably be considered as replacements for either traditional human roles or computer roles" [1]. The crowd-based data collection work described in this thesis fits into this intersection. The example applications mentioned in this section fall into various areas of collective intelligence, which is generally the notion that groups of people can achieve great things. While an in-depth

description of the components of Figure 2.1 is out of scope of this thesis, the interested reader is encouraged to consult Quinn and Bederson's survey in [1].

2.1.5 Marketplaces for Explicit Crowdsourcing

There are existing platforms for connecting users with crowd workers for explicit, for-pay human computation; I provide here a brief comparison of several platforms. My aim is not to provide an exhaustive list, but rather to highlight some of the key differences of the popular ones. The subsequent section contains an in-depth description of Amazon's Mechanical Turk (AMT), used for the experimental results throughout this thesis.

Platforms differ in the amount of work, i.e., the size of the task, that is typically asked of people. Labor marketplaces such as AMT have *microtasks*, a small unit of work that usually takes only minutes or less to complete. Larger tasks or projects are the aim of crowdsourced freelance services such as ODesk[7], e.g., designing a website or creating a mobile application. Who chooses which work to do—the crowd worker or the user requesting to have work done—also differs by platform; on AMT, workers decide which microtasks to complete, while other services such as MobileWorks[8] assign tasks to workers. The payment model may be per-task or an hourly wage.

The nature of the relationship between the workers and the crowdsourcing platform impacts how the quality of work is managed. Closer relationships entail extensive worker training and monitoring for quality; techniques for achieving higher quality work, such as those discussed in Section 2.3 of this chapter, are built into the service. Other platforms such as AMT provide users access to workers' performance statistics, which the users can leverage to build custom algorithms and techniques for managing quality. For example, AMT provides information for each worker regarding how many tasks they have worked on have rejected by users, which users can use to identify malicious workers. For some specific tasks, AMT workers who have consistently done well can earn a "master'" label, and users can restrict their work only to these individuals. The platform also provides mechanisms in its API for users to train workers for tasks they develop.

2.1.6 Section Summary

Crowdsourcing applications can be categorized by how they solicit human computation, the different motivations people have for responding to the solicitations, as well as the goals of the tasks. In this discussion, I do not distinguish between human computation and crowdsourcing, and in the remainder of this thesis the terms appear interchangeably throughout. For the majority of the data collection work done in this thesis, I leverage the AMT crowdsourcing platform for explicit, reward-based human computation micro-tasks.

[7] www.odesk.com
[8] www.mobileworks.com

2.2 Amazon's Mechanical Turk

As described in the preceding section, Amazon's Mechanical Turk (AMT) is a popular micro-task crowdsourcing marketplace, and the platform used for experiments described in this thesis. AMT exposes a simple application programming interface (API) for allowing users to ask arbitrary questions to crowd workers in exchange for payment. In this sense, AMT is reminiscent of elastic resources such as infrastructure as a service (IaaS): it is a lower-level service that offers developers the flexibility to incorporate crowdsourcing inside other applications. In this section, I provide an in-depth description of the relevant AMT commands and their parameters, as well as what the marketplace looks like for both the crowd worker and the user who is requesting work be done.

2.2.1 Relevant Terminology and API

AMT is a labor marketplace for micro-tasks: small units of work that typically take on the order of seconds or minutes to complete for a reward on the order of cents; micro-tasks do not require specialized knowledge, training, or expertise. Users who want work done are called *requesters*. Requesters only need to establish an AMT account with a credit card. The crowd consists of workers, also called "turkers", who choose which micro-tasks they want to work on. While the demographic of workers used to be primarily based in the United States, there has been a shift in recent years towards international workers, largely from India [25]. These demographics can influence both the interpretation of a task, as well as the particular experiences, knowledge, and biases workers may have as they complete a task; we discuss the impact of some of these differences in the next chapter.

AMT has a particular terminology for describing the micro-task work done on their platform, which we also use in explaining the experimental setups throughout the thesis:

- A *human intelligence task*, commonly called a *HIT*, is the smallest unit of work that a worker can do and be paid for. After completing a HIT, the results of that task are available for the requester to review and ultimately approve and pay for (or reject and not pay). For example, a HIT could ask a worker to extract the products bought given a picture of a shopping receipt.

- The requester may have a group of similar HITs, e.g., several different receipts to parse, which the platform automatically groups together in a *HIT group*; workers may choose to do as many HITs in a HIT group as they desire.

- If a requester wants multiple workers to complete the same HIT, he/she can create multiple *assignments* for that HIT; individual workers can only do one assignment for a given HIT and cannot tell how many total assignments there are for a given HIT. This feature can be used, for example, to ask several workers to parse the same receipt to compensate for any errors an individual worker might make.

As mentioned above, one of the strengths of the AMT platform is its programmability via a simple API. Requesters can post HITs, retrieve worker responses, and approve or reject those re-

sponses using a Representational State Transfer (REST) API. Relevant methods include `createHIT()`, `getAssignmentsforHIT()`, `approveAssignment()`, and `rejectAssignment()`.

- `createHIT()` posts a new task in the AMT marketplace with a particular title and description, as well as fields specifying (amongst others) the reward, number of assignments, amount of time a worker is allowed to work on the task, and when the task should expire. HITs that are created with matching values in these fields are automatically grouped into a HIT group. An important argument to this method is the *question* which describes what the task actually is; we describe the creation of questions below. This call returns a *HITId* that the requester (or his/her application) can use to inquire into the status of results for the task.

- `getAssignmentsforHIT()` returns all responses for a given HIT. Notably, it is also possible to extract a *workerId* for each worker who completed the HIT that can be used to (anonymously) identify individual workers across HITs.

- `approveAssignment()` and `rejectAssignment()` are the actions that the requester can take after viewing a response to a particular HIT. If the response is satisfactory, the requester can approve the HIT and the worker will be paid the reward associated with that HIT. If the requester rejects the assignment, the worker is not given the reward.

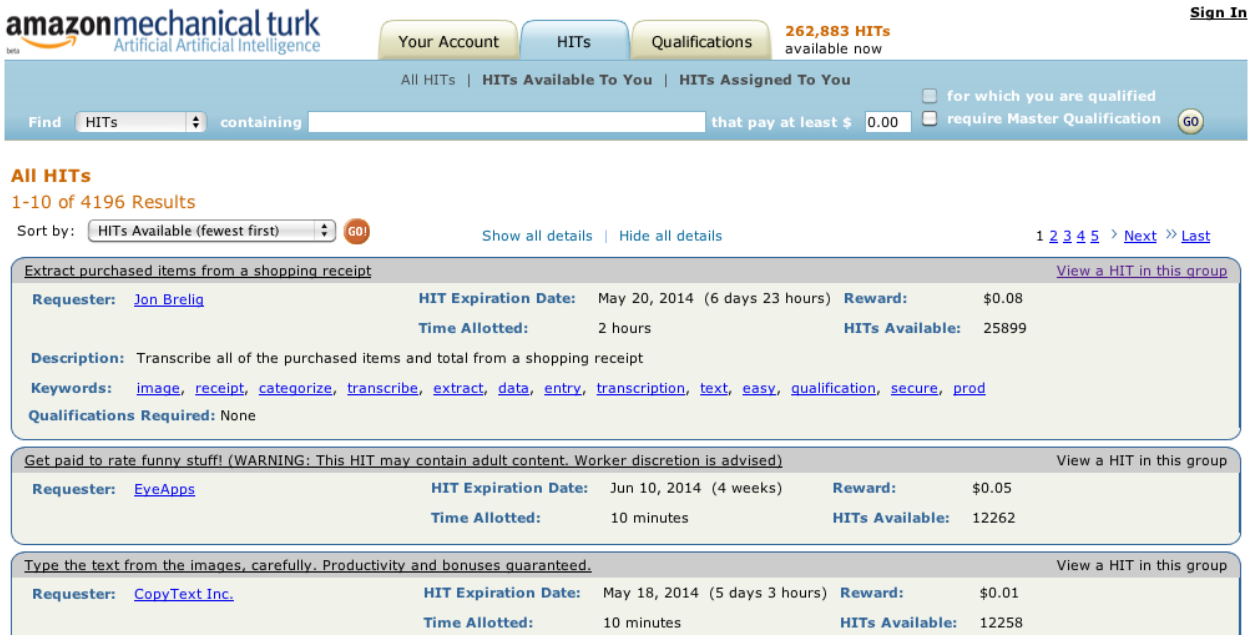


Figure 2.2: Mechanical Turk web interface for workers to select which HITs, or tasks, to work on.

2.2.2 Example Crowdsourcing Workflow on AMT

Suppose the user has a question he/she wants to ask the crowd, for example to categorize a set of pictures as having a cat in them or not. In this example, each picture will be used to create one HIT. The number of assignments per HIT determines the maximum number of distinct workers who may assess each picture; this requester may choose to create three assignments for each picture. The next step for the user is to design the question: the actual task interface that the workers will interact with. The question is specified in a markup language such as XML or HTML and is sent as a parameter to the `createHIT()` function call. One way that AMT provides for question design is to use a *QuestionForm*, an XML-based description that AMT will use to render an HTML task for workers to view in their web browser. Alternatively, the requester could use an *HTMLQuestion* or *ExternalQuestion* in which he/she can write the exact HTML that will be shown to workers, which will shown in an iFrame within the task UI created by the AMT platform. When using an *ExternalQuestion*, the HTML for the task is hosted on the requestor's own web server, which provides a way to incorporate server-side logic, e.g., to pull content from a database system. After creating the question, the requester can then issue `createHIT()` calls to AMT to create HITs for each picture of a cat.

Workers view available HITs in their web browser; Figure 2.2 shows an example screenshot of this view. Based on reading a HIT's description, and optionally looking at a preview of one of the HITs in the HIT group, workers decide which tasks they want to complete. After assignments have been submitted, the requester can view the responses from the workers and take subsequent actions. For the example task of categorizing pictures as having cats or not, the requester may write code that retrieves the answers for each picture via `getAssignmentsforHIT()` and calculates a majority vote to determine which pictures contain cats. The requester can also approve or reject the submitted assignments. If no action is taken, the assignments are automatically approved after a pre-determined delay.

When a worker is looking at available HITs, with the interface shown in Figure 2.2, several factors may influence which tasks he/she chooses to work on. The worker can quickly view HITs' titles and descriptions to decide if the task looks, for example, interesting, entertaining, or easy. AMT allows workers to preview what a HIT will look like by clicking "View a HIT in this group"; the preview shows the same HTML for the task that the worker will see if he/she chooses to work on it. After looking at the preview, the worker typically will then decide if the task is worth spending time on. The worker may consider if the offered reward matches the necessarily level of effort and if the task has straightforward instructions—a straightforward task increases the likelihood that the worker will complete it correctly and subsequently be paid. Another aspect of the the decision process is how many HITs there are in the HIT group, as it is desirable to amortize the cost of learning a task's instructions by completing a large number of those tasks.

The screen shot in Figure 2.2 also shows that workers can change how the listing of available HITs is sorted. The most popular sort choices are by number of HITs in the group and by time the HIT group was posted [26]. The former allows workers to complete many of the same type of task, while the latter provides a refreshed look at new tasks to work on.

Considering the worker's perspective, there are a number of considerations for the requester as

he/she designs a crowdsourcing task for the AMT marketplace. Because workers choose the tasks they want to work on, the requester wants to create a task that is appealing to workers. How to choose the best reward size for a given task is an open issue and, perhaps contrary to intuition, a higher reward may yield a faster response time from workers but not necessarily yield higher quality work [27]. As mentioned above, a task's instructions should be clear. Of particular importance is conveying exactly what responses or behavior warrant a rejection of the work done on the task. Seemingly unfair practices may start a conversation on a worker forum such as TurkerNation.com; a discussion about workers' bad experience with a particular requester could cause other workers to avoid completing work for that requester. The requester also needs to consider what types of quality control are appropriate for their task. Quality control, a topic all on its own, is discussed in the following section.

2.3 Quality Control

Explicit human computation can be extremely powerful, often helping to solve complex issues that algorithms alone cannot. However, leveraging human input can also lead to issues and errors that are unique to human beings. The source of these errors could lie with the crowd workers: they may intentionally produce incorrect or poor quality work due to laziness or because they are trying to game the system, or even unintentionally, due to confusion or personal issues such as fatigue or intoxication. On the other hand, the source of error could lie with the task itself if the question is ambiguous, subjective, or open to interpretation.

In this section, we give an overview of work related to controlling quality from crowdsourced work. Depending on the type of crowd platform, as described above, quality control techniques may or may not be incorporated within the crowdsourcing service.

2.3.1 Strength in Numbers: Majority Vote and Related Techniques

A standard quality control technique is the majority vote, typically employed for simple tasks. As the name indicates, for a given question that requires a human response, the requestor asks n crowd workers this same question; the answer that is provided by a majority of the n responses wins. In essence, this technique uses redundancy to reduce the impact of error from a few workers. Majority vote can be applied to both objective tasks, i.e., questions that have a single true answer, as well as subjective tasks that are opinion-based or open to interpretation. For example, an objective task might ask the worker to provide the name of a celebrity when shown a picture. A subjective task could ask which of several blurbs best describes a product.

For questions with a single true answer, the use of gold standard data can help determine whether a worker is providing incorrect answers (maliciously or otherwise). In the gold standard technique, some of the questions posed to workers have known answers. If a worker answers multiple of these questions incorrectly, this is an indication that his/her work is of low quality. The worker's responses may be simply discarded or, as is done by the crowd service CrowdFlower[9],

[9] www.crowdflower.com

these incorrect responses can be used to guide or train workers to provide better answers. Of course, the limitation of the gold standard approach is that the gold data needs to be generated. CrowdFlower discusses a strategy for scaling up this process with programmatic gold data creation [28].

More complex techniques involve developing a model of worker error for a particular task. Such a model can be useful to estimate how many crowd workers should be asked to answer a question to reach a quality threshold, or if asking for more input would have little impact on the result (i.e., diminishing returns). For example, in the Turkontrol project [29, 30], Dai et al. quantify the improvement of an artifact that is processed with the crowd via the TurkIt [31] iterative improvement workflow application. They learn models for a worker's probability of improving a given artifact, as well as his/her likelihood of identifying if the improved artifact is actually better (called a ballot task). The models are used to dynamically decide how many ballot tasks are necessary at a given phase of the iterative improvement. Experimental results with tasks writing image descriptions show that choosing the number of ballot tasks adaptively, versus a policy that uses a fixed number, costs less to achieve the same quality because extra ballot tasks are not wasted when the artifact's improvement is not contentious. Ipeirotis et al. [32] adapt an expectation-maximization (EM) algorithm that learns worker error rates for labeling tasks, while also compensating for their individual biases. They present experimental results classifying webpages with adult content, showing a 30% reduction in labeling cost and an increase in classification accuracy from 0.95 to 0.998, compared to the algorithm that does not consider individual biases.

2.3.2 Crowd Programming Patterns and Workflows

A key paradigm in explicit crowdsourcing is divide-and-conquer: breaking large tasks into smaller chunks for the crowd to work on. There has been significant work regarding how to perform this decomposition to achieve higher quality output.

In Soylent [33], the crowd is invoked to aid a user with word processing revisions from within Microsoft Word, including shortening or proofreading a selection of text. A major impact of this work was the introduction of the "find-fix-verify" (FFV) programming pattern for tackling worker quality issues. The FFV pattern is a way to deal with both "eager beaver" and "lazy turkers", i.e., workers who put in so much effort as to distort the original artifact or do the minimum amount of work to get paid, respectively. The general idea behind FFV is to divide the document revision task into a sequence of sub-tasks: first workers find errors in the document, then (different) workers provide corrections for those errors, and finally other workers verify those corrections. For example, first identifying a segment of text that could be shortened, then once those segments are agreed upon, actually performing the shortening; incorrect fixes are removed during the verification stage.

The map-reduce programming paradigm has also been an inspiration for crowdsourcing workflows. In the CrowdForge project [34], there are three types of tasks: tasks to break down a larger task into smaller ones ("partition"), tasks in which work is done by one or more workers ("map"), and tasks where workers' efforts are combined into a single output ("reduce"). For example, to write an encyclopedia article, the first step is to partition the work by creating an outline. Then

workers submit a single fact about their assigned topic. Finally, in the reduce phase, the facts are merged to form paragraphs. The requester can create complex flows with nested subtasks that can run in parallel. The system manages the flow of data and dependencies between tasks.

The Jabberwocky programming environment [35] is a social computing stack that includes a parallel framework "ManReduce". The programmer/requester can specify map and reduce tasks, as in CrowdForge, and either task type can be processed by a human or machine. The tasks meant for humans can be parameterized to designate the workers' required skills or characteristics needed for the task. On top of ManReduce is a custom programming language called Dog, which is similar to Pig [36] and Sawzall [37]; Dog programs are compiled down to ManReduce workflows.

Kulkarni et al. [38] investigated allowing the crowd itself to decompose a complex task into smaller, more manageable subtasks. They propose a "Price-Divide-Solve" pattern in which a task is divided if its current price is not fair for the amount of work required; after subtasks are solved, their results are combined in a separate merge stage. Experimental results show that completely unsupervised task decomposition is only sometimes successful, but access to more knowledgable workers and/or requester intervention increased success.

2.4 Human/Machine Symbiosis

Crowdsourcing platforms with a programmatic API for interacting with the crowd, such as AMT, create an opportunity to integrate human computation within software systems as well as to develop hybrid human/machine algorithms. In the former case, these APIs are used to enhance the functionality of existing applications with new features powered by the crowd. The end user is typically isolated from the management of crowdsourcing, and may not even realize that part of the application is powered by people. In the latter case, developers create algorithms or processing pipelines that use both machine computation and human computation to address a problem. For example, the crowd could be used to pre-process and/or post-process data from an automated system. In this section, we briefly highlight several influential example applications that illustrate both the power and the challenges of these classes of human/machine symbiosis.

2.4.1 Crowds Inside Software

Little et al. [31] developed the TurkIt project, which integrates the use of human computation within a normal imperative programming environment, using the AMT API. Human computation algorithms are encapsulated into functions, such as performing a majority vote or a crowd-based sort. To prevent the application from re-executing expensive human computation procedures in the face of software crashes or thrown exceptions, users can demarcate sections of the program that contain costly operations. After these operations run, their state is stored in a database and they are not executed again if the program is re-run.

The TurkIt project aims to support developers of human/machine applications. An example of an application that leverages TurkIt is Soylent [33], also mentioned in Section 2.3.2. Soylent enhances word processing software with several crowd-based extensions: "Shortn", "Crowdproof",

and the "Human macro". The shortn feature takes a selection of text and uses the crowd to produce an shorter snippet of text without changing the meaning. Crowdproof is used to provide crowd-based spelling and grammar checking. The human macro allows users to specify an arbitrary task for the crowd, such as properly formatting a citation. Users working on a document in Microsoft Word can right-click on a selection of text and choose to invoke any of these extensions directly. To get quality answers from the crowd, Soylent uses the find-fix-verify paradigm described earlier in Section 2.3.2.

Another human/machine application is VizWiz [39], a mobile application that helps the blind make sense of their environment with the help of the crowd. Users take a picture using their mobile phone and record a question (e.g., "which door is the women's restroom?"), and the application then creates a task on AMT with the picture and question. Using a low-latency crowdsourcing technique they call "quikTurkit", users are able to get real-time responses to their questions.

2.4.2 Hybrid Algorithms

Hybrid algorithms combine the processing power of a machine-based approach with human input. One example application is entity resolution (ER)[10], an aspect of data cleaning, which aims to determine if a given pair of items refers to the same real-world entity. While many automated ER algorithms exist (see [40] for a survey), there is still much room for accuracy improvement. Active learning approaches incorporate human input by choosing a sample of item pairs that humans label as matching or not matching, and then using those labels to better train the classifier. Sarawagi et al. [41] developed an active learning approach for choosing a set of challenging pairs of items for human verification; they show that their method reduces the training set size needed to achieve high accuracy.

Jeffery et al. [42] look at entity resolution in the context of pay-as-you-go data integration systems. They propose a decision-theoretic strategy influenced by the frequency with which items appear in the query workload to determine the order that item pairs are chosen for human evaluation. More recently, Wang et al. propose CrowdER [43], an approach for scaling a hybrid ER technique that leverages a paid crowdsourcing platform such as AMT. They first process the data with a machine-only algorithm to determine the likelihood that item pairs refer to the same entity; those pairs with likelihood above a given threshold are sent to the crowd for verification. To make the crowd-based component financially feasible, they batch multiple items in a single task (recall workers are paid per task). While determining the optimal batch size is NP-hard, they propose an approximation algorithm to choose the number of items in each task.

Humans have also been combined with information retrieval systems to provide enhanced search functionality. For example, Bernstein, et. al. [44] leverage crowdsourcing to provide inline answers directly in the search results page for uncommon queries; these direct answers have only been provided for a few of the most common query types, such as weather inquiries. After using search query log analysis to identify candidate web pages for sets of queries with succinct information needs, the system uses the crowd to extract, proofread, and format the answers that

[10]Entity resolution is also known as deduplication and record linkage, amongst other aliases.

will appear directly in the search results page. Similar to VizWiz, the Crowdsearch [45] mobile application helps people learn about their surrounding environment. Users take a picture using their mobile phone of, say, a building or landmark that they want to identify. The picture is first used to query an image search system for possible matches, and then the crowd is invoked to provide the final decision.

Another important class of hybrid systems is human/machine database systems, Traditional database systems can have difficulty processing certain queries, such as those requiring subjective comparisons or fuzzy matching for predicates—tasks well-suited for human computation. The level of abstraction built into relational systems facilitates the addition of new algorithms and techniques that incorporate the power of the crowd into existing systems. This topic is the focus of the next chapter and the context for the work in this thesis.

Chapter 3

Hybrid Human/Machine Database Systems

3.1 Introduction

A number of existing systems leverage the programmatic API provided by platforms such as AMT, creating hybrid human/machine systems. A particular class of these systems is hybrid human/machine database systems. Hybrid human/machine database systems are the context for the work described in this thesis; this chapter provides relevant background and motivation. The goal of these systems is to incorporate human computation in order to answer queries that the database system alone traditionally has difficulty with; for instance, queries that require human perception or knowledge about the world. Relational database systems are particularly amenable to the addition of new and modified crowd-based algorithms and techniques due to the substantial level of abstraction that is built in. In particular, these systems provide *data independence*, which allows them to be robust to changes in logical organization of the data as well as physical data layout. For example, a query that asks to sort data does not need to know whether the origin of that data is local storage or the crowd. Furthermore, the implementation of the sorting algorithm itself could be replaced with one involving human computation.

In this chapter, I describe the architectures and design choices of several hybrid database systems that aim to extend the functionality of relational database systems. The discussion is focused on query processing, and highlights how incorporating human workers in the system changes traditional query execution. Of course, these changes present many opportunities as well as challenges.

To provide necessary context and background for this discussion, we first provide an overview of relevant aspects of traditional relational database systems (RDBMS). The second subsection provides a detailed comparison of the architectures, query languages, and query processing techniques offered by existing hybrid systems. In the third subsection, we discuss particular implementations of crowd-based operators that are or could be incorporated into such systems. Finally, in the last subsection we describe those challenges that are particular to leveraging human workers for data gathering tasks. We provide a taxonomy of the axes of complexity introduced by crowdsourced data acquisition. This discussion will set the stage for the work described throughout the rest of the thesis.

3.2 Overview of Relational Database Systems

In this section we provide an overview of relational database management systems (RDBMSs). The focus is on aspects that are relevant to topics in this thesis; this section can be skipped by readers familiar with the basics of traditional RDBMSs. The interested reader can find more in-depth discussion and further details in a standard database systems text, such as [46].

3.2.1 The Relational Model

Prior to the RDBMS, data management systems were built on top of file systems—an approach that had a number of critical pitfalls. Data organization was limited to the filesystem directory hierarchy, and accessing data required knowledge of the particular layout. Furthermore, there was no efficient access to data items within a file. Developers who wanted better performance needed to maintain their own structures, such as hash tables or binary trees, to achieve faster data access. These performance enhancements were constantly being re-invented for different applications.

Edgar F. Codd aimed to solve these problems with the *relational model*, first introduced in [47]. In the relational model, data is represented as a set of *relations*, also commonly referred to as tables. Each relation consists of an unordered set of rows called *tuples*. Tuples are comprised of a set of *attributes* (columns); each attribute has a particular data type, such as integer or string. The description of a relation's attributes is called its *schema*.

Retrieving data is accomplished with transformations of relations via set operations described by the *relational algebra*. Each operator performs a simple manipulation on a set of data; to answer a user query, these primitives are composed into more a complex tree of operators through which data flows.

Relational algebra defines the following binary operators on relations (e.g., R and S) with the same schema:

Union The union $R \cup S$ defines a new relation containing all tuples that are in either R or S.

Intersection The intersection $R \cap S$ defines a new relation containing all tuples in both R and S.

Difference The difference $R - S$ defines a new relation containing all tuples in R and not in S.

Cartesian Product The cartesian product $R \times S$ forms a new relation by taking each tuple from R and matching it with each tuple from S. Unlike the previous three operators, which produce a relation with the same schema as the input relations, the result of a cartesian product has a schema that is the union of the input relations' schemas[1].

Additional operators transform a single relation:

Select A selection σ of R produces a relation with all tuples that satisfy a specified set of constraints, or predicates, applied to the attributes of R. Simple predicates include standard

[1] Schemas? Schemata? (Let's call the whole thing off)

arithmetic operators, however many systems support **user-defined functions** (UDFs) to evaluate more complex predicates. The selection operator can be visualized as a horizontal slice of the relation.

Project A projection π of R creates a copy of R with only a specified subset of attributes, or columns, from R. Projection can be visualized as a vertical slice of the relation.

A commonly known operator is the **Join** ($\bowtie$). The join is actually an optimization: it is a selection and projection applied to a cartesian product, but avoids producing the full results of the cartesian product. There are several flavors of join operators; a full description can be found in a database systems text, such as [46].

Finally, extended operators typically found in modern query languages include:

Sort Applying the sort operator τ to R produces a list of all tuples in R, ordered by one or more attributes.

Group-by A group-by γ partitions R into groups of tuples based on the values of one or more attributes.

Aggregation Aggregation operators, such as COUNT, SUM, and MAX, act on attributes of R. Aggregation is commonly used in conjunction with the group-by operator, and produces an aggregate value per-group. If no grouping is applied to R, aggregation will act on the entire relation to produce a single aggregate value.

Duplicate-elimination While relational algebra defines operators over *sets* of data, in practice systems have relations with *bag* semantics, i.e., duplicate tuples are allowed within a relation. Applying the duplicate-elimination operator δ to a bag of tuples produces a set of distinct tuples.

One of the greatest strengths of the relational model is that it provides *data independence*. Users can reference the database schema and specify which data they want using a *declarative* query language. A commonly used query language is SQL, based upon the tuple relational calculus which has been shown to be equivalent in expressive power to relational algebra [48]. By specifying queries declaratively, users need not know how the data is organized on disk nor the details of how it can be retrieved (*physical data independence*). Furthermore, the relational model ensures that queries are robust to changes in the schema (called *logical data independence*), e.g., the addition of new attributes to a relation will not affect queries referencing the relation's other attributes.

This level of abstraction is exactly the feature that makes it easy to insert new crowd-based algorithms and techniques into the system. In Sections 3.3-3.4 we return to this topic with a discussion of hybrid human/machine database systems; we first provide additional context in the next subsection describing traditional relational database architecture and query processing.

3.2.2 RDBMS Architecture and Query Processing

There are many existing commercial and open-source RDBMSs that typically share a canonical architecture. Figure 3.1 illustrates a simplified system architecture diagram with many of these core components. The parser, optimizer, and executor, in consultation with the statistics and metadata components, are the main parties responsible for translating a query written in SQL into operations to access the data. Not shown are the transaction and recovery managers which handle data integrity in the presence of concurrent users and system failures.

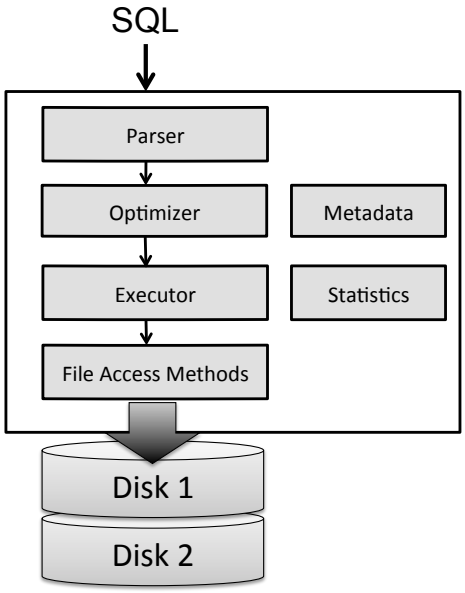


Figure 3.1: Architecture of a RDBMS with core components related to query processing

The first step of query processing is to parse a declarative SQL query and form an initial *logical query plan* using the relational algebra operators described above. For example, suppose there is a database with relations describing students and professors with the following attributes[2]:

```
Students(id, name, year, advisorId)
Professors(id, name, office, title)
```

If we want to know the names of professors who advise students who matriculated in 2014, we could issue this SQL query:

```
SELECT Professors.name
FROM Students, Professors
WHERE Students.advisorId = Professors.id AND year = 2014;
```

[2]For simplicity, we assume a student can only have one advisor

This query could then be translated into an initial relational algebra expression that first takes the cartesian product of `Students` and `Professors`, selects the tuples where the student's advisorId matches the professor's id, then further selects tuples for students starting in 2014, and finally projects the professors' names:

$$\pi_{Professors.name}(\sigma_{year=2014}(\sigma_{Students.advisorId=Professors.id}(Students \times Professors))) \quad (3.1)$$

The optimizer receives this initial logical query plan and performs two main phases of optimization. First it considers algebraic rewrite rules that transform the initial relational algebra expression into an equivalent expression that will evaluate more quickly, e.g., because fewer tuples will need to be processed at earlier stages of execution. One rewrite of expression 3.1 could "push down" the evaluation of the `year = 2104` predicate, such that many fewer tuples from the students relation need to participate in the cartesian product with the professors relation:

$$\pi_{Professors.name}(\sigma_{Students.advisorId=Professors.id}(\sigma_{year=2014}(Students) \times Professors)) \quad (3.2)$$

After modifying the logical query plan, the optimizer is tasked with formulating an actual execution strategy. This transformation creates a *physical query plan*, and it is informed by available implementations for the plan operators. These choices include different techniques for accessing data on disk, as well as algorithms for joining relations. This brief overview of query optimization is just the tip of the iceberg; more details can be found in [49] and a database systems text, such as [46].

3.3 Architectures of Hybrid Systems

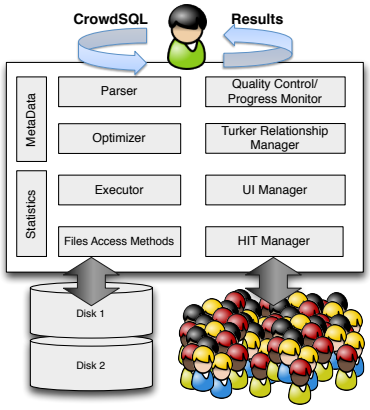
Relational database systems are powerful data management tools, incorporating decades of research and development. The level of abstraction and data independence provided by the relational model and query processing engine make it possible to augment the system with new and modified algorithms and implementations over time. The availability of APIs for using crowdsourcing presents an opportunity to incorporate human computation inside a query processing system, thereby allowing the system to answer a broader space of queries. Queries requiring human perception, subjective or ambiguous predicates, or knowledge about the world can be enabled using the crowd.

In this section, we compare and contrast three existing human/machine query processing systems proposed by different research groups. First we discuss changes made to the canonical database system architecture (Figure 3.1), in particular the new components added to assist with managing human computation. we then describe query processing in these hybrid systems. Namely, (1) how a declarative query is expressed and then transformed into tasks posted to a crowdsourcing platform, (2) how crowd responses are combined to yield the query result, as well as (3) how the systems reason about how much work to send to the crowd to manage crowdsourcing budget considerations.

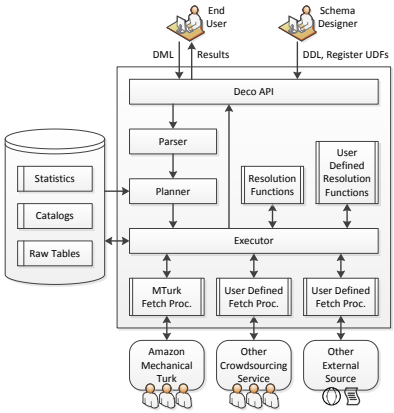
3.3.1 Comparison of Existing Architectures

Figure 3.2 shows architecture diagrams of three hybrid human/machine systems. These systems were developed around the same time by three different Database Systems research groups: UC Berkeley's CrowdDB [11], Stanford's Deco [12, 50], and MIT's Qurk [13, 51]. Note that while the work in this thesis could be incorporated into any of these systems, it was done in the particular context of the CrowdDB project. Each system incorporates human computation within a relational database, albeit with a few different design decisions, which we discuss in this section. However, the architectures share many of the components from a traditional RDBMS discussed earlier, including the query parser, optimizer/planner, and executor.

(a) CrowdDB architecture (Figure 1 in [11])



(b) Deco architecture (Figure 4 in [50])



(c) Qurk architecture (Figure 1 in [13])

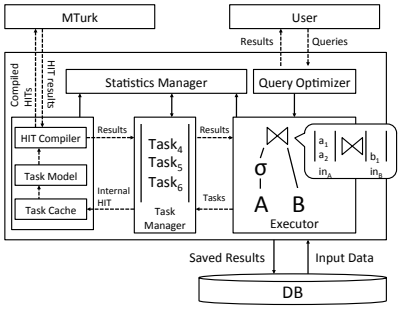


Figure 3.2: Architecture diagrams from three hybrid human/machine database systems.

Each system additionally has new and modified components explicitly for managing crowdsourced data. These components, while named differently and/or embedded within other components, accomplish several main tasks. Table 3.1 briefly summarizes these tasks and notes where they appear in each system. Each system treats the crowd as a data source that is accessed via APIs; the top row of the table lists the components in the systems that manage the posting of human computation tasks to a crowdsourcing platform. The bottom three rows of the table mention the components responsible for designing crowdsourcing tasks, aggregating results from the crowd, and reasoning about crowdsourcing budget. We discuss these topics in detail in the remainder of this subsection, referring back to this table.

Task	CrowdDB	Deco	Qurk
Making API calls to crowd platform, such as AMT	HIT Manager, Turker Relationship Manager	MTurk Fetch Procedures	HIT Compiler
Designing the crowdsourcing tasks	UI Manager	Described by fetch rules	Task Manager
Aggregating results from the crowd	Within physical query plan operators	Resolution functions	Task Manager
Crowdsourcing budget considerations	Executor	Executor (adaptive process)	Statistics Manager, Task Cache/Model

Table 3.1: Components of hybrid human/machine systems responsible for key tasks in managing crowdsourced data.

3.3.2 Collecting Data: Query to Crowd

Each system supports some variant of the relational data model, with minor changes to indicate which data has been or can be crowdsourced. Users interface with these hybrid systems using a declarative query language; all three systems use variants of SQL. Some include language extensions; for example CrowdDB adds the built-in functions `CROWDEQUAL`, which uses the crowd to determine if an attribute equals a given argument, and `CROWDORDER`, which sorts tuples based on human input. The Qurk system makes heavy use of UDFs[3].

To illustrate what a query looks like in these systems, the examples below query a relation containing images to determine which best depict the Golden Gate Bridge. This query written in CrowdDB's CrowdSQL is[4]:

```
SELECT p FROM picture
WHERE subject = "Golden Gate Bridge"
ORDER BY CROWDORDER(p,"Which picture visualizes better %subject");
```

A query achieving similar functionality in Qurk might look like this:

```
SELECT p FROM picture
WHERE subject = "Golden Gate Bridge"
ORDER BY betterVisualizes(p,subject);
```

In the hypothetical Qurk query, `betterVisualizes()` is a UDF that the developer would write.

At query runtime, the executor determines which parts of the query plan call for crowd input.

[3] described in Section 3.2.1

[4] Example 4 in [11]

Each of the three systems use pre-defined task templates that the system automatically translates into the HTML that will be posted to the AMT platform[5]; see the second row of Table 3.1 to see in which component the translation occurs in each system. CrowdDB and Deco use schema information to populate the content of the task templates. In CrowdDB, the `CROWD` keyword is used when defining the schema to denote which relations (`CROWD` tables) and attributes (`CROWD` columns) will be crowdsourced. To retrieve data for a `CROWD` table or column in CrowdDB, a simple HTML form is generated with text boxes labeled by the attribute names; when possible, existing data values are pre-populated in the text boxes. Deco uses developer-specified "fetch rules" to create task forms in a similar fashion: a rule of the form $A_1 \rightarrow A_2$, where A_1 and A_2 are two subsets of a table's attributes, pre-populates the task template with the existing values in A_1 and asks the crowd to input values for A_2. In Qurk, a special task definition is used to the populate task templates for a given UDF. Other interfaces and techniques for collecting data in crowdsourced tables are discussed in Chapter 5 of this thesis.

3.3.3 Aggregating Worker Responses

Recall from Chapter 2 that crowdsourcing platforms such as AMT provide the capability to ask multiple workers to complete the same task. This feature can be used to implement quality control strategies, such as the majority vote technique discussed in Chapter 2, or to aggregate multiple opinions, such as averaging numerical ratings for restaurants. In essence, the aggregation process involves "combining" multiple responses to yield a single value.

Each of the three hybrid database systems supports combining multiple crowd responses for a given task (see third row of Table 3.1). In CrowdDB, the majority vote technique is built into the task templates. Qurk users can define an arbitrary `Combiner` in the task definition for a UDF. The system currently provides the combiners `MajorityVote`, as well as `QualityAdjust`. The latter implements an algorithm from Ipeirotis et al. [32] that incorporates models of worker error and bias, mentioned in Section 2.3 in Chapter 2.

The systems differ slightly in how data collected from the crowd is stored. CrowdDB and Qurk first apply a aggregation or combiner strategy to the set of crowd responses, then persist the result to disk. In contrast, Deco persists the uncombined, "raw" data. The developer writes "resolution rules" that dictate how raw data should be combined in response to a user query. For example, the `majority-of-3` rule states that the final answer is the response given the most from a total of three worker responses. The rule `dupElim` (duplicate elimination) takes a set of responses and returns a set containing only distinct values.

3.3.4 Deciding How Much to Crowdsource

Annotations in the schema and query, such as the `CROWD` label in CrowdDB, indicate when human computation will be invoked. As described in Chapter 2, explicit crowdsourcing platforms such as AMT require requesters to pay crowd workers for correctly completing HITs. Furthermore,

[5] All systems describe the capability to add additional templates

many quality control strategies increase this cost; e.g., requiring a majority vote of responses from three crowd workers multiplies the crowdsourcing cost by three. Because of this dynamic, notions of crowdsourcing budget and the cost versus quality tradeoffs come to the forefront of the query execution strategy for these hybrid query processing systems.

Controlling the amount of crowdsourcing used for a query is typically built into the query language and execution strategies of these systems. The fourth row of Table 3.1 shows the components in each system responsible for determining the amount of crowdsourcing needed for a given query. For example, the `LIMIT` clause in a CrowdDB query specifies the maximum number of tuples in the query result. Both Qurk and Deco have query annotation parameters that allow the user to indicate a maximum crowdsourcing budget as well as a maximum amount of time to wait for query results.

Furthermore, because these systems are often times used to augment existing data with crowdsourced data, consideration is given at query time into how much new crowdsourcing effort is required. In Qurk, the statistics manager component consults a task cache to determine if more tasks need to be posted to the crowdsourcing platform. Deco proposes a two-phase execution strategy: given a query, it first materializes results using data contained in the raw tables. If the query result has fewer tuples than a user-specified minimum, the system then invokes fetch rules to satisfy the remainder of the query.

In general, as we explore in the next section and throughout this thesis, hybrid systems face a fundamental question in their execution strategy: for a given query, how many more worker responses are needed to satisfy a quality or confidence threshold? In the next section, we describe existing algorithms and techniques for implementations of crowd-based query plan operators, such as `SELECT` (σ) and `SORT` (τ). The remainder of the thesis then focuses on contributions to crowd-based set enumeration, an implementation of the physical `SCAN` operator that retrieves tuples from a relational table.

3.4 Crowd Database Operators

Hybrid human/machine systems include new relational operators that leverage human computation. Crowd-based operators differ from traditional operators in several key ways. With people involved, we have to revise and/or create new algorithms that consider human behavior and errors, as well as balance the cost versus quality tradeoff that is inherent to paid crowdsourcing. For some operators, it is necessary to rethink the meaning of correctness. However, leveraging the crowd also gives these operators access to human knowledge, experience, and perception. This allows for subjective and fuzzy evaluations over data items, enabling the database system to answer more complex queries.

In this section, we describe work in human computation for filtering, sorting, clustering, and joining relational data. We discuss different approaches with respect to user interfaces, worker error modeling, as well as understanding and optimizing crowdsourcing costs involved with these implementations.

3.4.1 Filtering Operators

A crowd-based filter uses human computation to determine which of a set of items satisfy a given constraint, or predicate. It is a modification of the `SELECT` (σ) operator in a traditional query plan.

In the Qurk [51] system, users write UDFs to specify custom filters to apply on a set of items. These UDFs are automatically translated into a task that can be posted on AMT; the system batches multiple item evaluations in a single task. Crowd workers are asked to indicate with a yes/no response if each item satisfies the predicate. By default, Qurk sends each item to be evaluated by five workers. These responses can be reconciled via custom "combiners" such as majority vote.

Parameswaran et al. [52] look at incorporating a filter's selectivity, i.e., how likely a given item is to satisfy the filter, as well as workers' false positive and false negative error rates in a strategy for crowd-based filtering. Given these three statistics, the objective is to determine a strategy that minimizes the number of questions asked to the crowd while keeping the total error below a threshold. A *strategy* for filtering an item is defined using a sequence of yes/no responses from workers. At each point in this sequence, say after one yes and two no responses, a strategy dictates whether it will continue gathering responses, or return with a 'Pass' or 'Fail' indicating whether the item meets the given constraint. The authors propose algorithms for determining the approximate optimal deterministic strategy and the optimal probabilistic strategy.

The above definition of a strategy assumes that items are equally difficult to filter, that there is no prior knowledge about whether items will pass the filter, and that workers have homogeneous skills and are of equal cost. Parameswaran et al. [53] augment the representation of a strategy's state beyond a count of yes/no responses to include worker identity and allow each item to have its own selectivity. With these changes, a strategy becomes a Markov Decision Process (MDP), which can be solved via linear programming (LP). However, the LP solution, while optimal, can be intractable due to the amount of state maintenance required for workers and their abilities. To ameliorate this issue, they propose a "posterior-based approach", which describes a strategy's state as the probability of a 'Pass' given the cost incurred so far; this representation does not grow with the number of workers, providing tractability.

Das Sarma et al. [54] investigate the cost versus latency tradeoff for filtering with the crowd. Given a large set of items, the goal is to find k items that satisfy a predicate. The lowest cost algorithm is sequential: send one item at time to be evaluated by the crowd, wait for the response between phases, and stop after receiving k correct items. However, the lowest latency algorithm has the crowd process the entire dataset in one parallel phase. Ideally, a hybrid algorithm would find the balance between these low-cost/high-latency and the high-cost/low-latency options. They devise several approaches for slightly "parallelizing" the sequential version to gain some of the benefit of the extreme parallel version; the user can customize the aggressiveness of the parallelization strategy. They explore both a deterministic setting, in which humans do not make mistakes, as well as an uncertain setting in which each item in the set may need multiple evaluations in order for the algorithm to deduce correctness.

3.4.2 Sorting, MAX, and Top-k Operators

Crowd-based sorting operators use human computation to order a given dataset by the value of one or more attributes, for example, images that best depict the Golden Gate Bridge. Related operators include finding the top k items and the `MAX` (i.e., top 1 item), from a set of ordered items sorted by the particular attributes. One strategy to achieve a crowd-based sort is to replace the comparator method used in a traditional sorting algorithm with a method that asks people to evaluate the comparison, in a *comparison task*, to see which is "better" with respect to some description. For example, CrowdDB's `CrowdCompare` operator [11] generates a comparison task in which workers compare two items. In general, a comparison task asks a worker to sort some number s of items.

Work in this space looks into task design as well as modeling of the quality versus budget tradeoff for these operators. In the discussion here, we first describe task interfaces used by the Qurk [13] for crowd-based sort. We then look at several approaches in the literature that tackle the top-k and/or `MAX` problem without completing a full crowd-based sort of the input, thereby saving on crowdsourcing costs. These approaches investigate the probability that the result of an algorithm for determining the top-k or `MAX` is the true value, as well as how to reason about the cost versus correctness tradeoff.

For the sort operator for Qurk [13], Marcus et al. [51] investigate task interfaces for sorting a dataset with reasonable crowdsourcing cost. A full sort of N items based on comparison tasks, in which each task a (small) number of items are compared, would necessitate $O\binom{N}{2}$ comparisons because the transitive property may not hold in crowdsourced comparisons. In other words, there would be a cycle in the graph that has edges from item i to j when $i > j$.

An alternate task interface for sorting in Qurk asks workers to rate individual items with respect to the sorting criterion on a scale from one to seven, rather than comparing them. Workers are also shown a small random sample of other items in the dataset, to give a sense of the total distribution. This approach requires only $O(N)$ crowdsourcing tasks, however experimental results show that the ratings approach is less accurate than the comparison approach for items that are similar. The authors compare the comparison-only and ratings-only approach with an additional hybrid strategy that first uses the crowd for ratings, then iteratively performs crowd-based comparisons on select "windows" of items to improve quality while keeping to a specified budget. The hybrid strategy outperforms the other strategies because it fixes local inconsistencies in addition to having the power to shift items that are far away in the sort order from where they should be.

To execute a crowd-powered top-k or `MAX` query, an algorithm could simply sort the whole dataset and return the top value(s). However, this approach may waste money performing crowd-based comparisons between items that may not be necessary. Naturally, there is a tradeoff between the cost of performing more comparison tasks and achieving higher quality, i.e, approaching the true top-k that would result from executing all comparisons. We describe next approaches that perform fewer comparisons and reason about the probability or likelihood that the true top-k or maximum item has been found.

Algorithms based on tournament-sort have been investigated in the literature for implementing crowd-based top-k [55, 56, 57, 58]. In general, the sort algorithm operates as follows: the dataset

is partitioned into disjoint subsets of size s, with $s > k$. The crowd completes comparison tasks, in which each task contains s items; multiple worker responses per task are typically gathered for quality control purposes. After worker responses are combined (described next) for each of the subsets, the top k items from each subset advance to the next round of the tournament, in which new subsets of size s are formed and compared by the crowd, and so on. Two questions arise in the implementation of this algorithm: (1) how many workers should be asked to complete a comparison task for a given subset of s items? and (2) how should the results of multiple comparison tasks be combined?

Polychronopoulos et al. [55] address the second question of how to combine crowd responses using a technique called *median rank aggregation*, which assigns the rank for an item as the median of its position in the different rankings produced by the workers for the the s items. Then, by determining if the median rank metric's uncertainty is beyond a threshold, their algorithm adaptively decides that more workers should evaluate a particular subset s of items. Experimental results show this adaptive strategy outperforms a basic strategy that uses a fixed number of comparison tasks per item subset, because fewer (more) comparison tasks are used for dissimilar (similar) items. When compared to the comparison-only strategy from Marcus et. al [51] described above, the algorithm in this work costs an order of magnitude less to achieve the same accuracy.

To address the first question above, Davidson et al. [56] represent worker error, the likelihood a worker will answer a comparison task incorrectly, with a variable error model in which the probability of error decreases the further apart items are in the true ordering. For example, a query might ask for the top-k most recent pictures of the same person; it is easier to detect that a picture of the person as an adult was taken more recently than a picture of that person as child, versus comparing pictures taken only a week apart. The error model can be learned using training examples for which the true comparison values are known. The sorting algorithm is based on tournament-sort, with the tournament tree divided into upper and lower levels. Comparisons in the earlier, lower level are evaluated using one crowd comparison. Match-ups in the upper level get N_L votes, with N_L based on the characteristics of the variable error model (i.e., exponential, linear, or logarithmic). Incorporating the variable error model, they provide formal bounds on the number of value questions needed to determine the correct top-k or `MAX` item(s) with high probability.

Guo et al. [58] look at assessing the state of a tournament sorting algorithm at a particular stage in its execution, specifically for determining the maximum item (evaluating `MAX`). The *judgement problem* is to determine which item has the greatest likelihood of being the maximum at the current stage. The *next votes problem* decides how to allocate more rounds of comparisons. In a particular round of a tournament-like sorting algorithm, workers may be extremely delayed in casting their votes, or in the worst case may not vote at all. However, if more votes are needed, the structure of the tournament (i.e., the scheduled match-ups) can be rearranged—the challenge is constructing this rearrangement.

They propose maximum likelihood approaches to both the *judgement* and *next votes* problems defined above, and show that each is NP-hard. They then develop and compare several heuristics to address them. A heuristic based on the PageRank algorithm proves effective for the *judgement problem*. For the *next votes problem*, two heuristics worked well. The first is a "greedy" algorithm, which weighs all eligible item pairs by the product of the scores derived from the PageRank algo-

rithm. A heuristic called "complete tournament" first executes a single tournament amongst the K highest scoring items. Then the $(K+1)$st item is paired with each of the first K, and the highest scoring pairs according to the greedy algorithm are compared. The parameter K is chosen so both steps of the heuristic have sufficient budget to complete.

Venetis et al. [57] present a generalization of the approaches to computing `MAX` with the crowd by describing a framework of families of algorithms; a family contains all possible parameterizations for a given algorithm. They describe a tournament-based algorithm using comparison tasks, as well as *bubble-max*. The bubble-max algorithm compares a pivot item with a set of random items not yet processed. The winner of this round becomes the new pivot, and the process continues until all items have been processed.

Parameters include r_i and s_i, the number of human responses and number of items in a comparison task, respectively, for the ith iteration of an algorithm. The goal is to determine the optimal parameterization for an algorithm that maximizes quality (i.e., the likelihood that the algorithm finds the true maximum item), subject to cost and time (i.e., rounds of crowdsourcing). However, the optimal settings of r_i and s_i are heavily influenced by the worker error model and quality/budget constraints of a particular scenario. They describe a number of strategies for determining the optimal parameter settings. Simulation results show that the best strategy first finds the optimal r and s assuming each remains constant across iterations of the algorithm, then performs a hill-climbing search technique to determine if "stealing" units of r and/or s from certain rounds and giving them to others yields quality improvements. This strategy outperforms alternate strategies for both tournament-based and bubble-max algorithms. Tournament is also superior to bubble-max given the same budget.

Work in the space of crowd-based sort, top-k and `MAX` varies from interface design to formalization of the problem. A common theme is the consideration of the budget versus quality tradeoff and reasoning about how many comparison tasks to evaluate using the crowd to determine the correct (or good-enough) top-k or max item with high likelihood. An important part of this reasoning process includes modeling worker error and relative item similarity to develop adaptive and dynamic algorithms.

3.4.3 Group-by Operators

The group-by operator effectively partitions, or clusters, a relation based on the value of one or more attributes. Davidson et al. [56] investigate using the crowd to cluster a given set of items into distinct types, where the number of types is assumed to be fixed but is unknown. They use "type" questions: a worker is shown two items and asked if they match for a particular type, e.g., do two photos depict the same person. Given that each question asked incurs crowdsourcing costs, the goal is to deduce how many questions will be needed for a particular dataset. Workers are assumed to have a *constant* error model, i.e., they will answer questions correctly with probability $1/2 + \epsilon$. They provide bounds on the number of type questions needed to achieve correct clustering with high probability.

Gomes et al. [59] look at a slightly different goal: can the crowd be used to *discover* categories, or clusters, in a large dataset? Workers are shown a small subset of the data and asked to label,

via color-coding, which items belong in the same category. The objective is to form a global categorization based on these individual clusterings. While the subsets given to workers overlap to facilitate aggregation, workers may disagree on a clustering or number of clusters even given the same subset. They propose a Bayesian model for aggregating the clusterings in which workers are pairwise binary classifiers in the space of items (pairwise item assessments can be derived from the workers' clusterings). Experimental results show superior performance over two existing clustering aggregation methods.

3.4.4 Join Operators

There are several ways in the literature in which the crowd can be involved in a crowd-based join. In CrowdDB [11], the CrowdJoin operator performs a index nested-loop join between two relations, at least one of which must be a `CROWD` table (i.e., its tuples are provided by the crowd). For each tuple of the outer relation, new tuples for the inner relation are gathered from crowd workers; the task UI is pre-populated with the join column values from the outer relation.

The crowd can also help evaluate the join predicate(s) between relations. In Qurk [51], workers are tasked with deciding if tuples from two relations match with respect to one or more attributes. A naive implementation of this join would attempt to ask the crowd about every possible pairing of tuples between the relations, i.e., the cartesian-product. To potentially reduce the number of tuples participating in the join, Qurk allows users to specify additional predicates that must be true for each relation if the join predicate is true; evaluating these predicates first has a cost only linear in the size of the relations. The additional predicates are also UDFs, and are specified in the query using the keyword `POSSIBLY`. Depending on how costly each predicate is to evaluate, the optimizer may decide to forego one or more of them (hence, the predicates are each *possibly* evaluated). The particular join implementation is a block nested-loop join: partitions of one relation are iteratively compared for matches with tuples from the other relation.

Three task user interfaces (UIs) are compared and evaluated for the crowd-based join. The first is the simple single page task that presents the worker with one item from each relation and asks if they have the same value for the given predicate. The second, "naive batching" places multiple of these pairs in the same task. The "smart batching" interface presents two columns of items and asks the worker to indicate all pairs that match. Experimental results show the accuracy amongst the different UIs was roughly equivalent, highlighting the potential of cost savings due to batching. The highest accuracy was attained with the smart batching UI using two items from each relation, coupled with the quality control mechanisms that model worker error and bias described in [32].

3.4.5 Summary of Crowdsourced Query Operators

In this section, we described crowd-based relational operators that can be plugged into hybrid human/machine database systems. These operators enable the system to evaluate queries that require human knowledge, perception, and experience, leveraging the programmability of crowdsourcing platforms such as AMT to develop algorithms and techniques that incorporate human computation.

A common theme throughout this section was the balance of crowdsourcing cost and the quality of the query result; this theme continues in the rest of this thesis as well.

The operators discussed in this section assume there is a set of data items that crowd workers are processing. However, the crowd can also be helpful for assembling the dataset itself. In a relational query plan, this data collection would be akin to the physical `SCAN` operator that retrieves content from a relational table. Rather than reading tuples from disk storage, a crowd-based `SCAN` gathers tuples using the crowd.

Naturally, there are challenges with crowdsourced data collection, and the next section begins a discussion of some of these issues. This thesis explores the implementation of a crowd-based `SCAN` operator; first to reason about quality and understand its semantics, then to discuss different user interfaces for interacting with the crowd for data collection tasks, as well as how to leverage automated processing via existing search engines to reduce the cost the amount of crowdsourcing needed for data collection tasks.

3.5 Challenges of Crowd-Based Data Collection

The primary focus of this thesis is to investigate crowd-based data collection, i.e., using human computation to gather sets of entities or items. For example, we might be interested in having the set of names of all graduate students in Computer Science who are currently on the job market, or the names of restaurants in San Francisco that have scallops on their dinner menu. This data gathering process is analogous to the `SCAN` operator in a physical query plan in a RDBMS. However, rather than accessing the data from disk storage, the individual items in the set may be spread out over the web, obfuscated in complex web design, or buried in human minds.

The crowd-based `SCAN` operator has a unique set of challenges that arise due to the nature of using people to gather data. In particular, not all data sets are equally difficult to collect. Table 3.2 below highlights some of the axes that complicate crowd-based data collection.

Axis	Range
Number of sources	0 . . . many
Domain knowledge	none . . . expert
Fuzzy membership	none . . . extreme
Set size	small . . . large/infinite

Table 3.2: Axes of complexity in crowdsourced data collection

Number of sources The number of different sources crowd workers would need to consult to find all items. If they can all be supplied from memory or through some reasoning process, then no sources are required. At the other extreme, every item might be found on a different webpage.

Domain knowledge The domain knowledge axis is an indicator of how much context or background is required to find and/or produce items in the set. Some sets can be enumerated by any layperson, while others may necessitate expert knowledge of a particular field.

Fuzzy membership Whether a given item belongs in a set may or may not be objective. For example, members of the set "countries in the world" could vary depending on who is asked, due to ongoing political conflicts. Furthermore, the web may contain conflicting information.

Set size Lastly, set size is the cardinality, or total number of items that constitute the complete set. A set could be small, large, or even infinite in size.

The particular combination of values along these axes influences how difficult it might be to gather all items for a set. For example, A set may require training or an expert to find its members, rather than unskilled workers on a marketplace such as AMT. As the number of sources increases, items may become more and more difficult to locate. More crowdsourcing budget might need to be allocated for determining item membership for fuzzy or ambiguous sets.

Figure 3.3 shows example target sets that are increasingly difficult to gather using the crowd. Easier sets include enumerating the numbers $1 - 100$ or the numeric palindromes less than 1000: workers do not need to consult references to contribute items, and each item's membership is objective and thus easily verified. Palindromes may have a slightly more challenging set definition. In contrast, finding the names of players on a sports team or actors in a movie typically requires consulting a source that lists these data items. Furthermore, it is not entirely clear if backup players or movie extras belong in the two respective sets, indicating a certain level of fuzziness. Collecting the names of graduate students on the job market is difficult for several reasons. The information resides on individual students' webpages (and possibly in their advisors' minds), and it will require the domain knowledge of people in the academia to determine where to look and who to ask.

task	# sources	domain	fuzziness
Numbers 1 to 100	0	None	None
Palindromes 1 to 1000	0	Little	None
Actors in a movie, given link	1	Little	Little
Sports team players	1	Little	Little
Population per country	~200	Little	Little
Companies with blue logo	Infinite	Little	Little
Professors at a university	100s	Medium	Little
...			
Graduating graduate students	100s	Expert	Medium

Figure 3.3: Example data collection tasks with varying difficulty in the complexity space.

Another complexity of crowd-based data collection is that it introduces a more complicated definition of answer quality. In addition to per-item quality, namely, does a given item belong in the set and are its attributes correct, there is a notion of quality of the set as a whole. In other words, the quality of a set can be measured by its *completeness*. Set completeness is an intrinsic issue for the budget versus quality tradeoff for crowd-based data collection.

Unfortunately, when it comes to the discussion of completeness, *you don't know what you don't know*. In order to determine completeness for a given set of items that the crowd is gathering, there needs to be knowledge of how many items are expected. It may seem natural to ask the crowd directly for the set size, or cardinality, but in practice this proves ineffective for most datasets. The only ways the crowd could provide set cardinality without actually counting items would be if the workers already knew or could find the cardinality information on the web. In the remainder of this thesis, we will investigate crowd-based data collection, with a particular focus on set quality, budget considerations, and the challenges of human behaviors.

Chapter 4

Enumeration Queries Using the Crowd

4.1 Introduction

As described in Chapter 3, a number of projects have begun to explore the potential of hybrid human/computer systems for database query processing. In these systems, human workers can perform query operations such as subjective comparisons, fuzzy matching for predicates and joins, entity resolution, etc. For example, recall that CrowdDB incorporates several SQL language extensions to involve people in query processing. Of particular relevance to the work presented here, the CrowdDB Data Definition Language (DDL) includes the special keyword `CROWD` to indicate when missing values of existing records or entire missing rows of certain tables can be obtained via human input, say by posing jobs on a crowdsourcing platform such as AMT. As shown in [11], these simple extensions can greatly extend the usefulness of a query processing system.

Of course, many challenges arise when adding people to query processing, due to the peculiarities in latency, cost, quality and predictability of human workers. For example, data obtained from the crowd must be validated, spelling mistakes must be fixed, duplicates must be removed, etc. Similar issues arise in data ingest for traditional database systems through ETL (Extract, Transform and Load) and data integration, but techniques have also been developed specifically for crowdsourced input [32, 60, 61, 20], discussed in Chapter 2.

While there has been work addressing the above concerns, none address a more fundamental issue that arises in hybrid human/machine systems; specifically, when the crowd can augment the data in the database to help answer a query, the traditional *closed-world assumption*, which assumes that the database is complete, and on which relational database query processing is based, no longer holds. This fundamental change calls into question the basic meaning of queries and results in a hybrid human/computer database system.

4.1.1 Can You Really Get it All?

In this chapter, we consider one of the most basic RDBMS operations, namely, scanning a single table with predicates. Consider, for example, a SQL query to list all restaurants in San Francisco serving scallops: `SELECT DISTINCT name FROM RESTAURANTS, DISHES WHERE CITY`

= 'San Francisco' and DISH LIKE 'Scallops'. In a traditional RDBMS there is a single correct answer for this query, and it can be obtained by scanning and joining the tables, filtering the records, and returning all matching records of the table. This approach works even for relations that are in reality unbounded, because the closed world assumption dictates that any records not present in the database at query execution time do not exist. Of course, such limitations can be a source of frustration for users trying to obtain useful real-world information from database systems.

In contrast, in a crowdsourced system such as CrowdDB, once the records in the stored table are exhausted, jobs can be sent to the crowd asking for additional records. The question then becomes: when is the query result set complete? Crowdsourced queries can be inherently fuzzy or have unbounded result sets, with tuples scattered over the web or only in human minds. For example, consider a query for a list of graduating Ph.D. students currently on the job market, or companies in California interested in green technology. These types of queries are the main use cases for crowd-enabled database systems, as each is labor-intensive for the user issuing the query to perform, but not executed frequently enough to justify the use of a complex machine learning solution.

In this chapter we address the question of "How should users think about enumeration queries in the open world of a crowdsourced database system?". We develop statistical tools that enable users to reason about tradeoffs between time/cost and completeness, and that can be used to drive query execution and crowdsourcing strategies.

4.1.2 Counting Species

The key idea of our technique is to use the arrival rate of new answers from the crowd to reason about the completeness of the query. Consider the execution of a "SELECT DISTINCT *" query in a crowdsourced database system where workers are asked to provide individual records of the table. For example, one could query for the names of the 50 US states using a microtask crowdsourcing platform such as AMT by generating HITs (i.e., Human Intelligence Tasks) that would have workers provide the name of one or more states. As workers return results, the system collects the answers, keeping a list of the unique answers (suitably cleansed) as they arrive.

Figure 4.1 shows the results of running that query, with the number of unique answers received shown on the vertical axis, and the total number of answers received on the x-axis. As would be expected, initially there is a high rate of arrival for previously unseen answers, but as the query progresses (and more answers have been seen) the arrival rate of new answers begins to taper off, until the full population (i.e., the 50 states, in this case) has been identified.

This behavior is well-known in fields such as biostatistics, where this type of figure is known as the *Species Accumulation Curve* (SAC) [62]. Imagine you were trying to count the number of unique species of animals on an island by putting out traps overnight, identifying the unique species found in the traps the next morning, releasing the animals and repeating this daily. By observing the rate at which new species are identified over time, you can begin to infer how close to the true number of species you are. We can use similar reasoning to help understand the execution of set enumeration queries in a crowdsourced query processor.

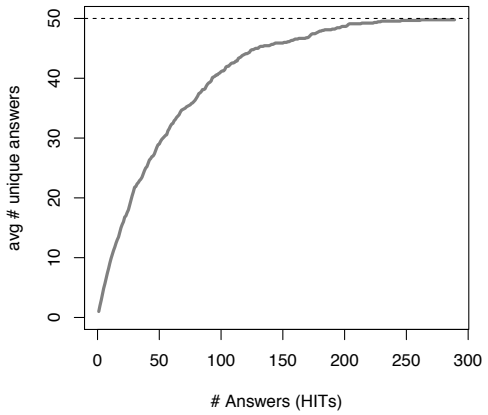


Figure 4.1: Accumulation curve for crowd-based enumeration of the US States: number of unique items seen for after total number of responses

4.1.3 Overview of the Chapter

In this chapter, we apply species estimation techniques from the statistics and biology literature to understand and manage the execution of set enumeration queries in crowdsourced database systems. We find that while the classical theory provides the key to understanding the meaning of such queries, there are certain peculiarities in the behavior of microtask crowdsourcing workers that require us to develop new methods to improve the accuracy of cardinality estimation in this environment. We also describe methods to leverage these techniques to help users make intelligent tradeoffs between time/cost and completeness. These techniques extend beyond crowdsourced databases and, for example, can help to estimate the completeness of deep-web queries.

To summarize, this chapter makes the following contributions:

- We formalize the process of crowdsourced set enumeration and describe how it violates statistical fundamental assumptions of existing species estimation techniques.
- We develop a technique to estimate result cardinality and query progress in the presence of crowd-specific behaviors.
- We devise a pay-as-you-go approach to allow informed decisions about the cost/completeness tradeoff.
- We examine the effectiveness of our techniques via experiments using Amazon Mechanical Turk (AMT).

This chapter is organized as follows: In Section 4.2 we provide background on species estimation, also called cardinality estimation. Section 4.3 describes how crowd-specific behaviors break assumptions on which species estimation algorithms are based. In Section 4.4 we develop techniques to improve the estimation in the presence of crowd-specific behavior, and Section 4.5 introduces

a pay-as-you-go technique. Finally, in Section 4.6 we cover related work and in Section 4.7 we summarize the chapter.

4.2 Background: Cardinality Estimation

To estimate progress as answers are arriving, the system needs an estimate of the result set's cardinality. We can tackle cardinality estimation by applying algorithms developed for estimating the number of species in a new context. In the species estimation problem [17, 63], an estimate of the number of distinct species is determined using observations of species in the locale of interest. These observations represent samples drawn from a probability distribution describing the likelihood of seeing each item. By drawing a parallel between observed species and answers received from the crowd, we can apply these techniques to reason about the result set size of a crowdsourced query.

4.2.1 Estimation Fundamentals

Receiving answers from workers is analogous to drawing samples from some underlying distribution of unknown size N. Each answer corresponds to one sample from the item distribution. We can rephrase the problem as a species estimation problem as follows: The set of HITs received from AMT is a sample of size n drawn from a population in which elements can be from N different classes, numbered $1 - N$ (N, unknown, is what we seek); c is the number of unique classes (species) observed in the sample. Let n_i be the number of elements in the sample that belong to class i, with $1 \leq i \leq N$. Of course, some $n_i = 0$ because they have not been observed in the sample. Let p_i be the probability that an element from class i is selected by a worker, $\sum_{i=1}^{N} p_i = 1$; such a sample is often described as a multinomial sample [17]. At various points in this thesis, we discuss the *skew* of a distribution, which measures the asymmetry amongst the p_i's. A distribution with low skew has a smaller difference between the largest and smallest p_i's than a distribution with high skew.

Uniformity assumption

If we initially assume a uniform item distribution, each class is equally likely to be selected: $(p_1 = p_2 = \cdots = p_N)$. This transforms the species estimation problem into a simple inference with the single parameter N. An approximate maximum likelihood estimator (MLE) is the solution N of the equation [64]:

$$c = N(1 - e^{-n/N}) \tag{4.1}$$

This solution is related to classic urn sampling problems such as the coupon collector or occupancy problems [65, 66].

Estimators that assume an underlying uniform distribution can work well enough for item distributions with low skew. When the item distribution is heavily skewed, however, unseen unique items are acquired more slowly than in the uniform case. Thus the cardinality estimate produced

by an estimator assuming equiprobable items will be an underestimate and can be thought of as a lower-bound [17].

Frequency of frequencies

To reason about skew in the underlying item distribution, some estimators use a statistic called the "frequency of frequencies", which we refer to as the "f-statistic". The f-statistic captures the relative frequency of observed items in the sample. For a population that can be partitioned into N classes (items), and for a given sample of size n, f_j is defined as the number of classes that have exactly j members in the sample, thus $\sum_{j=0}^{\infty} j f_j = n$. Notably, f_1 represents the "singletons" and f_2 the "doubletons". The goal of an estimation algorithm is to predict the value of f_0, i.e., the number of classes that exist but appear zero times in the sample.

To illustrate the effect of skew on the f-statistic, we can compare the crowd-based enumeration of the 50 US States described in the introduction with what would happen if we drew a random sample from a uniform distribution. Figure 4.2 shows two histograms: the light-colored bars represent the values of the f-statistic from the US States experiments after the crowd has completed 200 HITs; the darker bars show the values from a simulation, drawing 200 samples from a uniform distribution over 50 unique classes.

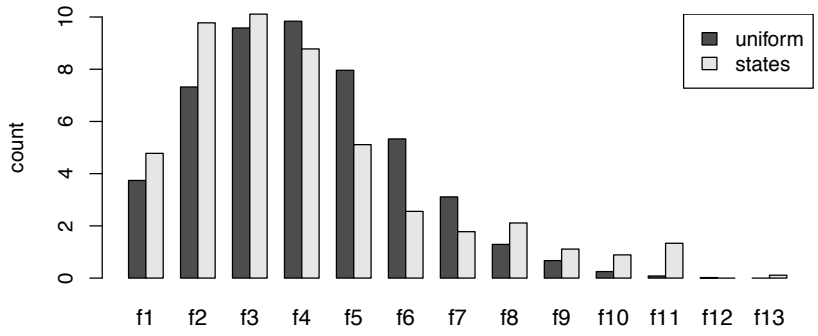


Figure 4.2: Comparison of the f-statistic histogram after 200 responses of the crowd-based enumeration of the US States vs. 200 samples from a uniform distribution over fifty items.

After 200 samples from a uniform distribution over 50 items, one would expect most items would have appeared approximately four times; indeed the dark bars are bell-shaped centered at f_4. In contrast, the states experiment has the most mass at f_3 and more mass than the simulation on the higher f's, indicating that some states appear very frequently (i.e., well-known states such as New York and California).

One might try to estimate the underlying distribution $\{p_1 ... p_N\}$ in order to predict the cardinality N. However, Burnham and Overton [67] show that the f-statistic is a sufficient statistic for estimating f_0, the number of unobserved species. Thus the goal is to form a cardinality estimate by predicting the value of f_0.

Parametric approaches attempt to predict f_0 by fitting an existing distribution to the f-statistic, such as a lognormal or inverse gaussian. The problem with the parametric approach is that the estimate will be poor if the chosen distribution does not fit the data well; furthermore the choice of distribution for one use case might not hold for another. Non-parametric approaches use only the f-statistic, thereby putting no restrictions on the underlying distribution.

4.2.2 Non-Parametric Estimators

Two common non-parametric estimators are *Chao84* and *Chao92*, developed by statistician Anne Chao; both are based on the f-statistic. Notably, these estimators are heavily influenced by the information about rare items (i.e., f_1 and f_2). The intuition behind this is that the presence of rare items indicates the likely existence of other items that are not currently represented in the sample (the f_0).

Chao84 estimator

In [68], Chao develops a simple estimator for species richness that is based solely on the number of singletons and doubletons found in the sample:

$$\hat{N}_{chao84} = c + \frac{f_1^2}{2f_2}$$

Chao finds that it actually is a lower bound, but it performed well on her test data sets. She also found that the estimator works best when there are relatively rare species, which is often the case in real species estimation scenarios.

Chao92 estimator

In [69], Chao develops another estimator which uses *sample coverage* to predict N. The sample coverage C is the sum of the probabilities p_i of the observed classes. However, since the underlying distribution $p_1 ... p_N$ is unknown, the Good-Turing estimator [64] using the f-statistic is used:

$$\hat{C} = 1 - f_1/n \tag{4.2}$$

Furthermore, the Chao92 estimator attempts to explicitly characterize and incorporate the skew of the underlying distribution using the *coefficient of variance* (CV), denoted γ, a metric that can be used to describe the variance in a probability distribution [69]; we can use the CV to compare the skew of different class distributions. The CV is defined as the standard deviation divided by the mean. Given the p_i's ($p_1 \cdots p_N$) that describe the probability of the ith class being selected, with mean $\bar{p} = \sum_i p_i/N = 1/N$, the CV is expressed as $\gamma = [\sum_i (p_i - \bar{p})^2/N]^{1/2} \,/\, \bar{p}$ [69]. A higher CV indicates higher variance amongst the p_i's, while a CV of 0 indicates that each item is equally likely (a uniform distribution[1]).

[1] This is why datasets with low CV can still have good estimates using equation 4.1

The true CV cannot be calculated without knowledge of the p_i's, so Chao92 uses an estimate $\hat{\gamma}$ based on the f-statistic:

$$\hat{\gamma}^2 = \max\left\{\frac{\frac{c}{\hat{C}}\sum_i i(i-1)f_i}{n(n-1)} - 1\,,\,0\right\} \tag{4.3}$$

The final estimator is then defined as:

$$\hat{N}_{chao92} = \frac{c}{\hat{C}} + \frac{n(1-\hat{C})}{\hat{C}}\hat{\gamma}^2 \tag{4.4}$$

Note that if $\hat{\gamma}^2 = 0$ (i.e., indicating a uniform distribution), the estimator reduces to $c/\hat{C}$.

4.2.3 Estimation in Database Systems

Work on distinct value estimation in traditional database systems has also looked into species estimation techniques to inform query optimization of large tables; tuples are sampled to estimate the number of distinct values present. Techniques used and developed in that literature [70, 71, 72] leverage knowledge of the full table size, which is possible only because of the closed-world assumption. In the species estimation literature, the difference between these two scenarios is referred to as *finite* vs. *infinite* populations, which correspond to closed vs. open world, respectively.

When crowdsourcing a set enumeration query, two key characteristics differentiate our scenario from distinct value estimation in traditional systems. Being in the open-world, we cannot take advantage of estimators that leverage knowledge of the full table size. And, perhaps more importantly, the nature in which crowd workers provide their answers creates a fundamentally different sampling scenario than that assumed by species estimation algorithms. We discuss worker behavior in detail next.

4.3 Progress Estimation for Crowdsourced Enumerations: Model and Analysis

As described above, various techniques have been devised in biostatistics to estimate the number of species as well as in the database community to estimate the number of distinct values in a table. They all operate similarly: a sample is drawn at random from a population (e.g., the contents of the relation) and based on the frequency of observed items (distinct values), the number of unobserved items (number of missing distinct values) is estimated. The techniques differ most notably in their assumptions. In particular, distinct value estimation techniques assume that the population (i.e., table) size is known. Unfortunately, knowledge of the population size is only possible in the closed world; in systems with crowdsourced enumerations, records can be acquired on-demand, thus the table size is potentially infinite. We focus the remaining discussion on species estimators suitable for the open world because they allow for an infinite population.

In this section, we describe our observations of how the crowd answers set enumeration queries and why existing estimation techniques yield inaccurate estimates. We also present a model for

crowdsourced enumerations and list the requirements for cardinality estimators that are tolerant to how workers behavior in response to enumeration queries.

4.3.1 The Problem with Existing Estimators

To gain an understanding of the crowd's ability to answer set enumeration queries and the impact of crowd behaviors on existing estimation techniques, we crowdsourced the elements of sets for which the true cardinality is known. We use the open-world-safe estimator "Chao92" [69] described in the previous section as it is widely used in the species estimation literature [73].[2] Figure 4.3 shows the observed Chao92 estimate ("actual") evaluated as answers arrive in one AMT experiment in which we crowdsourced the names of the 192 United Nations (UN) member countries and compares it to the expected behavior using simulation with the empirical data distribution derived from all runs of the UN experiment. We focus on a single representative experiment rather than an average over multiple runs to investigate the behavior a user would observe; averaging can also disguise the effects we describe next.

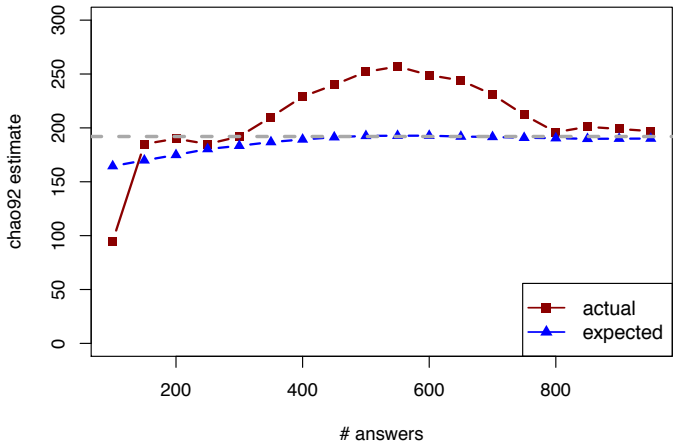


Figure 4.3: Chao92 cardinality estimate evaluated for increasing number of samples, or answers from the crowd, for the United Nations use case. "Actual" depicts the estimates for answers received during a crowd experiment; "Expected" is a simulation based on a with-replacement sampling process.

Note that in Figure 4.3 that the value of the estimate begins approaching the true value of 192 (horizontal line shown in grey), however it then significantly *overestimates* the true value for most of the remaining time of the experiment. This is surprising as our simulation shows that the estimate should become more accurate and stable as it receives more data ("expected" in Figure 4.3). As it turns out, the way in which crowd workers each provide their answers deeply impacts the behavior of an estimation algorithm. For example, in five runs of the UN experiment,

[2]We experimented with various other estimators for the open-world such as "Chao84"[68], "Jackknife"[67], and "uniform maximum-likelihood"[69] and also found Chao92 to be superior.

we observed various trends in how the crowd responds to a set enumeration query. A worker could enumerate the UN countries by traversing an alphabetical list, starting with Afghanistan. However, it was not uncommon for workers to begin their answer sequence with a few countries they knew of (e.g., United States, India, Pakistan, China, etc.), or to provide a completely non-alphabetical sequence. We even observed alphabetical traversals that began at the end or in the middle of the alphabet! In general, people may use different internal biases or techniques for finding items in the set (we discuss full list traversals in Chapter 5). We also noticed that individual workers complete different amounts of work and arrive/depart from the experiment at different points in time.

The next subsection formalizes a model of how answers arrive from the crowd in response to a set enumeration query, as well as a description of how crowd behaviors impact the sample of answers received. We then use simulation to demonstrate the principles of how these behaviors play off one another and thereby influence an estimation algorithm.

4.3.2 A Model for Human Enumerations

Species estimation algorithms assume a with-replacement sample from some unknown distribution describing item likelihoods. Figure 4.4(a) depicts this scenario: a sequence of items can be generated by the process ρ sampling with replacement from a distribution. Note that because a cardinality estimate is calculated after a given sample size, the particular order in which items in the sample arrive is irrelevant.

After analyzing the crowdsourced enumerations, for example in the previously mentioned UN experiment, we found that the assumptions of the traditional algorithms do not hold. In contrast to with-replacement samples, individual workers typically provide answers from an underlying distribution *without* replacement. Furthermore, workers might sample from different underlying distributions (e.g., one might provide answers alphabetically, while another worker provides answers in a different order).

This process of sampling significantly differs from what traditional estimators assume. It can be represented as a two-layer sampling process as shown in Figure 4.4(b). The bottom layer consists of many sampling processes λ_i, each corresponding to an individual worker i, that sample from some data distribution *without replacement*. The top layer process ρ samples *with* replacement from the set of the bottom-layer processes (i.e., workers). Thus, the sequence of responses from the crowd represents a with-replacement sampling amongst workers who are each sampling a data distribution without replacement.

The impact of the two-layer sampling process on the estimation can vary significantly based on the parameterization of the process (e.g., the number of worker processes, different underlying distributions, etc.). Next we study the impact of different parameterizations, define when it is actually possible to make a completeness estimation as well as the requirements for an estimator considering human behavior.

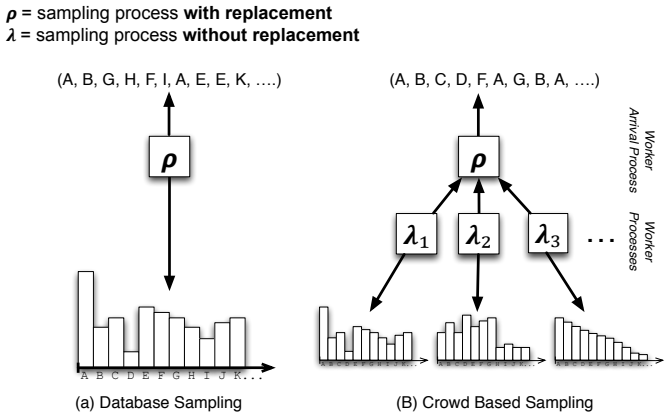


Figure 4.4: Comparison of the sampling processes, (a) assumed by traditional algorithms and (b) observed in crowd-based enumerations

4.3.3 Sampling Without Replacement and Worker Skew

We observed that when a worker submits multiple items for a set enumeration query, each answer is different from his/her previous ones. In other words, individuals are sampling without replacement from some underlying distribution that describes the likelihood of selecting each item. Of course, this behavior is beneficial with respect to the goal of acquiring all the items in the set, as low-probability items become more likely after the high-probability items have already been provided by that worker (we do not pay for duplicated work from a single worker). A negative side effect, however, is that the estimator receives less information about the relative frequency of items, and thus the skew, of the underlying data distribution. This can cause the estimator to over-predict due to the more rapid arrival of unseen items than would occur in a with-replacement sample.

Over-prediction also results when some workers complete many more HITs than others, a common occurrence on crowdsourcing platforms such as AMT; workers who do significantly more work have been called "streakers" [74]. In the two-layer sampling process, worker skew influences which worker supplies the next answer, i.e., skew in the ρ process—streakers are chosen with higher frequency. High worker skew can cause the arrival rate of unique answers to be even more rapid than that caused by sampling without replacement alone, causing the estimator to over-predict. The reasoning is intuitive: if one worker provides a majority of the answers, and he/she submits only answers he/she has not yet given, then a majority of the total answers will be his/her unique answers.

In an extreme scenario in which one worker provides all answers, the two-layer process reduces to one process sampling from one underlying distribution without replacement. In this case, completeness estimation becomes impossible because no inference can be made regarding the underlying distribution. Another extreme is if an infinite number of workers provide one answer each using the same underlying distribution, the resulting sample would correspond to the original scenario of sampling with replacement (Figure 4.4(a)). The latter is the reason why it is still possible to make estimations even in the presence of human-generated set enumerations.

To illustrate the impact on the Chao92 estimator of the number of workers participating in a crowd-based enumeration, we simulated different numbers of workers sampling from a uniform distribution over 200 items, averaging over 100 runs. Figure 4.5 depicts the values of the Chao92 estimator calculated for increasing numbers of samples for three scenarios: a with-replacement sample (equivalent to an infinite number of workers), and three or five workers each sampling without replacement from the item distribution. As expected, the with-replacement sample overes-

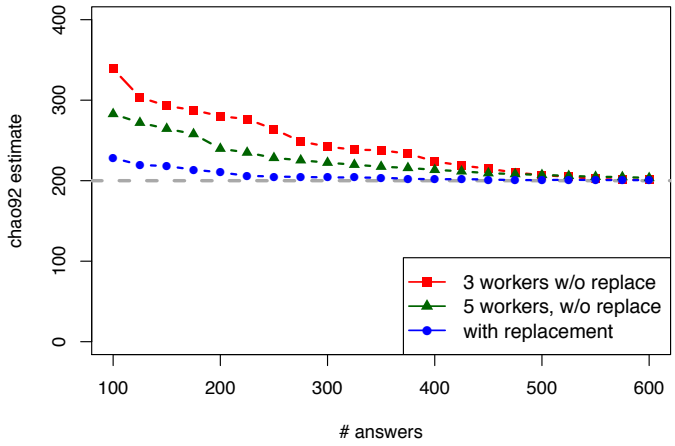


Figure 4.5: Chao92 estimator evaluated on simulated samples from uniform distribution over 200 items, with differing numbers of workers.

timates slightly because of the uniform data distribution, but quickly approaches the true value of 200. The without-replacement samples, having fewer workers, overestimate even more and remain in that state for longer.

4.3.4 Different and Skewed Data Distributions

Individual workers also may be drawing their answers from different data distributions. For example, the most likely item for one worker could be the least likely item for another. These differences could arise from varying cultural or regional biases, or alternate techniques for finding the data on the web. A mixture of multiple distributions over the same data results in a combined distribution that is "flatter" than its constituent parts, thereby becoming less skewed. In contrast, when the underlying data distribution is heavily skewed and shared amongst workers, the estimator will typically underestimate because there will not be a sufficient number of items representing the long tail of the distribution.

Figure 4.6 shows the impact of different data distributions combined with different worker skew on the Chao92 estimate. It shows four combinations: the absence/presence of worker skew (WS) and shared/different data distributions for workers (DD). For all cases, we use a power law data distribution in which the most likely item has probability p, the second-most likely has probability $p(1-p)$, etc.; we set $p = 0.03$. To simulate different data distributions, we randomly permute the original distribution for each worker.

The simulation shows that the worst scenario is characterized by a high worker skew and a single shared data distribution (WS=T and DD=F). With a shared skewed distribution, Chao92 will start out underestimating because all workers are answering with the same high-probability items. However, with high worker skew, the streaker(s) provide(s) many unique answers quickly causing many more unique items than encountered with sampling with replacement.

On the other hand, the best scenario is when there is no worker skew but there are different data distribution (WS=F and DD=T). By using different data distributions without overemphasizing a few workers, the overall sample looks more uniform, similar to Figure 4.5 with replacement, due to the flattening effect of DD on skewed data.

4.3.5 Worker Arrival

Finally, the estimate can be impacted by the arrival and departure of workers during the experiment. All workers do not necessarily provide answers during the lifetime of a query. Instead they come and go as they please. However, the estimator can be strongly impacted when streakers arrive who then suddenly dominate the total number of answers.

Figure 4.7 demonstrates the impact a single worker can have. It uses the same simulation setup as in Figure 4.6, but also simulates an additional single streaker starting at 200 HITs who continuously provides all 200 answers before anyone else has a chance to submit another answer. As the figure shows, it causes Chao92 to over-predict in all four cases. However, if workers use different data distributions the impact is not as severe. Again, this happens because DD makes the sample appear more uniformly distributed.

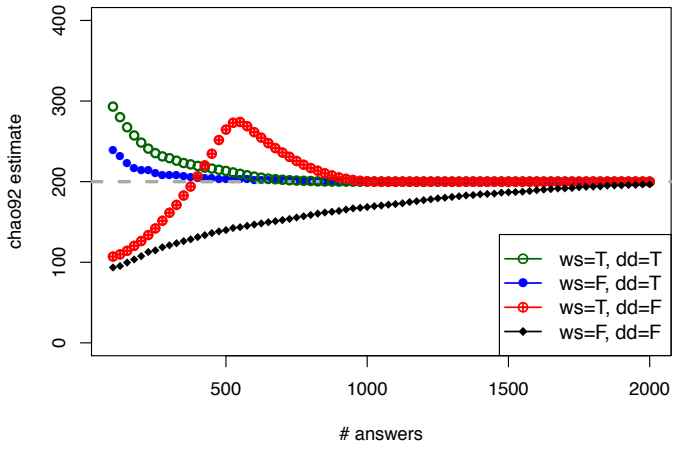


Figure 4.6: Chao92 estimator evaluated on simulated samples from a power law distribution, for combinations of worker skew (WS) and different distributions (DD).

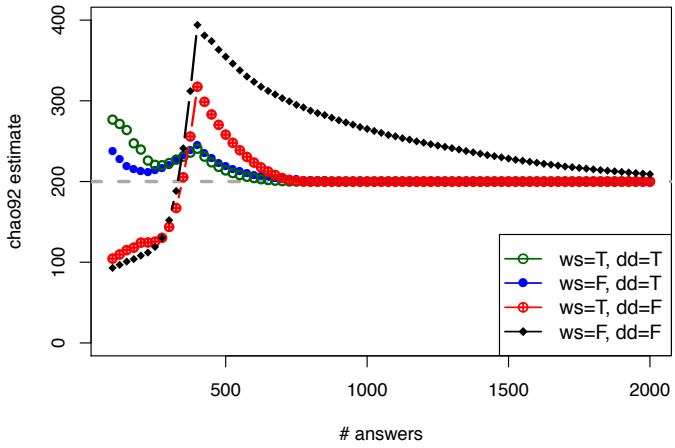


Figure 4.7: Chao92 estimator evaluated on simulated samples from a power law distribution, with additional behavior of a "streaker".

4.3.6 Discussion

Some of the behaviors that workers exhibit as they respond to enumeration queries are inherent in a marketplace such as AMT, such as how many tasks an individual worker chooses to complete. The order in which each worker provides his/her answers and how many he/she gives can depend on individual biases and preferences. The four elements of crowd behavior we outlined above (without-replacement sampling, worker skew, different distributions, and worker arrival) can each cause Chao92 to perform poorly. The most volatile of these behaviors is worker skew, particularly when the data distribution itself is skewed; a single overzealous worker could cause massive fluctuations in the estimate. Overestimation in particular is problematic because it may lead to unnecessary crowdsourcing costs in an attempt to enumerate more items of the set that do not actually exist.

Thus we want to make Chao92 more tolerant to the impact of such a streaker. We discuss our technique for a streaker-tolerant cardinality estimator next. Later, in Chapter 5, we revisit the scenario in which the estimator under-predicts due to one or more shared, highly skewed data distributions. Specifically, we note that in some cases, workers' answer sequences originate from a common list traversal. We develop a heuristic for detecting this behavior, which can be used to inform the decision to switch to an alternate data-gathering UI.

4.4 Streaker-Tolerant Completeness Estimator

Our goal is to provide a progress estimate for an open-world enumeration query based on the responses that have been gathered so far. In the previous section we demonstrated how having a crowd of humans enumerate a set creates a two-layer sampling process, and that the order in which items arrive depends heavily on different worker behaviors—which impacts the accuracy of the estimator.

In this section, we extend the Chao92 estimator to be more robust against the impact of individual workers. We focus our effort on reducing the impact of streakers and worker arrival. We do not consider the cases for which a good prediction is not possible, e.g., the extreme case of one worker providing all answers. We present our extension to Chao92 that handles streaker impact. We evaluate our technique by first proposing a new metric that incorporates notions of estimate stability and fast convergence to the true cardinality, then applying this metric to measure our techniques effectiveness on several use cases.

4.4.1 An Estimator for Crowdsourced Enumeration

The Chao92 estimator is heavily influenced by the presence of rare items in the sample; the coverage estimate $\hat{C}$ is based entirely on the percentage of singleton answers (f_1s). Recall from Section 4.3 the discussion of different crowd behaviors—many of these result in rapid arrival of previously unseen answers. When these new f_1 items appear "too quickly", the estimator interprets this as a sign the complete set size is larger than it truly is. We develop an estimator based on Chao92 that ameliorates some of the overestimation issues caused by an overabundance of f_1 answers.

Most of the dramatic overestimation occurs due to streakers, i.e., significant skew in the number of answers provided by each worker. Notably, problems occur when one or a few workers contribute substantially more answers than others, possibly also drawing answers from a different data distribution. Since other workers are not given the opportunity to provide answers that would subsequently increase the f_2s, f_3s, etc. in the sample, Chao92 predicts a total set cardinality that is too large. Thus our estimator is designed to identify any worker(s) who are outliers with respect to their contribution of unique answers in the sample (their f_1 answers).

The idea behind making the Chao92 estimator more resilient against streakers is to alter the f-statistic. The first step is to identify those workers who are "f_1 outliers". We define outlier in a traditional sense, namely, if a worker i's f_1 count $f_1(i)$ is two standard deviations outside the mean of all other workers' W f_1 counts. To avoid false negatives due to a true outlier's influence on the mean and standard deviation, both statistics are calculated without including the potential outlier's f_1 count. The f_1 count of worker i is compared to the mean $\bar{x}_i$ and the sample standard deviation $\hat{\sigma}_i$:

$$\bar{x}_i = \sum_{\forall j, j \neq i} \frac{f_1(j)}{W-1} \quad \hat{\sigma}_i = \sqrt{\sum_{\forall j, j \neq i} \frac{(f_1(j) - \bar{x}_i)^2}{W-2}} \tag{4.5}$$

We create $\tilde{f}_1$ from the original f_1 by reducing each worker i's f_1-contribution to fall within $2\hat{\sigma}_i + \bar{x}_i$:

$$\tilde{f}_1 = \sum_i min(f_1(i), 2\hat{\sigma}_i + \bar{x}_i) \tag{4.6}$$

The final estimator is similar to equation 4.4 except that it uses the $\tilde{f}_1$ statistic. For example, with a coefficient of variance $\hat{\gamma}^2 = 0$, it would simplify to:

$$\hat{N}_{crowd} = \frac{cn}{n - \sum_i min(f_1(i), 2\hat{\sigma}_i + \bar{x}_i)} \tag{4.7}$$

Figure 4.8: Example AMT task UI shown to crowd workers for enumeration queries

Although a small adjustment, $\hat{N}_{crowd}$ is more robust against the impact of streakers than the original Chao92, as we show in our evaluation next.

4.4.2 Experimental Results

We ran over 30,000 HITs on AMT for set enumeration tasks to evaluate our technique. Several `CROWD` tables we experimented with include small and large well-defined sets such as NBA teams, US states, UN member countries, as well as sets that can truly leverage human perception and experience such as indoor plants with low-light needs, restaurants in San Francisco serving scallops, slim-fit tuxedos, and ice cream flavors. Workers were paid \$0.01-\$0.05 to provide one item in the result set using the UI shown in Figure 4.8; they were allowed to complete multiple tasks if they wanted to submit more than one answer. In the remainder of this section we focus on a subset of the experiments, the use cases summarized in Table 4.1. The use cases with known cardinality and fixed membership are the US states (nine experiment runs) and UN member countries (five runs). The more open ended queries are Plants, Restaurants, Slim-fit tuxedos, and Ice cream flavors (one run each).

Query	Description
States	The names of the fifty states in the United States
UN	The names of the 192 member countries in the United Nations
Ice cream flavors	Names of flavors of ice cream
Plants	Names of indoor plants that can tolerate low-light conditions
Restaurants	Names of restaurants in San Francisco that have scallops on their menu
Slim-fit tuxedos	Names of tuxedo brands/models that have a slimmer fit

Table 4.1: Set enumeration queries used in experimental results.

Error metric

Due to the lack of a good metric to evaluate estimators with respect to stability and convergence rate, we developed an error metric Φ that captures bias (absolute distance from the true value), as well as the estimator's time to convergence and stability. The idea is to increasingly weight the magnitude of the estimator's bias as the size of the sample increases. Let N denote the known true value, and $\hat{N}_i$ denote the estimate after i samples. After n samples, Φ is defined as:

$$\Phi = \frac{\sum_{i=1}^{n} |\hat{N}_i - N|i}{\sum i} = \frac{2\sum_{i=1}^{n} |\hat{N}_i - N|i}{n(n+1)} \tag{4.8}$$

A lower Φ value means a smaller averaged bias and thus, a better estimate. The weighting renders a harsher penalty for incorrectness later on than in the beginning, in addition to penalizing an estimator that takes longer to reach the true value. The error metric also rewards estimators for staying near the true value.

Results: UN and States use cases

We first illustrate how $\hat{N}_{crowd}$ behaves for a representative set of UN member countries and US states experiments. For both experiments a UI similar to that in Figure 4.8 was shown by CrowdDB to ask for an UN member country, respectively US state, on AMT for $0.01 cents per task. Figures 4.9(a-g) show cardinality estimates as well as the Φ metric for the selected experiments. Each graph shows the Chao92 algorithm estimates (labeled "original") and the value of the error metric calculated for those estimates (Φ_{orig}), as well as the estimates and error (Φ_{new}) for the streaker-tolerant estimator (labeled "crowd estimator"). We observed that our estimate has an improvement over Chao92 for most UN experiments we performed, as Figures 4.9(a-d) show.

In the experiment run labeled UN 1, our estimates avoids the overestimation of Chao92 that occurred during the middle of the experiment. In the UN 2 experiment, one streaker dominated the total answer set at the beginning—a substantial outlier. Once his/her contribution was reduced dramatically, the remaining workers' answers had significant overlap because most were enumerating the list of nations alphabetically, resulting in a low cardinality because of the heavily skewed data distribution this scenario creates. Recall from the previous section that the expected behavior of the estimator in this case is to *under-predict*. In contrast, the third UN experiment run had several streakers at the beginning who each had very different data distributions (i.e., enumerating the list of nations from different alphabetical start points). While the heuristic helped level the f_1 contribution from these workers, overestimation still occurs due to the combined number of singleton answers from them. In a few cases, our estimator performs worse than Chao92, e.g., experimental run UN 4. Note that underestimation is expected when workers share a heavily skewed distribution; a streaker causing an estimate to be higher than it should incidentally results in a value closer to the true value.

The effect of our estimator compared to Chao92 is less significant in the States experiments, which exhibit less worker skew. Figure 4.9(e) and (f) show two US states experiments that have a moderate streaker problem and illustrate how our technique improves the prediction, whereas for

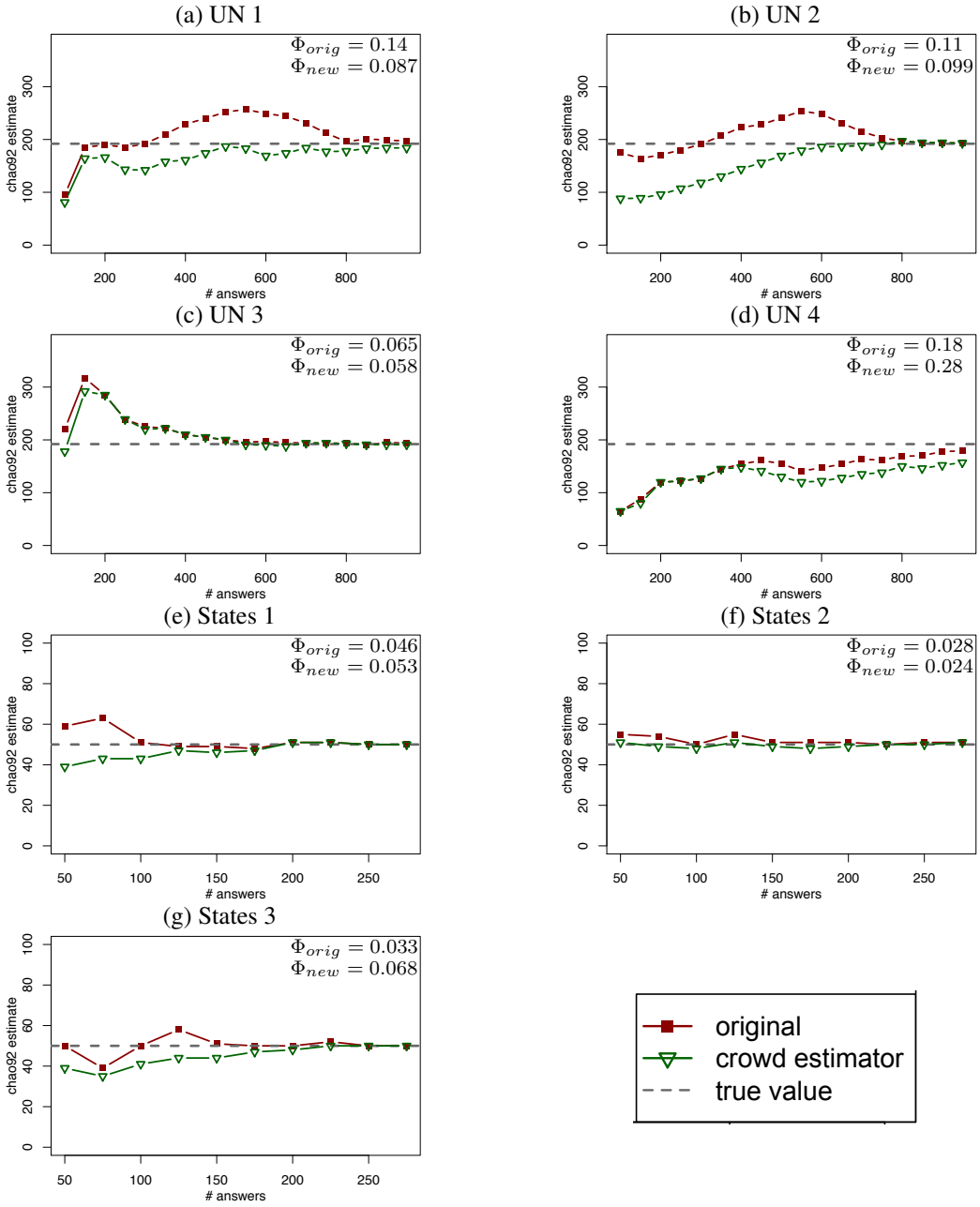


Figure 4.9: Estimator results on representative UN country and US states experiments

a third state experiment shown in Figure 4.9(g), our estimator reduces the impact of streakers but takes longer to converge for similar reasons as in the UN 4 experiment.

Results: open-ended use cases

The UN countries and US States use cases are both sets for which the true cardinality, as well as the sets' contents, is known; use cases with known cardinality allow us to evaluate the accuracy of the estimation algorithms. Here we look at use cases that are "open-ended", i.e.,, the set contents and cardinality are not known. For these results, we can only compare estimation algorithms. The open-ended use cases demonstrate several of the worker behaviors that we observed in the UN experiments as well; in particular, the presence of overzealous workers who contributed many more unique items than others.

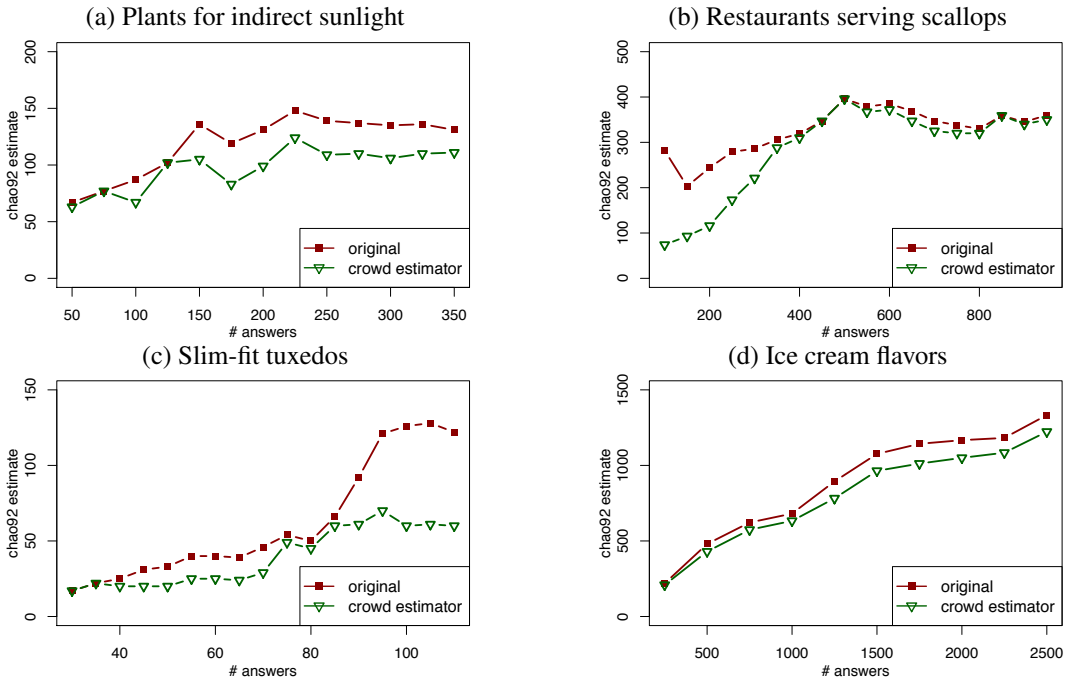


Figure 4.10: Estimator results for the open-ended cases

Figures 4.10(a-d) show the original Chao92 and our estimates for the plants, restaurants, tuxedos, and ice cream flavors experiments. In all cases, our estimator successfully reduces the impact these streakers have on the prediction of complete set cardinality. Note that we cannot evaluate the error Φ for these experiments because the true cardinality is unknown. During the plant experiment (Figure 4.10(a)), one worker from the beginning consistently contributed more unique answers than the other workers, e.g., a less well-known plant called "rabbit's foot"; many workers stuck to well-known answers (e.g., snake plant, peace lily). In contrast, in the restaurant experiment (Figure 4.10(b)) a streaker contributed many f_1 answers at the beginning, but other workers eventually provided many of those same answers. The tuxedos experiment (Figure 4.10(c)) demonstrates how a streaker who arrives later in the experiment affects the estimate, causing a sharp increase in the Chao92 estimate which is ameliorated by $\hat{N}_{crowd}$.

4.4.3 Discussion

In this section, we showed that our estimator successfully provides more accurate prediction for crowd-based set enumerations in the presence of overzealous workers (i.e., streakers). Our technique specifically tackles cardinality overestimation, which can be quite extreme and misleads the user into thinking he/she is lacking many more items in the set than he/she really is. It should be noted, however, that any heuristic, including ours, can only cope with a certain range of worker behavior that one could encounter when crowdsourcing a set. For example, if only one worker provides any answers, there is no information about the underlying data distribution for the estimator to take advantage of. On the other hand, if there are many workers producing few answers from a heavily skewed data distribution, an estimator is likely to underestimate because there will always be very few f_1 answers in the set. Most of the real experiments we ran on AMT did not fall into these extreme categories, and the heuristic is able to ameliorate the moderate impact of worker behavior on cardinality estimation.

4.5 Cost vs. Benefit: Pay-As-You-Go

For sets with finite membership, it makes sense to estimate the set size and employ the crowd to provide the complete set. However, the result set for some queries may have unbounded size, a highly skewed distribution and/or extreme worker behavior that make predicting its size nonsensical, as discussed in Section 4.3. I. J. Good, who worked with Turing on frequency estimation, stated in 1953: "I don't believe it is usually possible to estimate the number of species... but only an appropriate lower bound for that number. This is because there is nearly always a good chance that there are a very large number of extremely rare species" [64].

For these cases, it makes more sense to try to estimate the benefit of spending more money on crowdsourcing, i.e., predicting the shape of the SAC (e.g., Figure 4.1) in the near future. Eventually, the cost of getting a few more answers is prohibitively expensive or impossible and thus it makes more to sense to *pay as you go*. In this section we analyze pay-as-you-go techniques to predict this cost versus benefit tradeoff of getting more answers by expending additional effort.

4.5.1 Estimating Benefit via Sample Coverage

A query optimizer in an open-world system would want to estimate the benefit of increased crowdsourcing effort to consider the end user's quality goals. For the set enumeration query in CrowdDB, we are interested in how many more unique items would be acquired with m more HITs, given the current number of responses. For example, if we have observed 34 unique items after receiving 50 worker responses, we would like to estimate how many more unique items we would see if we issued another 50 HITs. Again, we leverage techniques from the species estimation literature, which developed techniques to evaluate the benefit of additional physical effort such as setting more animal traps. To our knowledge, this is the first time that these techniques are applied in the context of database queries.

Shen et al. [75] derive an estimator (hereafter *Shen*) for the expected number of species $\hat{N}_{Shen}$ that would be found in an increased sample of size m. The approach assumes there is an estimate of the number of unobserved elements $\hat{f}_0$ and that the unobserved elements have equal relative abundances. However, this cardinality estimate $\hat{f}_0$ can incorporate a coefficient of variance estimate (equation 4.3) to account for skew, as shown by Chao92. An estimate of the unique elements found in an increased effort of size m is:

$$\hat{N}_{Shen} = \hat{f}_0 \left[1 - \left(1 - \frac{1 - \hat{C}}{\hat{f}_0} \right)^m \right] \tag{4.9}$$

We present results based on the Chao92 estimate of $\hat{f}_0$. Our estimator $\hat{N}_{Crowd}$ is designed to reduce the impact of streakers to yield a more accurate estimate of total set size, i.e., as $m \to \infty$, however disregarding the rapid arrival rate of new times can cause local predictions with $\hat{N}_{shen}$ to under-predict. Thus we use the original Chao92 estimate for the pay-as-you-go prediction.

Note: an intuitive approach would be to extrapolate the shape of the SAC, which depicts the number of unique items as worker responses are received, as a means to estimate the benefit of additional responses. To this end, we evaluated the *spline* technique for extrapolating the curve as described in [76]. The "mean" SAC is calculated by permuting the data many times and averaging the SACs from each permutation. Afterwards, a cubic spline is fit to this smoothed version of the curve, which in turn is used for the final prediction. Unfortunately, because the spline-based estimator learns the shape of the curve only through the observed samples, it has no knowledge of expected behavior; its performance was inferior to the coverage-based estimator $\hat{N}_{Shen}$. Another technique in [76] models the "expected mean" SAC with a binomial mixture model. It performs similarly to the coverage approach.

4.5.2 Experimental Results

We evaluated the effectiveness of the $\hat{N}_{shen}$ estimator in determining how many more unique items would arrive if we crowdsource m additional worker responses; this analysis would be done after having already received n answers (HITs). Accuracy is calculated as the absolute value of the error (bias); the absolute value allows for averaging the errors. The tables in Figure 4.11 contain the errors for the queries we described earlier. For example, in the UN experiment after $n = 500$ the average error for the next $m = 50$ HITs is 1.6 (i.e., on average the prediction is off by 1.6). For some cells, we were not able to evaluate the prediction as we did not receive enough answers for at least one of the experiments. These cases are marked with a dash.

For all experiments, predictions for small m are easier since only the near future is considered, thus they tend to be more accurate. The larger the m, the further the prediction has to reach and thus the more error-prone the result, particularly if m exceeds the current HITs size n [75]). The pay-as-you-go results are also aligned with the intuition the SAC provides: at the beginning when there are few worker answers, it is fairly inexpensive to acquire new unique items. Towards the end, more unique items are harder to come by.

Error: Average UN Experiments			
m	$n = 200$	$n = 500$	$n = 800$
10	1	0.6	0
50	5	1.6	1.6
100	8.4	3	3

Error: Plant Experiment			
m	$n = 100$	$n = 200$	$n = 300$
10	2	3	0
50	7	3	2
100	7	1	-

Error: Ice Cream Experiment			
m	$n = 1$K	$n = 1.5$K	$n = 2$K
10	4	0	0
50	9	1	0
100	13	1	3

Error: Average States Experiments			
m	$n = 50$	$n = 150$	$n = 200$
10	1	0.67	0.44
50	2.6	1.8	0.78
100	5.7	2	-

Error: Restaurant Experiment			
m	$n = 200$	$n = 400$	$n = 800$
10	2	0	3
50	11	5	13
100	18	3	-

Error: Tuxedo Experiment			
m	$n = 30$	$n = 50$	$n = 70$
10	3	0	1
50	8	10	-
100	-	-	-

Figure 4.11: Pay-as-you-go cost-benefit predictions using *Shen*

Worker behavior also has an influence on the pay-as-you-go predictions. The Shen estimator tends to under-predict before the accumulation curve plateaus, as the curve is steeper than expected. This happens because workers sample without replacement—unique items appear more quickly than they would from a with-replacement sample. While minimizing all error is ideal, under-prediction is not catastrophic since the user will end up getting more bang for his/her buck than anticipated. There is also potential to use knowledge of worker skew and particularly the presence of streakers to inform the user when an under-prediction is likely. Thus $\hat{N}_{shen}$ provides a reasonable mechanism for the user to analyze the cost-benefit tradeoff of acquiring more answers in the set.

4.6 Related Work

In this chapter we focused on estimating progress towards completion of a query result set—an aspect of query quality. To our knowledge, the quality of open-ended questions posed to the crowd has not been directly addressed in literature. In contrast, techniques have been proposed for quality control for individual set elements [32, 20].

Our estimation techniques build on top of existing work on species or class estimation [17, 62, 63]. These techniques have also been used and extended in database literature for distinct value estimation. In [70], Haas et. al. survey different estimators, several of which we also investigate in this chapter. They do not use the algorithm we find superior because they observe it produced overly large estimates when used in the context of a finite population. Instead they propose a hybrid approach, choosing between the Shlosser estimator [71] and a version of the Jackknife estimator [67] they modified to suit a finite population. The Jackknife technique is used for tables in which

distinct values are uniformly distributed.

That work was extended in [72], in which Charikar et. al. propose a different hybrid approach. They note a lack of analytic guarantees on errors in previous work, and derive a lower bound on error that an estimator should achieve. They then show that their algorithm is superior to Shlosser in the non-uniform case, substituting it in the hybrid approach from [70]. Unfortunately, both the error bounds and developed estimators explicitly incorporate knowledge of the full table size—a closed-world luxury. Other database techniques include changing the sampling technique to take ad- vantage of blocks in memory, e.g., [77], or focus on distinct-value estimation in a single scan of the database [78].

Species estimation techniques were also explored for search and meta-search engines. For example, Broder et al. [79] develop an algorithm to estimate the size of any set of documents defined by certain conditions based on previously executed queries. Whereas [80] describes an algorithm to estimate the corpus size for a meta-search engine in order to better direct queries to search engines. Similar techniques are also used to measure the quality of search engines [81]. All techniques differ from those described in this chapter, as they do not consider the specific worker behavior and assume sampling with replacement.

Recent work also tries to explore species estimation techniques for the deep web [82, 83]. Again, the proposed techniques have strong assumptions regarding the sample which do not hold in the crowd setting. Some of these assumptions might not even hold in the context of the deep web. We believe that our techniques, which are more robust against biased samples, are applicable in the context of deep web search/data integration and consider it future work.

Although this work was done as part of CrowdDB [11], it could be applied to other hybrid human-machine database systems, such as Qurk [13] or Deco [50]. Both systems allow for acquiring sets from the crowd but do not yet provide any quality control mechanisms for set enumeration.

4.7 Conclusion

People are particularly well-suited for gathering new information because they have access to both real-life experience and online sources of information. Incorporating crowdsourced information into a database, however, raises questions regarding the meaning of query results without the closed-world assumption – how does one even reason about a simple `SELECT *` query? We argue that progress estimation allows the user to make sense of query results in the open world. By calculating an estimate of the expected result set size, or cardinality, for the enumeration query, an estimate of how complete the set is can be formed.

The species estimation literature provides an intriguing starting place to tackle cardinality estimation. However, applying existing estimators to sequences of responses from the crowd yields inaccurate results due to human behaviors. These estimators assume a with-replacement sample from a single item distribution, however it turns out that crowdsourced enumeration is actually a two-layer process: a with-replacement sampling process drawing from many without-replacement sampling processes over potentially different distributions.

A particularly troublesome issue is the presence of "streakers", workers who complete many more HITs than other workers, causing the estimator to wildly over-estimate cardinality. To ameliorate the problems caused by these over-zealous workers, we develop a *streaker-tolerant* estimator. Using set enumeration experiments run on AMT, we show that this estimator successfully reduces the impact of streakers and generates a more accurate estimate.

Of course, for some sets an accurate cardinality estimate is difficult; the set may be infinite, or virtually infinite, in size. Thus it makes more sense to reason about the expected benefit of asking the crowd for more responses. We again apply techniques from the species estimation literature o make this pay-as-you-go approach possible. Thus by adapting statistical techniques, we enable users to reason about query progress and cost versus benefit trade-offs in the open world.

Chapter 5

Getting it All: Worker Task Interfaces

5.1 Introduction

Thus far we have used a simple task interface for gathering answers from crowd workers, i.e., individual workers submit answers one at a time without using any knowledge of what answers have already been submitted by other workers (recall the simple UI in Figure 4.8). Using this simple interface allowed us to investigate how to reason about the quality of a result set using sampling techniques adapted from species estimation. A consequence of the simplicity is that we must pay for duplicate answers that have been submitted by different workers. Furthermore, recalling the shape of the accumulation curve exemplified by Figure 4.1, the more complete the set is, the more difficult and costly it is to find unseen items.

The question we investigate in this chapter is: can we design alternate techniques for crowd-based data collection that both (a) reduces crowdsourcing costs and (b) increases the likelihood of completing the set? For example, another possible task interface would incorporate the set of existing answers and prohibit the resubmission of answers that have already been provided, thereby saving money spent enumerating the set. Another idea is to detect in the stream of responses from the crowd when the set of items is available on a single web page; this would allow a switch to a data scraping task interface that would be more efficient.

Both of these approaches have challenges, however. The first approach prohibits an item to be given more than once, but this redundancy is exactly the feature that allows for progress estimation—recall the relative item frequency captured by the f-statistic. In the second approach, the challenge lies in detecting if the set of items might be available as a list. For use cases where one or two lists containing the full set exists, this switch could be helpful for completing the set at once. However, switching strategies for sets for which no single list exists would not make sense.

In this chapter, we describe how we developed these two techniques, addressing the challenges therein. We have designed a "Negative Suggest" user interface (UI) for interacting with the crowd. We describe the design decisions pertinent to its implementation, focusing on capturing item frequency information for cardinality estimation. We then illustrate the advantage of Negative Suggest for cutting enumeration costs with crowd experiments. We also use simulation to demonstrate

the impact of the UI on sets with varying skew, as well as to illustrate the cost versus estimation accuracy tradeoff when using different amounts of frequency statistics.

There has been work on designing crowdsourcing interfaces (see [84] for an explorative study). Most related to our Negative Suggest UI is the "block list" interface from the ESP game [7]. The UI shows *block words* the user cannot enter. Their focus is on labeling, not general information retrieval, and supports only a small set of block words. Fantasktic [85] helps users create crowdsourcing tasks, however their techniques do not address how to steer workers in certain ways, e.g., to avoid duplicate answers.

We also describe a technique to detect when workers are traversing the same list for answers. We refer to this effect as *list walking*. Detecting list walking makes it possible to change the crowdsourcing strategy: a worker can be asked to provide the full list, e.g., through the use of a browser plugin that allows him/her to scrape the data from a webpage. In cases where one or two lists containing the full set exists, this alternative interface could be helpful for retrieving all elements of the set at lower cost than the simple or Negative Suggest interfaces.

In summary, we make the following contributions in this chapter:

- We introduce and investigate alternative interfaces for gathering answers that reduce crowd costs.
- We develop a technique to retain cardinality estimation capability while reducing cost.
- We devise an algorithm to determine if workers are consulting data in lists, for which data scraping could be applied.
- We examine the effectiveness of our techniques via experiments using Amazon Mechanical Turk (AMT).

5.2 Reducing Redundancy with "Negative Suggest"

The "Negative Suggest" interface prohibits duplicate answers by allowing workers to see the contents of the set collected thus far. When using this interface, workers are indirectly interacting and competing with one another: as each worker submits a new answer, it is added to a "block list", the list of answers that cannot be submitted, in real time; thus the workers are effectively racing one another to find new answers. The challenges in designing this user interface involve how to effectively show existing answers to a worker, as well as how to mitigate frustration due to the nature of interacting with other workers.

5.2.1 Interface Design Considerations

The idea behind Negative Suggest is to interactively show workers which answers they cannot submit in real time as they are typing. As a worker types his/her next answer, existing answers that match the prefix of the worker's current attempt are displayed, providing an easy way for the worker to confirm that the answer is indeed new. Figure 5.1 shows an example of the Negative

Suggest user interface (UI) from one of our experiments run on AMT. At the top is a brief set of instructions, asking the workers to provide a new element of the set. The bar at the bottom of the task updates as the workers type, continually notifying them which answers they may not submit. To eliminate clutter in the UI, only existing answers that match the prefix of a worker's current attempt are shown. When the answer is verified as being new, the bar turns green and informs the worker that it is okay to submit the typed answer.

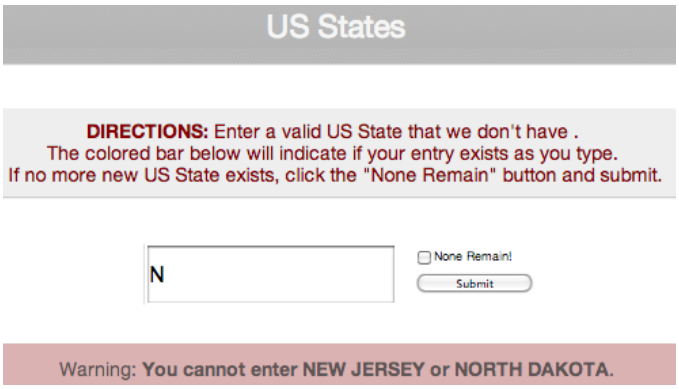


Figure 5.1: Example Negative Suggest interface that is presented to crowd workers

An important element of prohibiting existing items from being reentered is determining whether answers are actually the same, despite small typographic differences. In our experiments with enumerating the US States, we use a simple heuristic that probes existing answers as well as various aliases of these existing answers, including common abbreviations or misspellings. Furthermore, we leverage the Google Suggest API[1] to fix potential errors. Of course, these heuristics will work best on well-known terms because the reconciliation task is easier. For more open-ended queries for which reconciliation is less straightforward, entity resolution will require the help of the crowd as well. Crowd-based entity resolution is its own important area of research, see e.g., [43] and is beyond the scope of this thesis. It is future work to incorporate these approaches to assist in on-the-fly entity resolution.

Using the simple interface we examined in Chapter 4 (Figure 4.8), workers submit answers one-at-a-time with no knowledge of which answers other workers have submitted; thus there is no competition between workers to find new answers. In contrast, submitting answers in the Negative Suggest interface becomes increasingly difficult the more complete the set becomes. Some items will be easier to find than others — and some may extremely hard or near impossible to find. Workers may actually expend a significant amount of effort trying to find an unseen element of the set, and may become frustrated if they do not succeed and thus cannot be paid. To combat this frustration, we include in the interface a "none remain" checkbox, which allows workers to still be compensated for their effort to contribute more answers, even if they are unsuccessful. To prevent abuse, a worker is blocked from trying the task again if he/she has already submitted an assignment indicating he/she believes no more answers remain.

[1] http://google.com/complete/search?output=toolbar&q=

5.2.2 Duplicate Answer Submission

If we disallow workers from submitting an answer that has already been entered, we lose item frequency information and progress estimation becomes impossible. However, allowing the submission of some duplicate answers and controlling how the duplicates are used offers several advantages, namely, the set cardinality estimation as well as quality control techniques that we describe next. To this end, we introduce the notion of a *quorum*: a threshold that dictates how many times a particular item can be submitted by workers before the item is blocked from being entered again. For example, if the quorum $q = 3$, an item can be submitted up to three times (from three different workers) before it is added to the blocked list.

Recall the foundation of the non-parametric estimators such as Chao92: it is the rare items that provide the most information about the existence of unseen elements, and thus the total set size. Using a quorum, we will be able to retain frequency information in the f-statistic (e.g., if $q = 3$, we retain up to f_3), while avoiding having to pay for more than q worker submissions for any given item. Of course, the savings benefit associated with having a block list, in contrast to the simpler interface, will be influenced by the particular choice of q. Note that when $q = \infty$, the Negative Suggest interface is virtually the same as the simple interface: workers will be unaware of what answers others have submitted because no items will ever appear on the block list. In order to capture item frequency information, the quorum threshold must be set higher than $q = 1$; otherwise, there will be no way to distinguish between a rare item or a common one because the number of singleton answers f_1 will be equal to the sample size. In Section 5.2.3 we use simulation to demonstrate the tradeoff between estimate accuracy and cost savings for different quorum values.

In addition to providing the means to estimate set cardinality, allowing q item duplicates provides several quality control benefits. An important aspect of the Negative Suggest UI is that workers can immediately view the contents of the set so far, i.e., the answers provided by other workers. While this is helpful for finding an unseen item, existing items shown to workers in the task interface can also be seen as examples of *acceptable* answers. Incorrect items that are shown to workers can influence them to contribute even more errors (wrong answers can arise either from spammers or well-intentioned workers who misunderstood the directions). The quorum threshold can combat this problem by only showing in the UI answers that have been submitted at least q times. Thus we can increase the likelihood that answers appearing in the block list, and thus shown to workers, are truly members of the set. A similar technique is used in the ESP game [7]: once two individuals independently think of the same word to describe an image, that word is added to the block list.

Similarly, allowing duplicate answers can assist with the process of item correctness verification. For quality control, a worker-provided item is typically verified as correctly belonging to the set through some external verification process. In a crowd-based verification process, for example, an item may be deemed correct if at least v workers verify it. The particular value of v could be specified by the end user, who wants a certain level of assurance that items truly belong in the query result set. Using the Negative Suggest interface with a quorum q of duplicates allowed, part or all of the verification may already be accomplished (i.e., up to q of the necessary v verification

votes) because a worker who submits an answer implies that he/she believes it to be a member of the set. When $v \leq q$, no further verification cost need be spent for items that are provided q times.

5.2.3 Experimental Results

In this section, we first demonstrate the cost savings using Negative Suggest with quorum $q = 3$ versus using the simple interface from Chapter 4 with experiments run on AMT. We then perform a simulation-based sensitivity analysis to examine how the quorum size impacts potential savings. We show that reasonable estimates are obtained with low quorum, highlighting the potential of this UI got lowering the cost of crowdsourced set enumeration.

Cutting cost with Negative Suggest

One of the main motivations for the Negative Suggest interface is to reduce the expenditure on duplicate answers from the crowd. To compare completion performance between Negative Suggest and the simple interface, we look at how many worker responses were required on average to acquire the full set of US states. We focus our evaluation on the US States use case because the set members are verifiably correct or incorrect. We ran experiments on AMT using the interface shown Figure 5.1 using a quorum of $q = 3$; each experiment was run five times, with each run consisting of 300 $0.01 HITs on AMT.

Figure 5.2(a) shows how many worker answers on average it took to acquire the full set of 50 states, using the simple interface versus the Negative Suggest interface. With Negative Suggest, the set is completed with about 45% fewer worker answers. This is intuitive, as with Negative Suggest, even the most common answers cannot be submitted more than three times, while there is no such constraint in the simple interface. Notably, much of the benefit from the Negative Suggest interface occurs after the set is mostly complete. Figure 5.2(b) shows how many answers were required to reach 40 of the 50 states: Negative Suggest reaches 40 states in about 25% fewer answers. This new interface proves to be most valuable for garnering the *tail* answers, i.e., those items that are relatively unlikely to be given by workers.

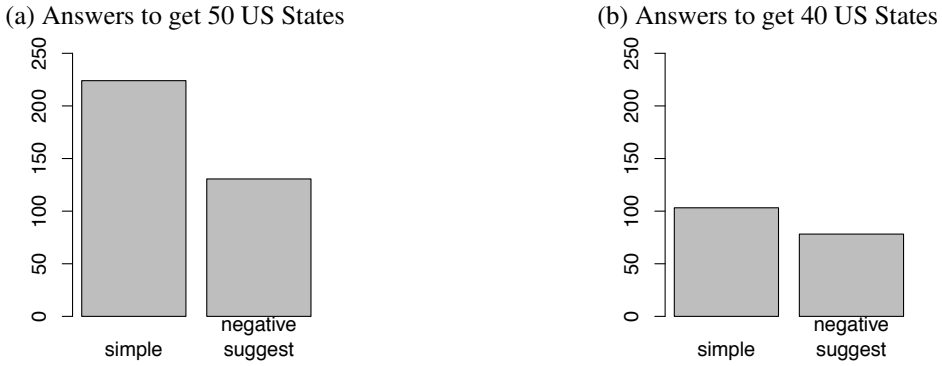


Figure 5.2: Number of worker answers before acquiring (a) 50 US States and (b) 40 US States for the simple and the Negative Suggest interfaces.

Impact of quorum parameter

In the Negative Suggest interface, using a higher quorum retains more relative frequency information, but also results in lower savings. To better understand this tradeoff, we simulate the process of workers providing items in a set. Keeping with the US States example use case, we simulated twenty-five workers (the average number of workers in US States experiments) sampling without replacement from a shared data distribution over fifty items. For a given quorum q, we discard any answer after it has appeared q times. We show results for a medium-skew distribution (CV$= 0.5$) and a high-skew distribution (CV$= 1$), based on the negative binomial distribution used in [69]. Results are averages over 100 simulations; simulations with larger set sizes had similar observations to those discussed below.

The accelerated arrival of unseen items is clearly visible in the accumulation curves for both distributions, depicted in Figure 5.3(a) and (b). In general, lower quorums result in expedited item acquisition, an effect that is magnified as the underlying data skew increases. As was seen with the US States experiment in the previous subsection, the advantage of a lower quorum threshold is most realized towards the end of the accumulation curves, where acquiring the items in the tail of the item distribution occurs.

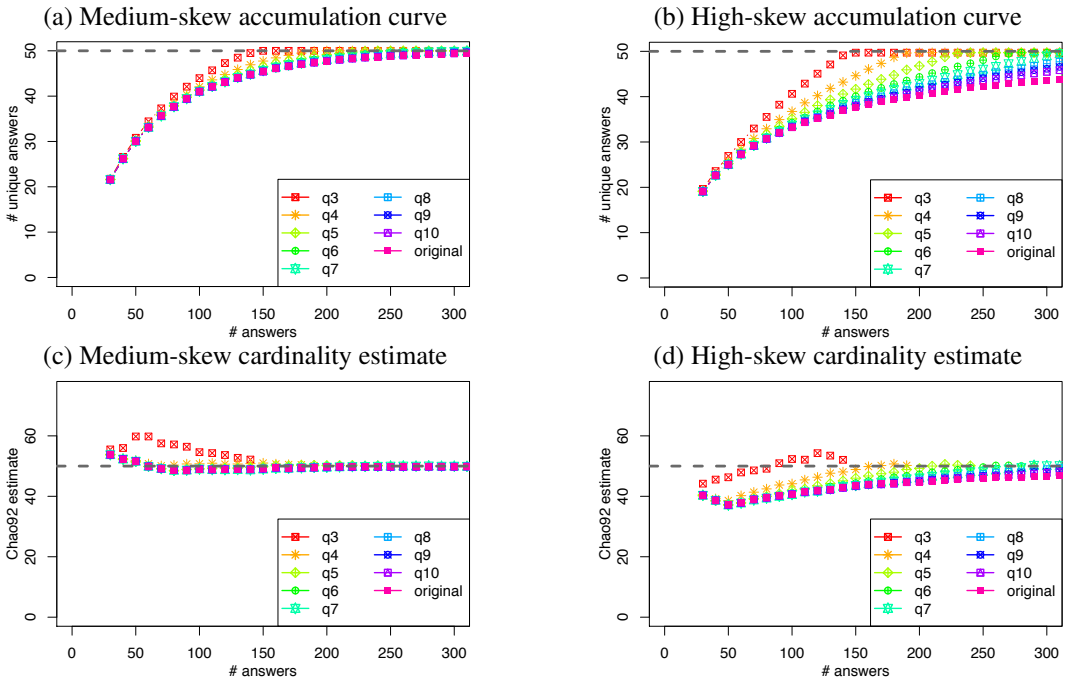


Figure 5.3: Impact of quorum for cardinality estimates and accumulation curves for the medium-skew and high-skew simulations. Quorum size increases from top to bottom in each graph

Figure 5.3(c) and (d) show the Chao92 estimator evaluated for various quorum values from the medium-skew and high-skew data distributions, respectively. In both cases, lower quorum will shift the cardinality estimate upwards because unseen items will appear more rapidly as existing

items are prohibited. A quorum $q = 3$ produces high but reasonable estimates, while $q > 3$ match increasingly closely with the estimate evaluated on the original data. Comparing Figure 5.3(c) and (d), we notice that the differences amongst the quorum values is more pronounced with higher data skew. This is intuitive: greater skew leads to the same answers being given repeatedly, and using a quorum further expedites the arrival of new items.

Disallowing the same answer from being given by more than q workers can create substantial savings when enumerating a set. In general, the more skewed the distribution, the greater the performance differences are amongst quorum values; distributions with lower skew are less sensitive to the particular value of q. While there is no optimal value for q, the results in Figure 5.3 show that there is little improvement in estimation accuracy after $q = 4$ for both the medium-skew and high-skew data distributions. The choice of the quorum value reflects the tradeoff between estimation accuracy and the advantages of completing the set sooner.

Verification threshold

Permitting duplicate answers via the quorum parameter can also aid with answer *verification*. Answer verification is a quality control technique that asks crowd workers to verify the correctness of a particular answer; the verification threshold dictates how many responses that confirm answer validity are required. For set enumeration queries, duplicate submissions of the same item can function as verification for that item—i.e., if three workers submit the same answer, then that item has also been verified three times. Naturally, the more duplicates permitted, the longer it will take for lower-probability items to reach the verification threshold.

In the previous simulation exploring the impact of the quorum parameter (Figure 5.3), an answer became a new member of the set when it is seen the first time, i.e., verification threshold $v = 1$. To illustrate the impact of a higher threshold v on the speed with which the full set is acquired, we repeat the simulation using the high-skew distribution and setting $v = 2$: an item is considered a member of the set when it has been submitted at least twice. Figure 5.4 shows the accumulation curves for this simulation, for varying quorum sizes as before. The rate at which the set is acquired is slower for all quorum values (compare with Figure 5.3(b)). However, the use of the quorum parameter still dramatically expedites the arrival rate of new items. Increasing the verification threshold widens the gaps between the curves for each quorum level with fewer answers, again making the case for a lower quorum.

5.2.4 Partitioning the Set Space

Disallowing more than a quorum number of duplicate answers using the Negative Suggest interface can lead to faster set completion than the simple interface. We can additionally use a divide-and-conquer technique, "partitioning", for further speed up. In this technique, the set space is split into disjoint *partitions*, e.g., an item's first letter; individual workers are encouraged to provide an answer within their suggested partition. The partitioning technique incorporates two main ideas. First, it guides workers towards answers that they might not have thought of on their own. It also encourages workers to provide answers that belong to different partitions of the set, thereby re-

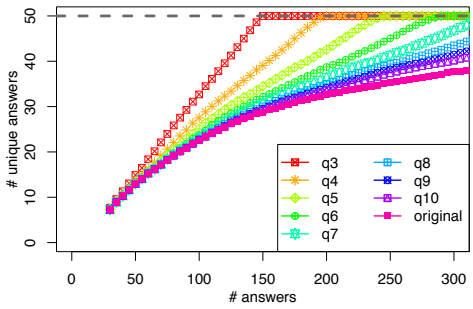


Figure 5.4: Accumulation curve when verification threshold $v = 2$ for the high-skew distribution

ducing worker contention and engendering diversity. We show in this subsection that suggesting partitions to workers can effectively increase answer diversity. While improvement in set completion time is limited when using a static partitioning scheme, we discuss the potential for a dynamic algorithm in Section 5.2.5.

A straightforward approach to divide and conquer is to construct partitions that are slices of the set space. For example, when using ranges of the item's first letter, a possible set of disjoint slices could be the seven partitions `A-D`, `E-H`, ... `U-X`, `Y-Z`. The task UI shown in Figure **??** is augmented to include a suggestion that the worker submits an answer that falls in a pre-determined partition, e.g., if the chosen partition is `A-D`, the task asks for an answer that starts with `A`,`B`,`C`, or `D`. Partitions are suggested uniformly at random, i.e., they are equally weighted amongst the tasks that are posted to the crowd platform.

The benefit of such a static partitioning scheme will be influenced by the underlying distribution that describes each item's probability of being provided by a worker. As an example, imagine highly skewed data in which the tail of the distribution comprises a small partition of size p_{small}, and a much larger partition p_{big} holds the remaining higher probability items. If p_{small} and p_{big} are given as hints at equal rates, the items in p_{small} will be provided by workers more frequently (assuming workers follow the hint) than without the use of partitioning. In other words, the observed distribution using partitioning will be less skewed and thus items in the set can be acquired at a higher rate.

We investigate the potential of partitioning with experiments enumerating the US States, run on AMT. We use the seven partitions consisting of letter ranges mentioned above; workers are given a "hint" that they should submit an answer that begins with a letter from a given partition. The partition suggestion is only a hint, there is no additional reward given, and workers are free to submit an answer outside their partition. We ran the experiment five times, with each run consisting of 300 HITs on AMT.

Figure 5.6 and 5.5 show the average number of responses in each partition after the first fifty answers using negative suggest and negative suggest plus partitioning (quorum $q = 3$), respectively; note that no suggestions are shown to workers for plain negative suggest, this data represents which partitions answers happened to fall in. We observed that when partitions are not suggested (Figure 5.5), the natural behavior for workers is to tend to submit more answers in the `A-D` and `M-P`

partitions, while `E-H` and `Q-T` have fewer answers. When workers are given suggestions based on the static partitioning scheme, the relative number of answers per partitions is more uniform, as shown in Figure 5.6.

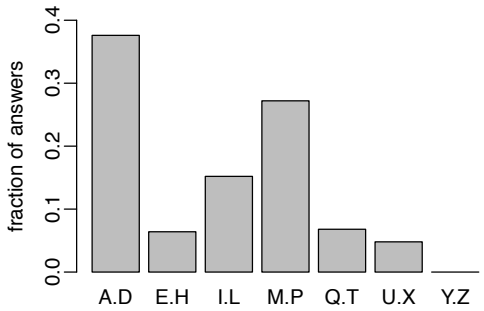


Figure 5.5: Negative Suggest without partitioning: average response distribution after 50 answers

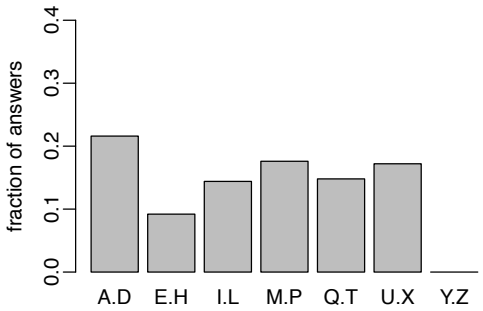


Figure 5.6: Negative Suggest with partitioning: average response distribution after 50 answers

Figure 5.7 depicts the number of answers on average until all 50 states have been acquired using the static partitioning scheme, alongside the simple interface and plain negative suggest. For the five runs of the US States experiments, we observed that the static partitioning scheme yielded only slightly faster set enumeration (less than 7%) than negative suggest alone, however we believe further improvement is possible with a dynamic partitioning scheme, discussed next. We expect performance gains to vary based on the partitioning scheme chosen as well as the set of items being enumerated.

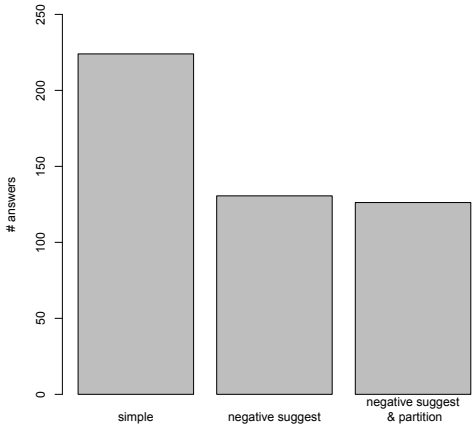


Figure 5.7: Number of worker answers before acquiring 50 US states for the simple, Negative Suggest, and Negative Suggest with static partitioning

5.2.5 Discussion

Estimating set completion progress is a powerful tool for reasoning about query results in the open world. After exploring the pure form of the problem with the simple UI for enumerating a set in the previous chapter, in this section we showed that we can retain the benefits of cardinality estimation with a different UI that reduces money spent on duplicate answers. We showed the cost savings that result when using Negative Suggest over the simple UI, using both simulation and results from experiments run on AMT. The savings will be greater for data sets with higher-skewed distributions, particularly if the skewed distribution is shared amongst the workers.

Partitioning the data set by encouraging workers to provide answers in different subsets of the set space can further expedite set completion, but gains can be heavily influenced by several aspects of the particular set being enumerated. The partitioning scheme may or may not facilitate a worker's search for a new answer. Ideally, the scheme would reflect a natural partitioning that exists either in human minds or on the web. If the partitioning scheme does not complement the set being enumerated, then crowd workers will be less likely to provide an answer in their suggested partition—thereby nullifying any advantage over negative suggest alone. Typically there will not be prior knowledge of the underlying data distribution to create a static weighted partitioning scheme. Future work includes devising a dynamic algorithm that adjusts how often particular partitions are suggested in response to the distribution of answers arriving.

5.3 List Walking

As described in Chapter 4, when workers share the same or multiple heavily skewed data distribution, particularly when workers contribute similar numbers of answers (i.e., low worker skew), the estimator may under-predict the total set size. Such a heavily skewed distribution can also occur if workers are traversing the same list for answers; indeed, we notice this behavior in some of our experiments. We refer to this effect as *list walking*.

Detecting list walking makes it possible to change the crowd-sourcing strategy to save money. In cases where one or two lists containing the full set exist, such as the UN countries, this switch could be helpful for "getting them all". However, switching strategies for sets for which no single list exists (e.g., ice cream flavors) would not make sense. Thus the goal is to detect if list walking is particularly prominent in the set of workers' answers to inform the decision regarding data gathering UIs.

In this section we devise a technique for detecting list walking based on the likelihood that multiple workers provide answers in the same exact order. We show that our technique can detect various degrees of list walking in our experiments.

5.3.1 Detecting Lists

The goal of detecting list walking is to differentiate between samples drawn from a skewed item distribution and the existence of a list, the latter of which leads to a deterministic answer sequence.

Simple approaches, such as looking for alphabetical order, finding sequences with high rank correlation or small edit-distance would either fail to detect non-alphabetical orders or disregard the case where workers return the same order simply by chance. In the rest of this section, we focus on a heuristic to determine the likelihood that a given number of workers w would respond with s answers in the exact same order.

List walking is similar to extreme skew in the item distribution; however even under the most skewed distribution, at some point (i.e., large w or large s), providing the exact same sequence of answers will be highly unlikely. Our heuristic determines the probability that multiple workers would give the same answer order if they were really sampling from the same item distribution. Once this probability drops below a particular threshold (we use 0.01), we conclude that list walking is likely to be present in the answers. We also consider cases of list walking with different offsets (i.e., both workers start from the fifth item on the list), but we do not consider approximate matches which may happen if a worker skips items on the list. Approximate matches in answer order may make the sample more random and hence more desirable for estimation purposes.

Preliminary setup: binomial distribution

Let W be the total number of workers who have provided answer sequences of length s or more. Among these, let w be the number of workers who have the same sequence of answers with length s starting at the same offset o in common. We refer to this sequence as the *target sequence* α of length s, which itself is composed of the individual answers α_i at every position i starting with offset o ($\alpha = \{\alpha_{o+1}, \ldots, \alpha_{o+s}\}$). If p_α is the probability of observing that sequence from some worker, we are interested in the probability that w out of W total workers would have that sequence. This probability can be expressed using the binomial distribution: W corresponds to the number of trials and w represents the number of successes, with probability mass function (PMF):

$$Pr(w; W, p_\alpha) = \binom{W}{w} p_\alpha^w (1 - p_\alpha)^{W-w} \tag{5.1}$$

Note that the combinatorial factor captures the likelihood of having w workers sharing the given sequence by chance just because there are many workers W. In our scenario, we do not necessarily care about the probability of exactly w workers providing the same sequence, but rather the probability of w *or more* workers with the same answer sequence:

$$Pr_{\geq}(w; W, p_\alpha) = 1 - \sum_{i=0}^{w-1} \binom{W}{i} p_\alpha^i (1 - p_\alpha)^{W-i} \tag{5.2}$$

The probability in equation 5.2 determines if the target sequence shared among w out of W workers is likely caused by list walking. We now discuss p_α, the probability of observing a particular target sequence α of length s.

Defining the probability of a target sequence

Not all workers use the same list, or use the same order to walk through the list, so we want p_α to reflect the observed answer sequences from workers. We do this by estimating the probability

$p_\alpha(i)$ of encountering answer α_i in the i^{th} position of the target sequence by the fraction of times this answer appears in the i^{th} position among all W answers. Let $r(i)$ be the number of times answer α_i appears in the i^{th} position among all the sequences W being compared, $p_\alpha(i)$ is then defined as r_i/W. For example, if the target sequence α starting at offset o is "A,B,C" and the first answers for four workers are "A","A","A", and "B", respectively, r_{o+1}/W would be $3/4$. Now the probability of seeing α is a product of the probabilities of observing α_{o+1}, then α_{o+2}, etc.

$$p_\alpha = \prod_{i=o}^{o+s} \frac{r_i}{W} \tag{5.3}$$

Relying solely on the data in this manner could lead to false negatives in the extreme case where $w = W$, i.e., where all workers use the same target sequence. Note that in this case p_α attains the maximum possible value of 1. As a result, p_α will be greater than any threshold we pick. We need to incorporate *both* the true data via r_i/W as well as a pessimistic belief of the underlying skew. As a pessimistic prior, we choose the highly skewed Gray's self-similar distribution [86], often used for the 80/20 rule. Only if we find a sequence which can not be explained (with more than 1% chance) with the 80/20 distribution, we believe we have encountered list walking. Assuming a high skew distribution is conservative because it is more likely that workers will answer in the same order if they were truly sampling than with, say, a uniform distribution. The self-similar distribution with $h = 0.2$ is beneficial for our analysis because when sampling without replacement, the most likely item has 80% $(1 - h = 0.8)$ chance of being selected and, once that item is selected and removed, the next most likely item has an 80% chance as well, and so on.

As a first step, we assume that the target sequence follows the self-similar distribution exactly by always choosing the most likely sequence. In this case α is simply a concatenation of the most likely answer, followed by the second most likely answer, and so on. Hence the likelihood of selecting this sequence under our prior belief is $(1 - h)^s$ and the likelihood that a set of w workers select this same sequence is:

$$(1 - h)^{sw} \tag{5.4}$$

Note that this probability does not calculate the probability of having *any given* sequence of length s shared among w workers; instead it represents the likelihood of having the most likely sequence in common. Incorporating the probability of all sequences of length s would be the sum of the probabilities of each sequence order, i.e., the most likely sequence plus the second most likely sequence, etc. We investigate that scenario in the next subsection.

To combine the distribution derived from data and our prior belief in the maximum skew, we use a smoothing factor β to shift the emphasis from the data to the distribution; higher values of β put more emphasis on the data. Using β to combine equation 5.3 with equation 5.4, we yield the probability of having the target sequence α (of length s) in common:

$$p_\alpha = \prod_{i=1}^{s} \left(\beta \frac{r_i}{W} + (1 - \beta)(1 - h) \right) \tag{5.5}$$

If $\beta = 1$, p_α only incorporates the frequency information from the data, so if all workers are walking down the same list, then the probability in equation 5.5 would be 1 (thus not detecting

the list use). Note also that when $\beta = 0$, p_α just uses the $80 - 20$ distribution and will reduce to $(1 - h)^s$.We demonstrate the effect of different values of β in the experimental results.

Defining another p_α

Recall that in the previous subsection, we only calculated the probability of the workers' sharing the single most likely sequence. In addition, we can also compute the probability of any sequence α under the 80-20 rule. Let $pos(\alpha_i)$ denote the position of answer α_i in the ranking from the 80-20 rule as $\prod_{j=1}^{s} p'_\alpha(j)$, where

$$p'_\alpha(j) = \frac{(1-h)h^{pos(\alpha_j)-1}}{1 - \sum_{i=1}^{j-1}(1-h)h^{pos(\alpha_i)-1}} \tag{5.6}$$

The p_α in equation 5.5 always assumes that the target sequence is the most likely sequence, which may not be true. As an alternative version, we can in fact evaluate the probability of observing the target sequence under the $80 - 20$ rule, which incorporates more information from the observed data. We first calculate the most likely sequence based on the W sequences (see Algorithm 1). The output of Algorithm 1 is an order of items, which we assume to be the "real" self-similar distribution and consider it as part of equation 5.5. Now using the probability of sequence α under the $80 - 20$ rule from equation 5.6, we obtain the following.

$$q_\alpha = \prod_{i=1}^{s} \left(\beta \frac{r_i}{W} + (1-\beta) p'_\alpha(i) \right) \tag{5.7}$$

Here $pos(\alpha_i)$ in equation 5.6 is the position in which the answer in the ith position of the target sequence (i.e. α_i) appears in the sequence 'winner' outputted by Algorithm 1.

Algorithm 1 Voted Sequence

Given: $sequences \leftarrow sequence_1 \ldots sequence_W$ of length s
$winners \leftarrow list()$
$carryovers \leftarrow list()$
for $i = 1 \rightarrow s$ **do**
  $answers_i \leftarrow sequence_1[i] \ldots sequence_W[i]$
  $possible_i \leftarrow answers_i + carryovers$
  $winner_i \leftarrow$ {the answer that appears most in $possible_i$}
  $append(winners, winner_i)$
  $carryovers \leftarrow possible_i - winners$
end for

Using equation 5.5 (or equation 5.7) as p_α in equation 5.2 yields the probability that workers could share the sequence α.

5.3.2 Experimental Results

To apply our heuristic to the AMT experiments, we prune the search space by using a window size s of at least 5 over the answers per worker. That is, for a sequence of answers of at least size s that have more than one worker in common, we compute the probability of that sequence using equation 5.2. If the probability falls below the threshold 0.01, we consider the sequence as being from a list. Our version of windowing ensures that we compare sequences that start at the same offset o across all workers, as equation 5.5 leverages the relative order that workers provide answers. We check for list walking over time (number of HITs) and quantify how many of the observed HITs were part of a list; this gives a sense of the impact of list use in the experiment.

For the experimental results, we revisit the use cases from Chapter 4 and summarized in Table 4.1. Figure 5.8 shows the number of affected HITs for several runs of the States and UN

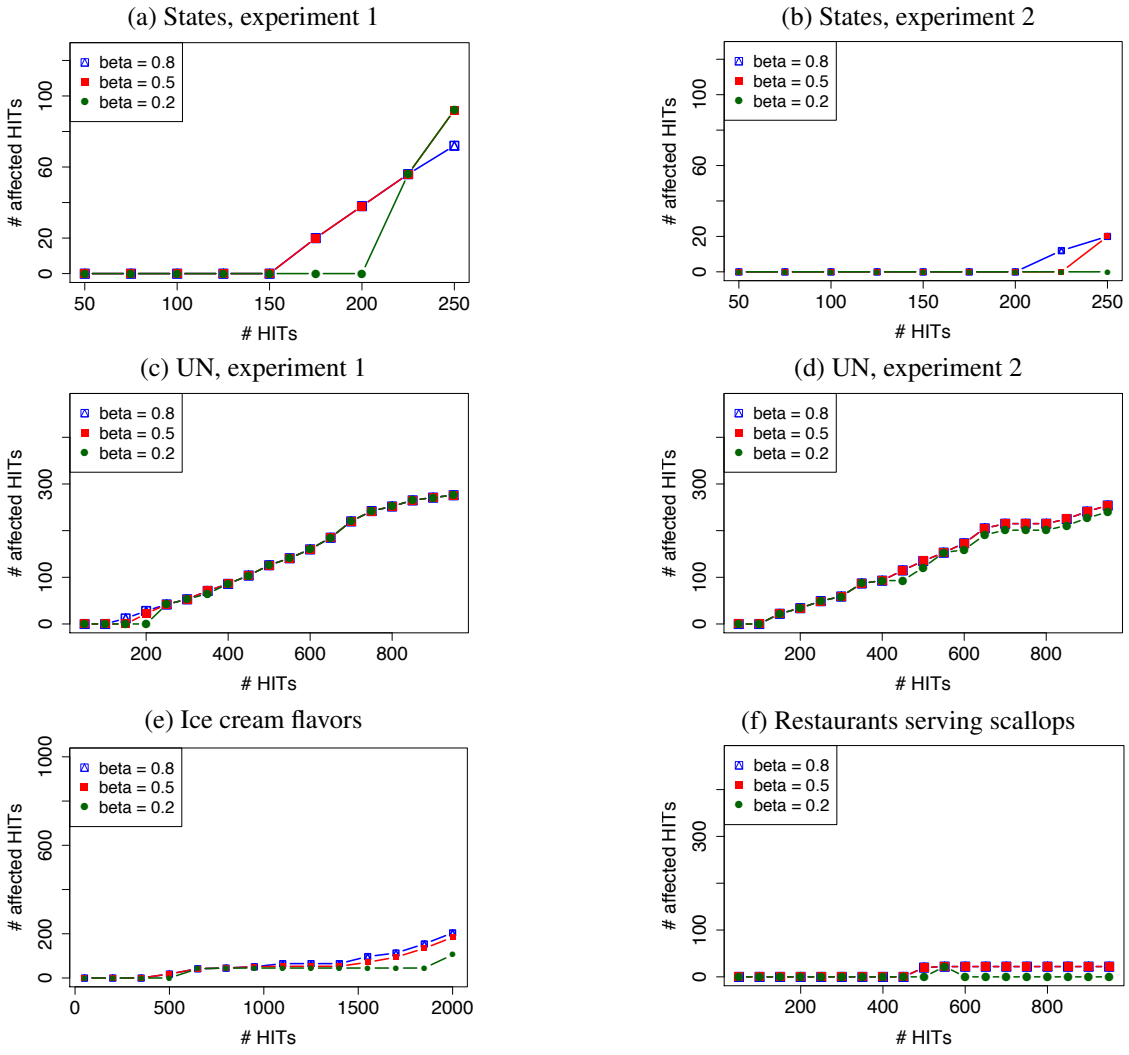


Figure 5.8: HITs detected as list-walking for different experiments

experiments, as well as the Ice cream flavors and Restaurants experiments. As in the previous chapter, we use representative single runs as opposed to averages to better visualize the effect that a user of the system would observe. The lines correspond to using equation 5.5 with different β values $0.2, 0.5, 0.8$; we elide results with equation 5.7 as we did not see a significant difference in the results.

In general, the differences between β values are minor; however we do note that at times lower values detect fewer lists or need more HITs to detect lists. Other than those shown here, the States experiments showed no list walking. While there are webpages that show the list of US states, we posit that workers did not find thinking of the states' names too difficult to warrant consulting the web. All runs of the UN experiment exhibited some list use, with the list being the alphabetical list of countries that can be found online. However, we also notice that in one of the experiments a few workers submitted answers from the list in reverse alphabetical order.

Interestingly, we detect some list walking in the Ice cream flavors experiment. While searching for the original sources, we found a few lists of ice cream flavors on the web—several are menus for existing ice cream parlors such as the "Penn State Creamery" and "Frederick's Ice Cream". Other sources include a list of the "15 most popular ice cream flavors" as well as forum thread on ChaCha.com discussing ice cream flavors. We detected one short list in the Restaurants experiment, however we were unable to determine its origin. We did not detect any list walking in the Slim-fit tuxedos or Plants experiments.

Overall, our results show that our heuristic is able to detect when multiple workers are consulting the same list and how prevalent list walking is. For example, it reports that for run 2 of the UN experiment around 20-25% of all HITs are impacted by list walking. Whereas for the Ice cream flavors experiment less than 10% are impacted. As expected, the more practical use cases for a crowd-based set enumeration strategy do not have substantial list walking.

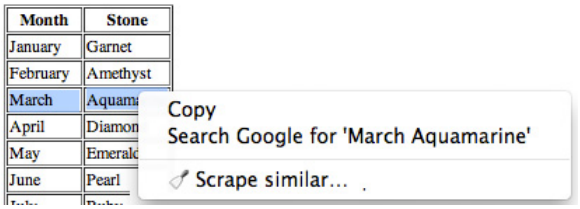
5.3.3 Scraper User Interface

We can use list walking detection as an indicator to switch to an alternative interface for collecting data: scraping information from the web. Scraping makes sense when the entire list can be found in one or a couple of locations. Rather than paying workers to simply copy the items one by one, we can ask them to use existing tools for scraping structured information.

To test the feasibility of this crowd-based scraping approach, we experimented with asking workers on AMT to scrape data using a web browser extension[2]. An example of using this particular tool is shown in Figure 5.9, in which the worker is given an HTML page with birthstone information. Workers simply highlight a section of the webpage that includes the structured data, and right-clicks (Figure 5.9(a)) to bring up the plugin UI (Figure 5.9(b)). The plugin automatically detects the data that can be exported using an XPath expression over the webpage HTML. AMT workers successfully installed the browser extension and performed web scraping for several example data sets, such as the birthstone list shown in Figure 5.9(a), illustrating the potential of the data scraping approach.

[2]http://mnmldave.github.io/scraper

(a) Scraper context menu in web browser



(b) Plugin UI

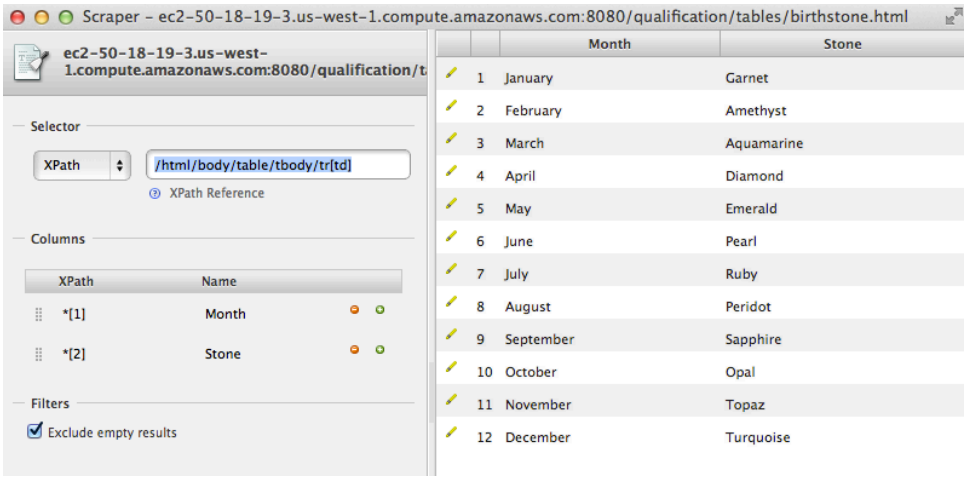


Figure 5.9: Scraper context menu and plugin UI

5.4 Conclusion

In Chapter 4, we showed that statistical techniques can be used to estimate the cardinality of a crowd-based set enumeration query—thereby providing a way to reason about query result quality in the open world. As responses from workers arrive one-by-one, we can use information about the amount of item duplication to infer how large the set will eventually be when complete. Unfortunately, because each worker response incurs a cost, duplicate items further increase the cost of enumerating the set.

In this chapter, we show that it is possible to reduce the cost of enumeration queries using different approaches and interfaces to perform crowd-based data collection. Rather than presenting crowd workers with a simple task interface that accepts a single response regardless of what other workers have submitted, we develop the "Negative Suggest" interface that disallows duplicate responses over a threshold quantity. Experimental results using the US States use case, as well as simulation results, demonstrate that Negative Suggest effectively reduces the number of crowd answers needed to complete the set. In particular, the interface expedites the acquisition of the set's lower-probability items. By retaining a small amount of redundancy, we can preserve the item frequency information needed to estimate cardinality, while reducing the overall cost to enumerate

the set.

We also develop a technique to detect the amount of "list walking" taking place, i.e., when workers are providing their answers based on a list of items. Based on the prevalence of list walking in the stream of worker responses, it may make sense to switch to a task interface which asks a worker to scrape the data from the webpage that contains the full set. We describe the results of experiments run on AMT in which workers successfully scraped structured information from the web using a browser extension. By changing the task interface that workers use for set enumeration queries, we can reduce the cost of crowdsourced data collection.

Chapter 6

Crowd-Supported Search

6.1 Introduction

In the previous chapter, we developed techniques to reduce the cost of crowd-based set enumeration by using different task interfaces. Another strategy for lowering crowdsourcing costs is to incorporate automated systems into the data collection process. Existing automated search systems may be a good starting place for some data collection queries, particularly those queries for which there is a dedicated service for retrieving items in the desired data set, such as Google.com or Bing.com for images and Amazon.com for products. Rather than having crowd workers perform redundant searches, it makes more sense to first query the automated system to create a set of potentially correct results and then use human input for the validation of those results.

However, search systems can still have difficulty with particular types of queries. For example, queries that require:

- *subjective, qualitative, and/or visual analysis*. For example, the image search query "confused people using a computer alone in the picture". A search engine may incorrectly filter images due to the term "alone", producing fewer correct results than a search for just "confused people using a computer".

- *consultation with outside resource(s) to decide if a given result satisfies the query*. For example, a restaurant search engine sent a query for "Hawaiian restaurant serving poke" may be able to interpret the request for Hawaiian food, but may not know which restaurants currently have poke on the menu.

- *parsing of a description, possibly to infer if a result's properties match the query*. For example, product search on Amazon.com for "brown boots with buckle heel height 1-2 inches" does not yield results with the requested heel height.

When the user attempts to query an automated search system with these types of queries, he/she may receive few, if any, relevant results. This happens because the system cannot interpret, or misapplies, some of the search terms, which can actually cause correct results to be filtered out.

For these types of queries, we want to separate out the parts of the query that the automated system can handle from the ones that are better suited for *human computation*.

However, given an ad hoc query, we do not know a priori which search terms will be difficult for the automated search system to process. To address this issue, we can query the system (without cost to the user) with various reformulations of the query using different subsets of the original search terms. This process will yield a set of potentially correct results that can then be evaluated by a human to determine which results actually belong in the final query result. Rather than the end user filtering through a set of results that could be very large, particularly if the user has many queries[1], we can leverage paid crowdsourcing for these human computation tasks.

Enhancing current search capabilities with the crowd combines the best features of humans and machines, an idea that is also espoused in hybrid human/machine database systems [11, 13, 50], described in Chapter 3, as well as in recent work in web search, e.g., supporting rich search queries [87] or displaying user-friendly search result summaries in-line with search results [44]. In our scenario, we aim to take advantage of free automated processing to gather potential query results before turning to paid crowd processing to produce the final output. Of course, paying crowd workers to evaluate all the results from the query reformulations could be prohibitively expensive, so we must be judicious with which results are selected for crowd processing.

In this chapter, we devise an algorithm for determining which of these results should be sent to the crowd for evaluation. We leverage ideas from the multi-armed bandit (MAB) problem from machine learning, in which an algorithm seeks to determine the best strategy from a set of possible strategies. In our scenario, the set of possible strategies is the set of query reformulations. Complications arise, however, due to the fact that many search engines produce results that are ordered by relevance, i.e., ranked. Ranked data means that the quality of the search results decreases with rank. Additionally, search results generated using a subset of the query's search terms are not necessarily ranked with respect to the full query. These properties of ranked search results complicate the decision of which search results to send to the crowd because the best set of results may come from *several* of the query reformulations' results. Thus we adapt an MAB algorithm to address these challenges.

The algorithm we propose, *crowd-bandit*, adapts the traditional MAB approach in two main ways: 1) it leverages human computation to help filter the query reformulations considered, 2) it adjusts the quality-assessment of query reformulations as it is processing results. To motivate our algorithm, we first demonstrate the pitfalls of alternate approaches—highlighting the particular challenges of ranked data. We then evaluate the effectiveness of our algorithm with experiments using image search queries processed using the crowd via Amazon's Mechanical Turk crowdsourcing marketplace. We also demonstrate that the crowd-bandit algorithm is effective for search results that are unordered, or set-oriented.

In summary, the contributions of this chapter are:

- We formally define the problem of crowd-supported search, and demonstrate the challenges raised by ranked data.

[1]For example, the user may be looking for several images to use in presentation slides

- We develop an algorithm for deciding which search results to send to the crowd to process, adapting work from the multi-armed bandit literature.
- We propose and develop algorithms for pay-as-you-go cost prediction.
- We provide a detailed evaluation using image search queries and human computation using Amazon's Mechanical Turk crowdsourcing marketplace.

6.2 Human-Machine Search Query Processing

Certain parts of a search query can be difficult for automated systems to process, particularly when they require human reasoning. We propose a two-stage query processing strategy to tackle this issue. First, we use the system to generate many sets of potentially correct results using query reformulations derived from the original search terms, incurring no cost to the user. In the second stage, the crowd is used to filter those results to form the final output that is returned to the user.

In this section we first establish terminology and describe crowd-supported search in more detail. We present a formal description of the problem and its constraints, and discuss why it is difficult—motivating the need for a heuristic algorithm.

6.2.1 Process Pipeline and Terminology

For the space of problems we consider in this chapter, a user query consists of an *entity* and a set of properties, or *predicates*, that must be true of that entity. For example, in the previously mentioned image search query, "confused people using a computer alone", the entity is "people" and the three conjunctive predicates are (1) confused, (2) using a computer, and (3) alone. In this work we assume that the set of predicates P is known.

Recall that for a given query, we do not know a priori which combination of predicates will yield high or low quality search results when used to query the automated search system. Thus we investigate all possible combinations by sending each as search terms to the automated system. Figure 6.1 depicts our two-stage process. As shown in the box labeled "Reformulation", given a set of predicates P, we form $2^{|P|}$ of these combinations. Querying the search system with each of the combinations yields $2^{|P|}$ corresponding *candidate results sets*; i.e., there is one set of candidate results generated by each predicate combination. Each set of candidate results consists of one or more *items*. These candidate results sets are the input to the box labeled "Selection Algorithm", which is responsible for choosing items from one or more of the candidate results sets, forming the *answer set*. The answer set is processed by the crowd in order to generate the final query result for the user. Devising an algorithm for choosing the best answer set is the challenge we address in this chapter.

For example, suppose there is a crowdsourcing budget to process 10 items. The selection algorithm could choose 3 items from the candidate results from predicate combination K_1, 7 items from combination K_2 (and thus 0 items from the other combinations). These 10 items are sent to

the crowd to determine which are correct, i.e., which satisfy all predicates in the user's original query. The correct items are then returned to the user.

While for reasonable sizes of P the number of generated candidate results sets may be manageable, paying crowd workers to evaluate each item from each candidate results set could grow prohibitively expensive. Instead we must be judicious with which items we send to the crowd. We do so by pruning the space of all candidate results such that we only ask the crowd to process items that are likely to match the query. We describe this problem more formally next.

6.2.2 The Predicate Combination Search Problem

In our model, the user provides a query consisting of a set of conjunctive predicates P. Each possible combination of these predicates can be used as search terms to query an automated search system. Thus each combination yields a set of candidate results. The goal is to produce a final query result with the highest quality, i.e., maximize the number of items that match *all* predicates.

We are given a crowdsourcing budget with which we can process a fixed number of items. We want to know how many items from each set of candidate results we should select to send to the crowd such that the final query result has the highest number of correct items. This goal can be formalized as a search problem:

Given a set of conjunctive predicates P, there are a set of predicate combinations $K = \mathcal{P}(P)$. Let the set of functions F represent the processing done by the automated search system, where each function $f_i \in F$ corresponds to processing combination K_i. We denote the amount paid to crowd workers to evaluate one item b. For simplicity, later in our experimental results we assume it costs one budget unit to evaluate the correctness of one item.

We generate a set of candidate results sets $S = \{S_1, \ldots, S_{|K|}\}$ from processing K with the set of functions $F = \{f_1 \ldots f_{|K|}\} : K \rightarrow S$.

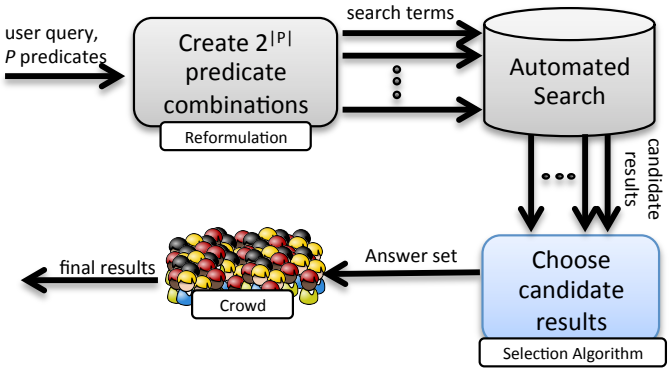


Figure 6.1: Given a query with a set of predicates P, we generate $2^{|P|}$ search queries against an automated system. These searches yield $2^{|P|}$ sets of candidate results. An algorithm decides which candidate results to send for crowd processing, returning the final query result to the user.

DEFINITION 1 *An* answer set A *is a subset of items chosen from the union of the sets* $S = \{S_1, \ldots, S_{|K|}\}$.

We define two functions on an answer set:

- $score(A) : A \rightarrow \mathcal{N}_0 = \sum_{i=1}^{|A|} c_i$, where $c_i \in \{0, 1\}$ indicates if the ith item a_i satisfies all predicates in P.
- $cost(A) : A \rightarrow \mathcal{N}_0 = b|A|$, where b is the cost of processing one item.

DEFINITION 2 *Given a set of conjunctive predicates* P *and a total budget* B*, the* Predicate Combination Search Problem *is to find an answer set* A *such that* $cost(A) \leq B$ *and the expected* $score(A)$ *is maximized.*

6.2.3 Problem Variants

The predicate combination search problem has four variants, based on properties of (1) the candidate result sets and (2) the final query result. Depending on the type of automated search system used, the candidate result sets generated from each of the predicate combinations may be unordered or ranked. The former case, which we call *set-oriented*, could be produced by a database system or a search engine that returns results ordered by something other than relevance. In the *ranked* case, results are ordered by relevance to the search terms; this is the nature of results produced by many search engines. Similarly, the final query result could be ranked or unordered; a ranked result would order the results by how well each item satisfies the specified predicates.

In this chapter, we focus on two of the four variants: we seek an unordered final query result set and we investigate both unordered (set-oriented) and ordered (ranked) candidate results. As we describe next, the ranked variant is a more challenging problem and thus the majority of this chapter addresses that case. We show later that the techniques developed for the ranked variant can also be applied to the set-oriented case.

We focus on unordered final results because we believe this is practical for when a set of items (not necessarily ordered) is desired: perhaps as training data for a classifier, or if the user wants to choose from a set of viable options. For example, some images may match better with the color theme or layout of the user's powerpoint slides. Or, in product search for boots, the user may want a set of boots to consider trying on for comfort and look. A natural extension would be to investigate producing a ranked list of query results, in which the results are ordered by how well they match the query predicates. In this case, a quality score would need to be determined for each item the crowd evaluates, e.g., a value between $0 - 1$.

6.2.4 The Optimal Answer Set

The optimal answer set can be defined differently depending on whether the candidate results produced by the functions F are set-oriented or ranked. In the set-oriented scenario, the likelihood that one randomly chosen item from a candidate results set satisfies all predicates is equal to that

of any other randomly chosen item. Thus if each set S_i has an [unknown] associated correctness probability p_i, the optimal way to spend a crowdsourcing budget B is to take a random sample of size B/b from the set with the highest p_i. More formally,

DEFINITION 3 *For set-oriented candidate results, the optimal answer set A_{set} is a random subset of items from the set S_m, where $m = \arg\max_i p_i$.*

Figure 6.2 shows an example; the highlighted region corresponds to an example answer set. Each of the predicate combinations K has a corresponding set of items, some of which may be correct with respect to all the predicates (green, light-colored balls) and some may be incorrect (red, dark-colored balls). An answer set is random subset of one of the sets of items; in this case the highlighted region corresponding to the answer set has three items, two of which are correct.

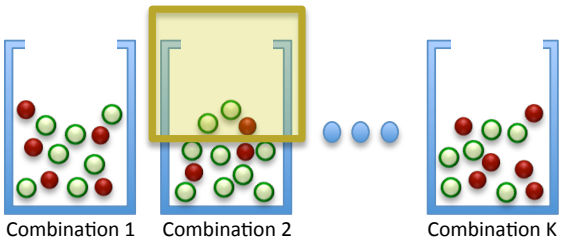


Figure 6.2: Example answer set for the set-oriented, or unordered, variant of the predicate search problem.

In the ranked scenario, the rank of an item in the candidate results set should be correlated with the likelihood of that item being correct. Thus it makes more sense to take the "top" of each candidate results set, where the top could be defined differently for each results set:

DEFINITION 4 *For ranked candidate results sets, an answer set A_{ranked} is the union of items from the sets $\hat{S} = \{\hat{S}_1, \ldots, \hat{S}_{|K|}\}$, where each $\hat{S}_i$ contains the first l_i items from the ordered set S_i, for some $l_i \in \{0, 1, \ldots, |S_i|\}$.*

Note that if $l_i = 0$, then the set S_i contributes no items to A. An algorithm must decide how many items to take from each set (the l_i for each S_i). The optimal answer set corresponds to finding the optimal setting of each l_i.

Figure 6.3 depicts the definition of an answer set for the ranked version: now we have K *ordered* candidate result sets, and our highlighted answer set selects some number of items from the top of each set. In this case, it takes the first two items from the first combination and one item from the second combination (i.e., $l_1 = 2$, $l_2 = 1$). Note this implies that $l_{\forall i \neq 1,2} = 0$.

In practice, the candidate results sets are not guaranteed to be precisely ordered with respect to all predicates in P. One reason for this is that search engine ranking algorithms are imperfect. Furthermore, the candidate results sets are ordered by only the specific subset of the predicates used to generate them. This particular nature of the ranking poses challenges to finding a solution to the predicate combination search problem, which we discuss in detail in the next subsection.

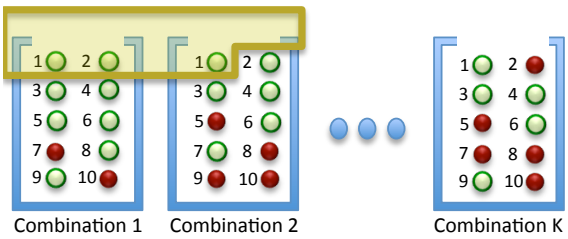


Figure 6.3: Example answer set for the ranked, or ordered, variant of the predicate search problem

6.2.5 Need for a Heuristic

Unfortunately, determining the optimal solution would necessitate a brute force search through the $2^{|P|}$ predicate combination space. This is because we cannot leverage the combinations' relationship with one another to prune the search space.

Of course, a brute force exploration of the whole space is also infeasible because we do not know a priori which candidate results satisfy the query; learning these values would incur crowdsourcing costs. Instead we need a heuristic algorithm for generating a final query result set through intelligent exploration of the candidate results sets. We describe next the challenges in devising an exploration algorithm, focusing on the more challenging version of the problem that deals with ranked candidate results sets.

6.3 Challenges of Ranked Candidate Results

In the ranked search problem, K predicate combinations are each processed with an automated system that returns a list of candidate results in which items are ordered by relevance, e.g., the output of search engines. Our goal is to intelligently prune the space of candidate results generated by the predicate combinations, in order to maximize the number of relevant items delivered to the user.

6.3.1 Image Search Queries

As described in the introduction, one of our motivating use cases is image search. We will use this example use case throughout the rest of the chapter to both illustrate the challenges posed by ranked candidate results, as well as to present performance results in our experimental section. Table 6.1 describes example image search queries[2]; each has three predicates, resulting in $K = 7$ predicate combinations. For the current discussion we will focus on the queries called "confused" and "jordan". A more in-depth description of experimental setup is provided later on.

Using the crowd, we determined the correctness of the first 100 items from each predicate combination's candidate results, for each query. Figures 6.4(a) and (b) show these correctness re-

[2]these queries represent real information needs to find pictures to use in talk slides

Query name	Free-form text	Predicates
confused	confused *person* using a computer alone	(1) confused, (2) using a computer, (3) alone
jordan	*Michael Jordan* with basketball full body black background	(1) with basketball, (2) full body, (3) black background
pair	two *people* using computers white background	(1) two, (2) using computers, (3) white background
professor	cartoon *professor* with gray hair and mustache	(1) cartoon, (2) has gray hair, (3) has a mustache

Table 6.1: Image search queries each with three predicates used in experimental results. For each predicate combination, search engine queries are constructed by applying the predicates to the italicized entity in the free-form text.

sults for the "confused" and "jordan" queries. Each line corresponds to one predicate combination, showing how many items were correct after n items were processed, for $1 \leq n \leq 100$.

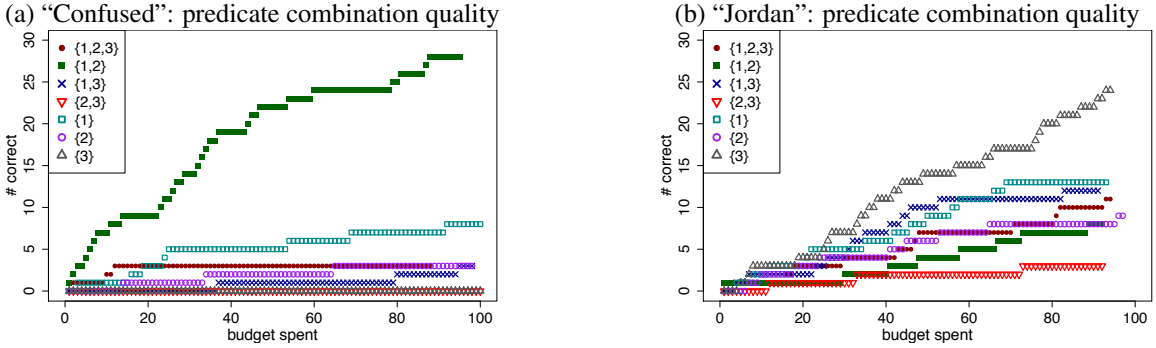


Figure 6.4: Number of correct items found in the first 100 ranked candidate results from each predicate combination, for the (a) "confused" and (b) "jordan" queries. Combinations are denoted in the legend, corresponding to those shown in Table 6.1.

For the "confused" query, there is a single dominant predicate combination, consisting of the two predicates "confused" and "using a computer" (predicate combination $\{1,2\}$). The performance of this combination is best throughout the first one hundred images returned; while its rate of correctness does slow with increasing rank, it remains superior to all other combinations. The second highest correctness arises from the combination made up of the single predicate "using a computer" ($\{1\}$). In contrast to the other high-scoring combination, its rate of success does not appear to be correlated with rank.

Unlike in the "confused" query, the baseline combination that includes all predicates in the "jordan" query performs reasonably well, particularly at lower rank. The best combination at the beginning is the two-predicate combination "basketball" and "full body" ($\{1,2\}$), but its rate of suc-

cess remains relatively low and it is surpassed by nearly all other combinations as rank increases. The clear overall winner is the single predicate "black background" ($\{3\}$) whose correctness count grows beyond the other combinations after its first eight items; it is followed relatively closely by the single predicate "basketball" ($\{2\}$). For this query, the two best combinations also have the highest correctness counts for their own predicate.

The results of the "jordan" query highlight the impact of predicate correlations. For example, the single predicate "black background" was sufficient to yield images that reflect all three predicates. We hypothesize this is because images of Michael Jordan with a black background are more likely to depict him in a stylized manner dunking a basketball. As candidate results sets generated from single predicates tend to perform well for their own predicate, correlations can be advantageous for overall quality in the final query result.

6.3.2 Naïve Approaches

In what follows, we refer to these two example queries as we describe the pitfalls of possible approaches to the predicate-combination search problem.

Fixed pruning approach

Our objective is to judiciously explore the performance of the candidate results sets produced by each predicate combination to produce a final query result of maximum quality for the user. A natural strategy would be to explore a fixed number of candidate results from each combination and then choose the one with the highest recall, or average correctness. This strategy would be akin to looking at the first page of query results from a search engine. The problem with this strategy, which we will call the "fixed pruning" approach, is that it can waste considerable crowdsourcing budget exploring combinations that yield few or no correct items. Such is the case for the "confused" query, which has many combinations with poor quality; with larger K the amount of wasted budget would be even greater. For a given query, we will not know which of the K predicates combinations will have low quality.

Early stopping approach

To counter the above effects, another intuitive strategy would be to leverage the ranked nature of the candidate results sets by disqualifying a predicate combination once it yields its first incorrect item, exploring the set in order of rank. However, consider this aggressive pruning strategy in the context of the "jordan" query in Figure 6.4(b). Its best performing predicate combination for the first 100 search results turns out to be the single predicate "black background". This predicate combination has no correct items at the top of its ranked list, though; thus it would be pruned in this strategy before its full potential is realized.

The key to understanding why the aggressive pruning strategy is problematic is to understand how the candidate results sets are ordered. Recall that each set of items is generated by sending a particular predicate combination as search terms to an automated system. Thus each candidate

results set is a ranked list in which items decrease in relevance *with respect to the predicates in its combination*. For example, if a particular combination consists of two out of three total predicates, we expect its list to be sorted in correctness according to the two predicates, while the correctness of the third predicate in the list can be in random order. While search engines do not produce perfect rankings even for the given search terms, the likelihood of an item being correct for the predicate(s) its ordered by will be greater at the top of the list, which serves as an upper bound on the likelihood of the item being correct for all predicates. In the case of the single predicate "black background" in the "jordan" query, its candidate results set for the first 100 items is mostly correct for its own predicate, and the items that are additionally correct for all predicates appear randomly within the 100 results.

The ranked nature of the data also implies that different predicate combinations can grow worse at different rates as rank increases. In other words, the relative performance of the combinations can shift as their result sets are traversed. This behavior is present in both the example queries shown in Figure 6.4(a-b). An algorithm that does not acknowledge these shifts in performance may give up exploration too soon, and get stuck in a local maximum.

Crowd-voting approach

Another natural approach would be to leverage the crowd to help prune the space of candidate results sets, with the intuition that humans should be able to tell which predicate combination will yield the best results for the query. To determine the best combination, simply ask several crowd workers to decide which they feel is best and take a majority vote. However, as described above and exemplified by the "jordan" query (Figure 6.4(b)), there may not be a single best predicate combination due to the ranked nature of the candidate results sets — leading to disagreement amongst the workers. For example, when we experimented with this approach, the first three responses for the "jordan" query were votes for three different predicate combinations. This approach has potential, however: we did find that the crowd is adept at identifying which combinations are unequivocally low-quality. We incorporate this idea into the algorithm we developed, described next.

6.4 Crowd-Bandit Algorithm

Recall that our goal is to prune the space of candidate results generated by processing K predicate combinations using an automated system; we will send select items to the crowd to determine if they satisfy the full query. We can think of the problem as an iterative process, where for each budget unit we spend we ask the following question: from which candidate results set should we take the next item? As motivated in the previous section, for a given candidate results set we always take an item in the order of the ranking.

This process is similar to the classic multi-armed bandit (MAB) problem, in which a gambler tries to maximize the reward from a sequence of lever pulls on K slot machines, where each of the K machines has an associated reward distribution. In our case, we have K ranked candidate results sets and we seek to determine how many items should be taken from the top of each list in

order to maximize the number of items that match all predicates; the reward is $\in \{1, 0\}$, indicating if the chosen item was correct or incorrect.

Pitfalls of traditional MAB

The traditional MAB problem aims to minimize its lost rewards while it explores the K slot machines, until it finds the single optimal machine. However, as described in the previous section, the ranked nature of the candidate results sets can cause shifts in relative performance with increasing rank. Thus the objective of seeking the single best "bandit" is not appropriate in our scenario. This fundamental difference also influences how we might leverage the crowd to prune the search space.

Due to the ranked nature of data, we need to modify traditional algorithms for the MAB problem. We present next a hybrid crowd-bandit approach to determine which candidate results should be evaluated to see if they satisfy the full query. In particular, our approach is based on the greedy class of MAB algorithms and incorporates the crowd to help pre-filter which "bandits" will be considered.

Pre-filtering bandits using the crowd

As noted earlier, there is not one best predicate combination because candidate results sets are (mostly) ranked. Thus asking the crowd to decide which is the best is the wrong approach. However, while humans may not be adept at determining the optimal subset of the various candidate results, they can figure out which predicate combinations yield poor quality and should be removed from consideration. In other words, rather than the MAB algorithm considering $K = \mathcal{P}(P)$ candidate results sets, instead it will only consider some $K' \leq K$.

Our approach asks crowd workers to choose which predicate combination they believe will yield the best set of results given a textual description of the information need. While several predicate combinations may receive votes, some may receive no votes at all. These are the combinations that will be removed from consideration by the adapted MAB algorithm. We describe the details of this crowdsourcing task in the experimental setup later on.

Bandit algorithm core

Given a set of bandits, a set of candidate results sets in our case, a MAB algorithm iteratively decides from which bandit it will take an item from. In our scenario, at each step the algorithm is given as input the correctness ratio thus far for each predicate combination, i.e., the number of correct items drawn divided by the total items drawn from that combination. An ϵ-greedy algorithm would choose the combination with the highest average correctness, or with $epsilon\ \epsilon$ probability choose a combination at random.

The problem with this traditional approach is that it does not adapt well to shifts in the combinations' relative performance, as described in the previous subsection. Namely, a combination that is performing poorly in recent history may continue to be chosen because it performed well in the

past. It may take a substantial number of items being evaluated as incorrect before the combination's correctness ratio drops low enough for a competing combination to surpass it. To ameliorate this issue, we alter the greedy algorithm to use a sliding window of performance history, i.e., only consider the last w drawn results when deciding which combination is currently best. Furthermore, in order to encourage exploration of a combination that has not been correct recently, the algorithm continues drawing from a combination's candidate results while it continues yielding correct items.

Enhancement: leveraging overlaps

Candidate results sets may have some items in common; we call such items *overlaps*. It may seem intuitive to evaluate all overlapping items first, as this would allow the algorithm to learn about the performance of more predicate combinations at the lowest crowdsourcing cost. This strategy, however, ignores the rank information, namely, that low-ranked items have a lower likelihood of being correct, even if they are overlaps. Furthermore, it may seem that items that appear in multiple results sets would be more likely to be correct. As we observed in our experimental results, this property does not hold. For example, Figure 6.5 shows for the "jordan" query the number of total overlapping items each predicate combination participated in; the dark bars represent how many of these overlaps were actually correct. Thus our algorithm only pre-processes overlaps that appear in the top $overlap\_threshold$ of each candidate results set it appears.

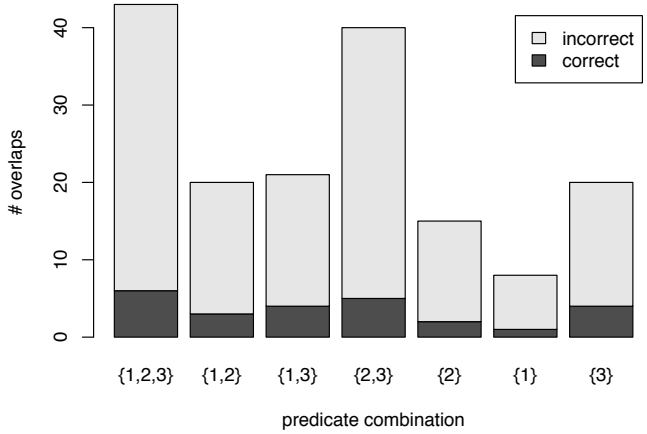


Figure 6.5: Overlapping items in the "jordan" query

Other MAB algorithms considered

The UCB algorithm [88] picks the combination with the highest score, defined by the average correctness ratio plus a weighting metric that influences how exploration is done. The weight incorporates how many times a combination has been chosen relative to the total number of items drawn so far, encouraging exploration of under-utilized combinations. This aspect of the score creates a periodic round-robin effect, which facilitates adaptivity because the algorithm explores

throughout (although decreasing in frequency over time). While we saw better performance with the "tuned" version that also includes reward variance, we found that the UCB algorithm did not perform as well as the approach based on the ϵ-greedy algorithm.

Algorithm pseudocode

Pseudocode for the core of our greedy crownd=bandit algorithm appears in Algorithm 2. We maintain a list of eligible combinations (i.e., those combinations that have not yet been pruned); the list of eligible combinations may be pre-filtered by the crowd, as we discuss above. We also keep a sliding window of correctness statistics, which can be probed for information about each combination. Before the main iterative loop begins, the algorithm first processes all items that appear in multiple candidate results sets, and are ranked $overlap\_threshold$ or higher in each (lines 3-9). The algorithm then also draws once from each combination (not shown). Next it iteratively decides which combination to draw from, while there is budget remaining, using the current window of correctness statistics. If the combination chosen in the previous iteration yielded a correct item, it is chosen again (lines 11-12). Otherwise, the algorithm selects the combination with the current highest correctness rate or, with ϵ probability, a random eligible combination (lines 13-14). Finally, it processes an item from the chosen combination, updates the statistics window and slides the window forward (lines 18-20).

6.5 Experimental Results: Image Queries

In this section, we investigate the effectiveness of the crowd-bandit algorithm in pruning the space of candidate results to generate a set of image search results for the end user. We continue with the image search examples described earlier in Table 6.1. After detailing our experimental setup, we show the algorithm's performance on queries, comparing it to several naïve approaches introduced earlier, as well as discuss how various characteristics of the queries impact algorithm behavior.

6.5.1 Experimental Setup

We use Google Image search as the automated system. For each predicate combination we query the Google search API[3] to find and extract the first 100 image search results. We use Amazon's Mechanical Turk (AMT) crowdsourcing marketplace for paid crowd processing to evaluate each image's adherence to the three predicates. In the tasks we post, workers are presented with five images and asked to evaluate/verify the predicates by indicating "yes", "no", or "unsure" for whether each predicate applies to each image. They are paid $0.02 for each set of five images. Ten workers are asked to evaluate each picture, and we count an image as satisfying a predicate if the majority of those ten workers select "yes". For each query, we used the crowd to process all images for each predicate combination in order to evaluate different algorithms.

[3] http://developers.google.com/custom-search

Algorithm 2 Bandit algorithm core

$eligible \leftarrow$ list$(1, 2, \ldots, K)$
$window \leftarrow$ list()
$overlaps \leftarrow$ items in at least two of $S_1 \ldots S_K$ and $\forall j$ rank$(item_j) \leq overlap_threshold$

for all $item \in overlaps$ **do**
  $correctness \leftarrow$ evaluateCorrectness($item$)
  for all i where $item \in S_i$ **do**
    $window$.add(i,$correctness$)
  end for
end for

repeat
  if previousCorrectness $== 1$ **then**
    choice $\leftarrow$ previousChoice
  else if generateRandom(0,1) $< \epsilon$ **then**
    choice $\leftarrow$ selectRandom($eligible$)
  else
    choice $\leftarrow \arg\max_{i \in eligible} \frac{\sum window.get(i)}{window.get(i).size()}$
  end if

  $correctness \leftarrow$ drawNextAndEvaluate(choice)
  $window$.add(choice,$correctness$)
  $window$.removeFirstEntry()
until until budget exhausted

We use AMT for the crowdsourced pre-filtering of predicate combinations as well, paying \$0.05 per task. In each task, the worker is first presented with a description of the images desired (the query entity and a bulleted list of the predicates). They are then shown each of the $K = 2^{|P|}$ search engine queries corresponding to the predicate combinations, and are asked to select which query they think yields the best set of results using Google image search. We remove from the set of combinations the bandit algorithm considers those predicate combinations that received zero votes after K workers have made their selection.

6.5.2 Algorithms Compared

In the results that follow, we compare the crowd-bandit algorithm to three alternate algorithms.

- The first is the **all-predicate** algorithm, which corresponds to an algorithm that does no exploration of the predicate search space. Instead, it draws all its items (in order) from

the candidate results set that is generated from the combination that contains all the query predicates.

- The second alternate algorithm is the **fixed pruning** strategy mentioned in the previous section. This strategy explores every predicate combination for a fixed amount n, drawing items in a round-robin fashion, then spends the remainder of its budget drawing from the predicate combination that has the highest average correctness at that point. For our experiments, we set $n = 10$, akin to looking at the first page of search results.
- Finally, to illustrate the benefit of the crowd-based pre-filtering component, we include a version of the bandit algorithm, **bandit-only** that does not have any predicate combinations pre-filtered by the crowd.

For both bandit-based algorithms, we set $\epsilon = 0.1$, and each of the windowing and overlap threshold parameters to 10.

6.5.3 Summary of Crowd Pre-Filtering Outcomes

The crowd-bandit algorithm uses the crowd to first filter which predicate combinations, or bandits, that the bandit component of the algorithm will consider. Table 6.2 shows for each query which predicate combinations received a nonzero number of votes from the crowd, when asked to decide which combination yields the best set of search results. The emboldened combination represents the combination that received the most votes. In all cases several of the combinations are removed, reducing the exploration that the MAB-based part of the algorithm will need to perform. We discuss this more next.

Query name	Crowd-voted combinations
Confused	{1,2,3} and {**1,2**}
Jordan	{1,2,3}, {1,3}, and {**3**}
Pair	{**1,2,3**} and {1,2}
Professor	{**1,2,3**} and {1}

Table 6.2: Predicate combinations that received votes for each query during the pre-filtering stage.

6.5.4 Observations from Performance Results

Figure 6.6 depicts the number of correct items found for increasing budget by each algorithm for each query; results are averages over 100 runs of the algorithms. Each increment along the x-axis represents an item that was evaluated by the crowd to determine whether it matched all predicates.

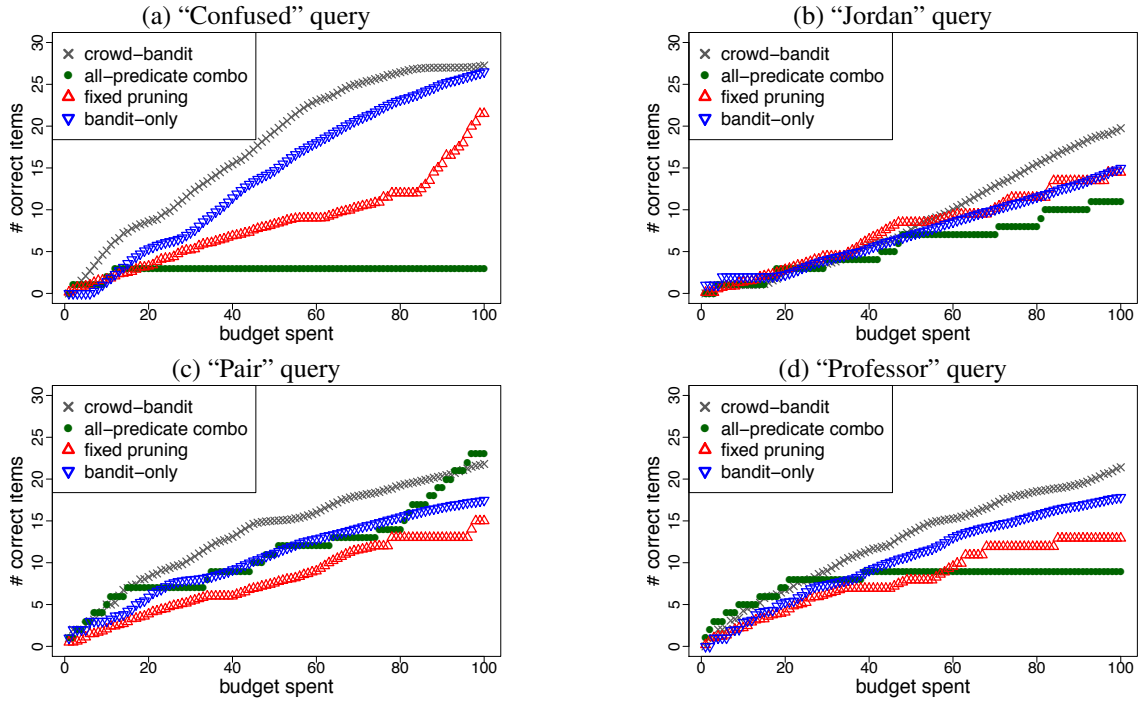


Figure 6.6: Number of correct items found for increasing crowdsourcing budget for each algorithm, for the four image search queries described in Table 6.1.

Algorithm performance impacted by relative quality of the predicate combinations

For some queries, there is one predicate combination whose candidate results set is strictly better than the others when traversed by rank, like "confused". This behavior is well-suited for a greedy bandit algorithm, since it continues to draw from the combination that it finds best at the beginning, unless another candidate results set can overpower it. As the window of performance slides in the greedy algorithms, the dominant predicate combination continues to re-prove itself as the best. Crowd pre-filtering eliminates from consideration all combinations but {"confused", "using a computer"} and {"confused", "using a computer", "alone"} for the "confused" query, allowing the crowd-bandit algorithm to avoid randomly exploring the other combinations and likely getting incorrect items. This behavior gives it a boost in performance over bandit-only.

The difference in quality between the predicate combinations deeply impacts algorithms that attempt more explicit, upfront, exploration of all combinations, namely fixed pruning. When more of the combinations have similar quality, as in "jordan", "pair", and "professor", the budget that this algorithm spends exploring all options is not severely wasted. However, if more of the combinations have low correctness rates, it is hard for fixed pruning to make up that loss when it does cease its exploration. For example, fixed pruning performs better on the "jordan" query than "confused" because the differences in performance between the predicate combinations are less stark in "jordan"; thus for "confused" the algorithm wastes a lot of budget during its exploration. Recovery from over-exploration of low-quality combinations is possible over time if the chosen combination

on which to spend the remaining budget has a high correctness rate, as seen in Figure 6.6(a) for "confused".

Windowing helps adjust to relative quality shifts between combinations

The windowing technique proves advantageous for queries with combinations that shifts in relative performance. In "professor", the combination that is the best at the beginning hits a permanent dry spell. The greedy algorithm is able to notice this change more rapidly because of the sliding window of correctness history. A similar scenario occurs with the "jordan" query for the bandit-only algorithm. Initially the two-predicate combination $\{1,2\}$, corresponding to "basketball" and "full body", performs best, however it is quickly surpassed by all of the other combinations; the windowing helps the algorithm avoid getting stuck on that combination. With crowd pre-filtering in the crowd-bandit algorithm, this combination is eliminated all-together.

All-predicate algorithm is unpredictable

There can be cases when the all-predicate algorithm actually performs relatively well, as is the case with the "pair" query; the combination using all predicates has the second-best performance, and closely trails the best combination. This will happen when the search engine handles the predicates well, but this scenario cannot be known at query run time. The unpredictability of the all-predicate algorithm is exemplified by comparing the "confused" and "jordan" queries — it performs abysmally for the former, yet reasonably for the latter (Recall Figures 6.4(a) and (b)). For the "pair" query, the bandit-only approach is still able to perform well, despite expending resources on exploration. Again, the crowd-bandit algorithm filters out poor-quality predicate combinations, and finds a balance between the best performing combination and the all-predicate combination.

6.5.5 Additional Discussion

In general, the greedy MAB-based algorithms effectively tackle the challenge of adapting to the characteristics of the predicate combinations' candidate results sets, taking the "tops" of the better-performing results and avoiding the pessimal ones. The addition of the crowd-based pre-filtering of predicate combinations that the bandit component of the algorithm considers further reduces exploration into low-quality candidate results. For a given query we will not know a priori which predicate combination(s) will yield quality results, highlighting the importance of an adaptive algorithm. The all-predicate and fixed pruning algorithms suffer because of their rigid nature. The crowd-bandit algorithm is robust to a variety of scenarios that result in real queries, succeeding when the alternative strategies fail.

6.6 Cost Prediction: Pay-As-You-Go

In previous sections, the objective was to iteratively explore the candidate results, deciding which items to send to the crowd, given a maximum crowdsourcing budget. However, the cost of acquir-

ing more correct items may increase over time as the exploration algorithm traverses the candidate results sets. Having an estimate of this cost would allow the user to reason about the cost versus benefit tradeoff of getting another correct item. In this section we devise two algorithms to predict the cost for retrieving the next correct item: a simple difference quotient method and a stochastic technique that considers the techniques used in the crowd-bandit algorithm. We compare the algorithms using the "confused" and "jordan" queries.

6.6.1 Simple Difference Quotient Estimator

By considering the total cost of an item as a discrete function f over the number of items, we can estimate the cost of the next item by using the first derivative of f (the difference quotient). Thus after correct item t, an estimator for the cost $\hat{C}_{t+1}$ of correct item $t+1$ is:

$$\hat{C}_{t+1} = \frac{\Delta f(t)}{\Delta t} = \frac{f(t) - f(t-h)}{h} \tag{6.1}$$

A variant of this estimator sets $h = 1$, simplifying the estimation to predict the cost of the next item as the same as the observed cost for the previous item:

$$\hat{C}_{t+1} = C_t \tag{6.2}$$

Using the difference quotient is a simple technique, but it is sensitive to cost fluctuations, in particular when $h = 1$. For instance, if by chance there are two correct items one soon after the other, the algorithm will predict that the cost for the next item is also low. The approach also does not consider the details of our crowd-bandit algorithm, which will impact the quality of the estimator.

6.6.2 Greedy-Windowing Cost Estimation

To overcome the shortcomings of the simple difference quotient method, we develop a new algorithm that considers the stochastic characteristics of our windowing exploration algorithm. We use the concept of *regret* from the MAB problem to make a cost estimation.

Recall that with a set of predicates P we have the predicate combinations $K = \mathcal{P}(P)$. Let $k_1, \ldots, k_{|K|}$ be all predicate combinations and $\hat{p}_i$ be the observed success rate for a predicate combination i, defined as

$$\hat{p}_i = \frac{\#\text{valid items with combination } i}{\#\text{items retrieved from combination } i} \tag{6.3}$$

Let us further define k^* as the combination with the highest observed success rate $\hat{p}^* = max_{i=1}^{m} p_i$ at a particular point in time. The regret at a particular step for the MAB problem is defined as the expected difference between the reward sum associated with an optimal strategy and the sum of the actual collected rewards. While we cannot know the optimal strategy, we can estimate the regret R at step n based on the items observed so far using the success estimates $\hat{p}$ [88]:

$$\hat{R}_n = \hat{p}^* n - \hat{p}_j \sum_{j=1}^{|K|} \mathbb{E}\Big(T_j(n)\Big) \tag{6.4}$$

where $T_i(n)$ is a random variable denoting the number of items received from combination i during the first n items.

Unfortunately, the regret equation does not provide information about regret on a per-turn basis, which our estimator will need to predict the cost of finding the next correct item. To this end, we develop a concept of regret per turn that incorporates correctness likelihood for the next item. Recall that the greedy aspect of the algorithm causes it to chooses the combination k^* with highest success rate, or a combination at random with ϵ probability. In our scenario, the reward is $\in \{0, 1\}$ and thus the next item being correct is a bernoulli trial; this allows us to add per-turn regret directly[4] to $\hat{p}^*$ to achieve the adjusted probability $\hat{p}_r$:

$$\hat{p}_r = (1 - \epsilon)\hat{p}^* + \epsilon \frac{\sum_{\forall i: k_i \neq k^*} p_i}{m - 1} \tag{6.5}$$

The cost for the next item is then calculated as the expected number of turns required to receive the next item, where $Y(n)$ is a random variable denoting the number of trials to receive n successful items:

$$C_{t+1} = \mathbb{E}\Big(Y(1)\Big) = \sum_{i=1}^{\infty} \Big((1 - \hat{p}_r)^{i-1} \hat{p}_r i\Big) \tag{6.6}$$

This estimator does not take the windowing semantics of the algorithm into consideration, which is problematic because it ignores the impact of windows on the statistics. For instance, if the windows are chosen to be very small, there may be no statistics for the success rate $\hat{p}_i$ if, during the last window-size number of items, no correct item was found. We propose a simple adjustment to overcome this problem: we correct the likelihood $\hat{p}_r$ by taking the maximum of $\hat{p}_r$ and the overall success rate observed so far $\hat{p}_a$:

$$\hat{p}'_r = max(\hat{p}_r, \hat{p}_a) \tag{6.7}$$

Using the maximum ensures that if no statistics exist for the $\hat{p}$ and the algorithm has randomly chosen one of the combinations (line 14 in Algorithm 2), then the estimation will consider the overall success rate to calculate a reasonable estimate.

6.6.3 Cost Prediction: Experimental Results

The goal of the pay-as-you-go (PAYG) prediction is to estimate how much it will cost to yield another correct item, information the user issuing the query can use to decide to end the process. We now show results for the PAYG cost predictions described above, applied to our algorithm.

[4] i.e., we do not need to convolute the distribution for the different predicate combinations and can assume $\hat{p}_i$ as the probability of success for each combination i

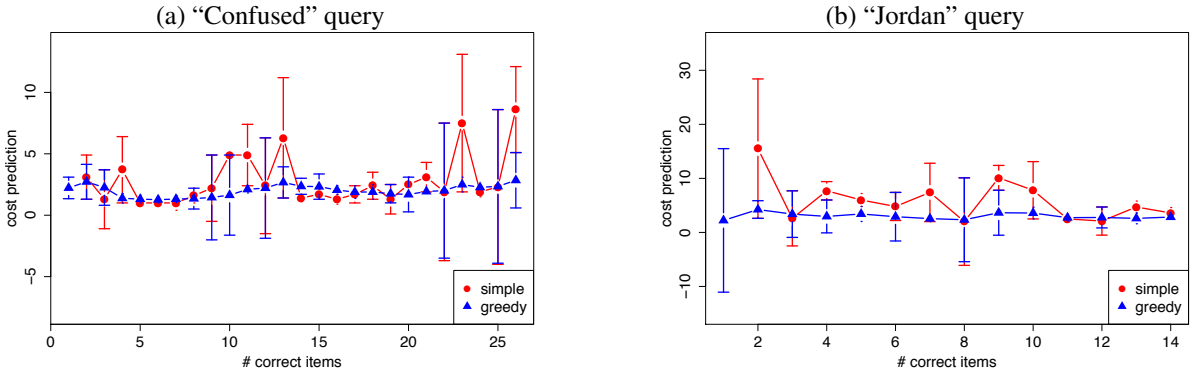


Figure 6.7: Comparison of the simple and greedy approaches for pay-as-you-go cost predictions, for the “confused” and “jordan” queries. Error bars represent the absolute value error between the predicted cost and actual cost to get the next correct item.

We compare the simple different quotient and greedy PAYG approaches, and show that the greedy strategy is able to achieve lower prediction error. We present results with the two image search queries featured throughout this chapter, “confused” and “jordan”.

The PAYG strategies estimate the cost of the next correct item, i.e., how many more items will need to be evaluated before another is found, after each correct item found so far. We calculate the prediction error for both strategies by taking the difference between the predictions and the actual cost observed when running the greedy windowing algorithm. Figure 6.7 shows the cost predictions for both queries; error bars indicate the absolute value of the difference between the predicted cost and actual cost. The results represent averages over ten runs of the greedy windowing algorithm for each query.

As expected, the greedy strategy outperforms the simple strategy. The greedy strategy produces more stable predictions and has lower total average error. For the “confused” query, it has total error of 38.2 compared to 49.4 for the greedy and simple strategies, and 45.5 versus 49.7 for greedy and simple, respectively, for the “jordan” query. This improvement in performance for the greedy strategy is due to its use of more information about the success probability of the predicate combination that the exploration algorithm is likely to to choose. Periods of larger error tend to correspond to unexpected plateaus in accumulation of correct items (e.g., Figure 6.4(a)); note that the ranked nature of the data renders these shifts inherently difficult to predict. However, despite this challenge the PAYG approaches yield reasonable predictions, demonstrating that looking ahead to the near future is effective.

6.7 Set-Oriented Candidate Results

In the ranked variant of the predicate combination search problem, the likelihood of a particular candidate result item being correct is correlated with its position in the list. Due to this characteristic, the relative performance of the various predicate combinations can shift with increasing

rank. The optimal answer set may include the highest-ranked items from multiple candidate results sets. In contrast, in the set-oriented case, any candidate result item is equally likely to be correct. In this simpler scenario, the optimal solution is straightforward: all candidate results should be taken from the single predicate combination with the highest correctness likelihood. A successful algorithm for the set-oriented case must quickly learn which predicate combination is the best in order to spend the majority of its budget on that set of candidate results.

In this section, we show that the crowd-bandit algorithm we developed to tackle the ranked problem variant is also effective for set-oriented candidate results sets. Using the candidate results sets and their crowd-based correctness evaluations from the image search experiments used earlier on, we create simulated data for this set of experiments. For each algorithm, candidate results are processed in random order, rather than in rank-order; i.e., at each step, an algorithm selects a predicate combination to draw from and then samples without replacement from that combination's set of candidate results. As before, results shown are averages over 100 iterations of the simulation. Figure 6.8 shows the performance of the four algorithms: all-predicate combination, fixed pruning, bandit-only, and crowd-bandit.

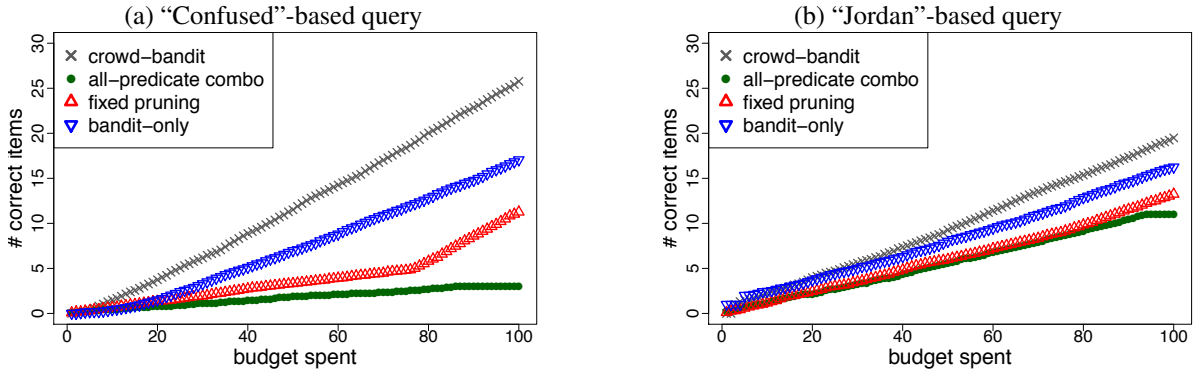


Figure 6.8: Algorithm comparison for set-oriented case with simulated data based on image search queries.

As expected, the two bandit-based algorithms are able to achieve a higher number of correct items at each budget point than the other algorithms. This makes sense, as the set-oriented case more approaches the natural scenario that MAB algorithms were originally designed to address. The crowd-based pre-filtering of low-quality predicate combinations gives the crowd-bandit the extra boost of performance over bandit-only.

The magnitude of the gains in performance for the bandit algorithms is, again, influenced by the difference in quality between the predicate combinations. For example, recall from Figure 6.4 that the difference between the optimal and suboptimal combinations in "Confused" is much greater than in the "Jordan" query. This stark difference between the combinations in the "confused"-based query dramatically impacts the fixed pruning algorithm: its performance remains low until it discovers the best combination and devotes the remainder of its budget to that, causing an obvious shift in the graph. With smaller differences between the combinations, as in the "Jordan"-based query, fixed pruning is not as paralyzed.

6.8 Related Work

Crowdsourcing has been used for information retrieval (IR) [89]. Most related to our approach is the work on using crowdsourcing for relevance evaluation [90, 91]. Currently, our techniques only consider an item to be relevant or not. Incorporating more advanced techniques, including ranking results or correcting inter-worker disagreement [92], is an interesting area of future work. Other proposed hybrid IR techniques aim to improve query results by increasing the overall answer quality for many queries, e.g., by labeling data [93, 89], which is not focus of this chapter, or by crowdsourcing query structure in order to query structured data sources [94]. Using the crowd to rewrite queries with different search terms [87] would be a complementary, pre-processing step to our work.

Submitting different forms of a query to a search engine is similar to the process of querying multiple systems in parallel, as in federated search [95] and meta search engines [96]. These systems post a user query to several search engines or databases in parallel and combine the results into one search set of results. Work in metasearch models [97] investigates models to achieve the optimal combination of results with upper bounds on performance. However, these techniques fall short if combined with the crowd, as they do not consider the quality feedback nor the cost of crowdsourcing. Taking samples of search results can also be used for search suggestions and query reformulations [98].

As mentioned in Chapter 3, Hristidis et al. [99] looks at top-k search in a database that produces unranked search results. Searching the database incurs a cost, so their objective is to design a sequence of conjunctive queries to yield the top-k at low cost. They assume in their scenario that search results correctly match search terms, which is in contrast to our problem setup, and why we leverage the crowd.

We adapt the ϵ-greedy algorithm [100] from the MAB literature. There is a lot of theoretical work on MAB, typically on bounds on asymptotic behavior and regret [101, 88]; recent work has been applied to web search to produce better-ranked search results [102].

6.9 Conclusion

In this chapter we address the problem of determining which parts of a query are difficult for a search engine to process, and are better left for human computation. We use a two-stage process to address this issue. Given a query with a set of predicates P, we create $2^{|P|}$ query reformulations and issue each of these to the automated system, generating $2^{|P|}$ sets of candidate results, i.e., sets of items that could potentially satisfy the original query. The crowd then processes a chosen subset of these candidate results for correctness.

We define the choice of which subset of candidate results to send to the crowd as the *predicate combination search problem*, and provide a formal definition. The optimal solution depends on the nature of candidate results sets. If the sets are unordered, or set-oriented, there is a single best predicate combination and the optimal solution chooses items from only that combination's candidate results. For ordered, or ranked, candidate results sets, the optimal solution may actually be

the top items from several of the predicate combinations' candidate results. However, in both cases achieving optimal would require a costly brute-force strategy—necessitating a heuristic approach.

We devised the *crowd-bandit* algorithm, which chooses the best subset of candidate results from amongst all the candidate sets to send to the crowd to evaluate. It leverages key ideas from the multi-armed bandit problem: exploit the sets of candidate results that yield more correct items, while also exploring other sets to learn about their performance. Due to the ranked nature of the candidate results sets produced by many search engines, we had to augment the traditional algorithm to adapt more quickly to changes in performance between the sets of candidate results. We also selectively use the crowd to help pre-filter the sets that are considered by the bandit-based algorithm.

In our experimental result with image search queries, we compare the crowd-bandit algorithm to several alternative strategies: a baseline strategy that does not attempt query reformulation, as well as strategies that naïvely explore the space of candidate results. For a given crowdsourcing budget, the crowd-bandit algorithm yields more correct items than each of these other approaches. The crowd-bandit algorithm additionally outperforms the other strategies for the simpler, set-oriented version of the problem.

Chapter 7

Future work

In this thesis, we developed statistical techniques for reasoning about query result quality, and the tradeoff between quality and cost, for hybrid human/machine database systems. Thus far we focused on crowdsourced data collection queries, i.e., using the crowd to enumerate sets of items. In this chapter we discuss several interesting directions for future work, building on the work from this thesis. First we describe a query optimizer for human/machine database systems, the component of the system that decides on an execution strategy for a given query. We then briefly discuss future work in managing crowdsourced data, as well as the potential of expert crowd routing.

7.1 Query Optimizer for Human/Machine Database Systems

An intriguing direction of future work is the development of an optimizer for hybrid human/machine database systems. As described briefly in Chapter 2, in a traditional database system, the query optimizer is responsible for choosing an execution strategy that minimizes the cost of processing the query; cost is typically calculated as the amount of resources, like disk input/output (IO), that are needed to execute the query plan. Thus, a common approach used in traditional RDBMSs is *cost-based optimization*: various query plans are compared, and one is chosen that minimizes cost. The optimizer also uses heuristic rules to adjust the plan that can help reduce cost[1].

In hybrid systems, the definition of cost needs to be extended to incorporate the costs associated with using human computation within query plan operators. In particular, in these systems the optimizer must consider a crowdsourcing budget when choosing a execution plan. Techniques that determine how to reduce the amount of human computation used can help reduce the cost of the overall query plan. This section outlines plans for several optimization techniques for use in hybrid database system that aim to lower the cost of human computation in crowd-powered operators, with a focus on data collection tasks.

[1]Recall the rewrite rule in Chapter 3 used to transform expression 3.1 to expression 3.2: a filter predicate on one relation is executed before taking the cartesian product of two relations, reducing the number of tuples that will need to be processed by that join operator.

7.1.1 Dynamic Query Processing

When determining an execution strategy, a query optimizer chooses from amongst several physical implementations for each logical operator that appears in the query plan. These choices are typically fixed at the start of query execution. However, work in adaptive query processing (see [103] for a survey) looks at shifting the execution strategy while the query is running in response to changes in the environment. For example, Avner et al. [104] develop a technique to dynamically re-order joins in a distributed query processing setting if the latency of scanning a data source for a given relation grows due to network issues.

There is an opportunity to employ dynamism in query plans in hybrid systems as well, both to switch between physical operators and within a single operator. These shifts can be advantageous for reducing crowdsourcing cost in enumeration queries. One such shift was discussed in Chapter 5 to react to "listwalking" behavior: for some set enumeration queries the result might already exist on a single webpage, thus switching from the simple or Negative Suggest task interfaces to one in which a worker scrapes the data is likely to be more cost effective. Deciding the appropriate threshold to trigger this switch is an open problem for research, as is the development of a hybrid approach that uses both task interfaces in tandem.

Another potential dynamic strategy for enumeration queries would be to adapt the task interface to target less well-known items—the tail of the item distribution. This type of technique could complete the set more quickly and thus at lower cost. The potential of this approach was demonstrated in Chapter 5 with the use of "partition hints"; asking workers to provide an answer that fell in a specific partition of the set space helped diversify the range of answers received. However, both the set of partitions and the ratio with which they were given as hints were fixed at query runtime. A dynamic approach could observe and then cluster the stream of answers from the crowd, and use this information to adaptively create partitions to suggest as hints.

7.1.2 Prefetching

When workers gather data to answer a given query, it can be useful if they gather additional data not relevant to the query but may be useful for future queries. This "prefetching" can save time and money if it is as easy for the worker to provide multiple pieces of information. For example, if a task asks for a professor's phone number given a name, it could also ask for the professor's office location at the same time because these two facts are usually co-located; the additional effort required from the worker is minimal.

A different notion of prefetching would be to gather items of a set that can be used to calculate the query result, rather than only acquiring the query result itself. Consider the following aggregation query that asks for the professor who has been teaching the longest (assuming the relation is initially empty):

```
SELECT name, MIN(year_started) FROM Professors;
```

While workers could provide the most senior professor directly, evaluating the query in this man-

ner requires workers to first find each professor and then inspect his/her profile. It would take a similar amount of effort to first capture all the names and start years and then determine which professor has been teaching the longest. In the latter approach, we can evaluate the query but have the added benefit that the database would be more complete.

On the other hand, a query may specify a predicate that aligns with how the data is naturally partitioned on the web, such as alphabetical or numerical "indexes". In other words, there are times we can take advantage of an existing index over the data we want to access (or in the case of the `MIN` query, a materialized view). A lack of index necessitates a complete table scan to find tuples satisfying the query. The challenge is determining which regime a particular query falls into.

7.1.3 Predicate Ordering

Typically predicates are evaluated as early on as possible in a query plan in order to limit the amount of data that has to be processed in the remaining plan operators. This optimization technique assumes that evaluating the predicate is low-cost, however this is not necessarily true of computationally expensive predicates (e.g., those defined as a UDF). Some predicates may cost more than others; in hybrid systems, this cost may entail worker think time, number of votes to reach consensus, etc., in addition to monetary cost. The optimizer in hybrid systems also may need to learn the costs associated with particular predicates on the fly, to detect, for example, which predicates may be more ambiguous and/or subjective and thus potentially costly to reach agreement upon. Choosing an evaluation order that "fails fast", i.e., cheaply reduces the number of tuples that must be processed upstream, will minimize query processing cost. This is the idea behind *expensive predicates* [105].

Work on expensive predicates describes augmenting the traditionally optimized query plan to evaluate predicates in order by their *rank*, defined as (selectivity-1)/cost-per-tuple. In addition to the challenges associated with reasoning about cost, selectivity information may be unknown for a crowdsourced relation, and so the question of whether the crowd itself can be leveraged to estimate predicate selectivity arises.

7.1.4 Joins with Crowdsourced Relations

This thesis focused on crowdsourced enumeration for tuples in a single relation. However, joins between two crowdsourced tables are possible. In many traditional join algorithms, one table is the outer relation, and one is the inner relation; the outer probes the inner for matching tuples. CrowdDB [11] discusses joins between a `CROWD` table and a non-crowd table, using the latter as the outer relation. In general, the choice between outer and inner is informed by resource costs when executing the query.

For joins between multiple `CROWD` tables, one challenge is determining the amount of enumeration necessary for each table to respond to the query. One way to address this issue is to leverage *foreign key constraints*, a type of schema information, which detail the constraints between items across relations. For example, a constraint in a database describing course enrollment information might have a constraint that a student cannot enroll in a course if that course does not exist.

As an example, consider a query that asks for the names of actors who have acted in a movie that has won a "best movie" award using the following simple schema (assume each is a crowdsourced table):

```
Movies(name); Acted_in(movie, actor); Best_movies(movie,year)
```

There are two interesting properties to note about this query. First, there is a zero-or-one entity relationship between the `movies` and `best_movies` relations, thus the latter is strictly smaller in cardinality than the former. The second observation is the set of "best movies" can be enumerated without needing to populate the set of all movies, since this information is available separately on the web. These two points indicate that the most cost-effective query plan uses `best_movie` to probe the `acted_in` relation, rather than attempting to populate the full `movies` table.

7.2 Managing Crowdsourced Data

Hybrid human/machine database systems leverage human computation when needed, crowdsourcing data in response to a query and then storing the query results. These stored results can be used to answer future queries, possibly augmented with additional input from the crowd. An open area of research is how to manage the storage of crowdsourced data; for example, should it be periodically expunged or revalidated? Recall one of our example queries that asks for the names of restaurants that have scallops on their menu. Menus change over time, thus the results of this query should change as well. It may be necessary to reconfirm correctness or remove items from the stored query result. Having access to the origin of crowdsourced data may be helpful for managing crowdsourced data; this provence information must also be managed.

7.3 Expert Crowd Routing

Part of developing a query optimizer for hybrid database systems involves continuing to devise techniques for understanding and modeling human behaviors and quality. We can use these techniques towards the development of *expert crowd* query processing systems. In this thesis, we focused on homogenous crowds, i.e., we assume that each worker is equally capable but does not possess in-depth domain knowledge. However, for a given question, there may be a particular person or group of people whose expertise could be leveraged. In various communities, e.g., a corporation, medical field, law firm, academic area, etc., certain individuals will be better suited than others to answer certain questions. The expert crowd routing system will determine, given a question or information need, who is the appropriate person or people to ask. The challenges include translating a given information need to the set of skills required to address it, which may require decomposing the question into parts aimed for different experts.

Chapter 8

Conclusion

By combining human computation and machine computation, we can develop new software systems that tap into human knowledge, perception, and experience—allowing these systems to tackle problems that machines alone cannot. We began this thesis with an overview of the types of human computation and crowdsourcing, highlighting example applications as well as describing challenges regarding quality control when leveraging the crowd. In this discussion, we focused on the slice of the human computation taxonomy most relevant to the nature of crowdsourcing used in this thesis. In particular, hybrid human/machine systems take advantage of explicit crowdsourcing through APIs, provided by platforms like Amazon's Mechanical Turk (AMT), to integrate human computation into software systems. Relational database systems are especially amenable to the addition of crowd-powered algorithms due to the inherent level of abstraction that is built in.

Hybrid human/machine database systems form the context of the work done in this thesis. We investigated techniques for data collection, i.e., enumerating a set of items, with the help of the crowd. The first challenge we addressed was how to reason about the results of these crowd-powered enumeration queries. Without the closed-world assumption, the meaning of query results in a hybrid system is unclear. We showed that by adapting algorithms from the literature on species estimation, we can calculate an estimate of query progress; this estimate provides the means to reason the quality of an enumeration query result by estimating the completeness of the result. These same techniques can be applied to reason about the cost versus quality tradeoff: for an increased crowdsourcing budget, how much more can the query result be improved? Incorporating an understanding of the process with which the crowd supplies responses allowed us to ameliorate the impact of human behavior on the estimation algorithm.

When crowd workers provide answers one at a time and without knowledge of one another's responses, we obtain a sample of answers that we can use to calculate an estimate of the total query result set size; the statistical information we leverage is the frequency with which distinct items appear in the sample. However, having many workers submit the same answer is costly. In Chapter 5, we investigated the "Negative Suggest" task interface: workers are shown which responses have already been given and thus they cannot submit. We showed that by allowing a small amount of redundancy, e.g., permitting the same answer to be given up to three times, we can retain the frequency information needed for estimation while reducing overall cost of the query.

Another opportunity to save on crowdsourcing cost is recognizing when the result of an enumeration query is available on a single web page; if so, the results be gathered efficiently with a data scraping tool. To this end, we developed a technique based on the binomial distribution that determines if workers are providing their answers from the same list. If the prevalence of this "list walking" behavior is high, the data gathering strategy can be switched to use the data scraper.

For some enumeration queries, existing search engines can be used to assist with acquiring the query result, reducing the amount of crowdsourcing effort needed. In Chapter 6, we described a two-phase process of combining search engines and the crowd for tackling enumeration queries. First, given a user query, we generate sets of candidate results. Next we use decide which items from all the candidate results to send to the crowd for final evaluation. To inform the decision process, we developed an algorithm based on multi-armed bandit techniques that intelligently draws items from the sets of candidate results by learning which query reformulations provide more correct items. The algorithm adaptively adjusts its strategy as the relative performance of the query reformulations shifts. We showed with image search queries that this approach produces query result sets with more correct items than less adaptive or static approaches.

In summary, this thesis describes statistical tools and techniques for crowdsourced data collection for hybrid human/machine database systems. We have demonstrated that is possible to reason about quality, cost, and the tradeoff between the two, for crowd-powered data gathering tasks. More broadly, we have shown that it is conceivable to incorporate human computation into a query processing system and retain a disciplined understanding of the system's output—increasing the practicality of hybrid human/machine database systems for tackling traditionally difficult queries.

Bibliography

[1] A. J. Quinn and B. B. Bederson, "Human computation: A survey and taxonomy of a growing field," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '11. New York, NY, USA: ACM, 2011, pp. 1403–1412. [Online]. Available: http://doi.acm.org/10.1145/1978942.1979148

[2] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann, "Design lessons from the fastest q&a site in the west," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '11, 2011.

[3] G. Wang, K. Gill, M. Mohanlal, H. Zheng, and B. Y. Zhao, "Wisdom in the social crowd: An analysis of quora," in *Proceedings of the 22nd International Conference on World Wide Web*, ser. WWW '13, 2013.

[4] C. J. Lintott, K. Schawinski, A. Slosar, K. Land, S. Bamford, D. Thomas, M. J. Raddick, R. C. Nichol, A. Szalay, D. Andreescu, P. Murray, and J. Vandenberg, "Galaxy zoo: morphologies derived from visual inspection of galaxies from the sloan digital sky survey," *Monthly Notices of the Royal Astronomical Society*, vol. 389, no. 3, 2008.

[5] J. M. Hellerstein and D. L. Tennenhouse, "Searching for jim gray: A technical overview," *Commun. ACM*, vol. 54, no. 7, July 2011.

[6] S. Cooper, A. Treuille, J. Barbero, A. Leaver-Fay, K. Tuite, F. Khatib, A. C. Snyder, M. Beenen, D. Salesin, D. Baker, and Z. Popović, "The challenge of designing scientific discovery games," in *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, ser. FDG '10, 2010.

[7] L. von Ahn and L. Dabbish, "Labeling Images with a Computer Game," in *Proc. SIGCHI*, 2004.

[8] L. Von Ahn, "Human computation," Ph.D. dissertation, Pittsburgh, PA, USA, 2005.

[9] J. C. R. Licklider, "Man-Computer Symbiosis," *IRE Trans. of Human Factors in Electronics*, vol. 1, 1960.

[10] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, Apr. 2010.

[11] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin, "CrowdDB: Answering Queries with Crowdsourcing," in *Proc. of SIGMOD*, 2011.

[12] A. Parameswaran and N. Polyzotis, "Answering Queries using Humans, Algorithms and Databases," in *Proc. of CIDR*, 2011.

[13] A. Marcus, E. Wu, S. Madden, and R. Miller, "Crowdsourced Databases: Query Processing with People," in *Proc. of CIDR*, 2011.

[14] B. Trushkowsky, T. Kraska, M. J. Franklin, and P. Sarkar, "Crowdsourced enumeration queries," in *ICDE*, 2013.

[15] B. Trushkowsky, T. Kraska, M. J. Franklin, P. Sarkar, and V. Ramachandran, "Crowdsourcing enumeration queries: Estimators and interfaces," *IEEE TKDE*, To appear.

[16] B. Trushkowsky, T. Kraska, and M. J. Franklin, "A framework for adaptive crowd query processing," in *HCOMP (Works in Progress / Demos)*, 2013.

[17] J. Bunge and M. Fitzpatrick, "Estimating the Number of Species: A Review," *Journal of the American Statistical Association*, vol. 88, no. 421, 1993.

[18] N. Lanxon, "How the oxford english dictionary started out like wikipedia," http://www.wired.co.uk/news/archive/2011-01/13/the-oxford-english-wiktionary, Jan. 2011.

[19] D. Grier, "The math tables project of the work projects administration: the reluctant start of the computing era," *Annals of the History of Computing, IEEE*, vol. 20, no. 3, Jul 1998.

[20] A. Doan, M. J. Franklin, D. Kossmann, and T. Kraska, "Crowdsourcing applications and platforms: A data management perspective," *PVLDB*, vol. 4, no. 12, pp. 1508–1509, 2011.

[21] A. Doan, R. Ramakrishnan, and A. Y. Halevy, "Crowdsourcing systems on the world-wide web," *Communications of the ACM*, vol. 54, no. 4, 2011.

[22] J. Goldman, K. Shilton, J. Burke, D. Estrin, M. Hansen, N. Ramanathan, S. Reddy, V. Samanta, M. Srivastava, and R. West, "Participatory sensing: A citizen-powered approach to illuminating the patterns that shape our world," *Foresight & Governance Project, White Paper*, pp. 1–15, 2009.

[23] A. J. Oliner, A. P. Iyer, I. Stoica, E. Lagerspetz, and S. Tarkoma, "Carat: Collaborative energy diagnosis for mobile devices," in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '13, 2013.

[24] B. L. Sullivan, C. L. Wood, M. J. Iliff, R. E. Bonney, D. Fink, and S. Kelling, "ebird: A citizen-based bird observation network in the biological sciences," *Biological Conservation*, vol. 142, no. 10, pp. 2282 – 2292, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S000632070900216X

[25] J. Ross, L. Irani, M. S. Silberman, A. Zaldivar, and B. Tomlinson, "Who are the crowdworkers?: Shifting demographics in mechanical turk," in *CHI '10 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '10, 2010, pp. 2863–2872.

[26] L. B. Chilton, J. J. Horton, R. C. Miller, and S. Azenkot, "Task search in a human computation market," in *Proceedings of the ACM SIGKDD Workshop on Human Computation*, ser. HCOMP '10, 2010.

[27] W. Mason and D. J. Watts, "Financial incentives and the performance of crowds," *ACM SigKDD Explorations Newsletter*, vol. 11, no. 2, pp. 100–108, 2010.

[28] D. Oleson, A. Sorokin, G. P. Laughlin, V. Hester, J. Le, and L. Biewald, "Programmatic Gold: Targeted and Scalable Quality Assurance in Crowdsourcing," in *Proc. of HCOMP*, 2011.

[29] P. Dai, Mausam, and D. S. Weld, "Decision-Theoretic Control of Crowd-Sourced Workflows," in *Twenty-Fourth National Conference on Artificial Intelligence*, 2010.

[30] ——, "Artificial Intelligence for Artificial Artificial Intelligence," in *Proc. of HCOMP*, 2011.

[31] G. Little, L. B. Chilton, M. Goldman, and R. C. Miller, "Turkit: human computation algorithms on mechanical turk," in *Proceedings of the 23nd annual ACM symposium on User interface software and technology*. ACM, 2010, pp. 57–66.

[32] P. G. Ipeirotis, F. Provost, and J. Wang, "Quality management on Amazon Mechanical Turk," in *Proc. of HCOMP*, 2010.

[33] M. S. Bernstein, G. Little, R. C. Miller, B. Hartmann, M. S. Ackerman, D. R. Karger, D. Crowell, and K. Panovich, "Soylent: a word processor with a crowd inside," in *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, ser. UIST '10, 2010, pp. 313–322.

[34] A. Kittur, B. Smus, S. Khamkar, and R. E. Kraut, "CrowdForge: Crowdsourcing Complex Work," in *Proceedings of the 24th annual ACM symposium on User interface software and technology*, 2011.

[35] S. Ahmad, A. Battle, Z. Malkani, and S. Kamvar, "The jabberwocky programming environment for structured social computing," in *Proceedings of the 24th annual ACM symposium on User interface software and technology*, 2011.

[36] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig latin: a not-so-foreign language for data processing," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, 2008.

[37] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan, "Interpreting the data: Parallel analysis with sawzall," *Scientific Programming*, vol. 13, no. 4, 2005.

[38] B. H. Anand Kulkarni, Matthew Can, "Turkomatic: Automatic recursivetask and workflow design for mechanical turk," in *Proceedings of the Conference on Human Factors in Computing Systems (WIP)*, ser. CHI '11, 2011.

[39] J. P. Bigham, C. Jayant, H. Ji, G. Little, A. Miller, R. C. Miller, R. Miller, A. Tatarowicz, B. White, S. White, and T. Yeh, "Vizwiz: Nearly real-time answers to visual questions," in *Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '10. New York, NY, USA: ACM, 2010, pp. 333–342. [Online]. Available: http://doi.acm.org/10.1145/1866029.1866080

[40] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate record detection: A survey," *IEEE Trans. on Knowl. and Data Eng.*, vol. 19, no. 1, Jan. 2007.

[41] S. Sarawagi and A. Bhamidipaty, "Interactive deduplication using active learning," in *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '02, 2002.

[42] S. R. Jeffery, M. J. Franklin, and A. Y. Halevy, "Pay-as-you-go user feedback for dataspace systems," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '08, 2008.

[43] J. Wang, T. Kraska, M. J. Franklin, and J. Feng, "Crowder: Crowdsourcing entity resolution," *Proc. VLDB Endow.*, vol. 5, no. 11, July 2012.

[44] M. S. Bernstein, J. Teevan, S. Dumais, D. Liebling, and E. Horvitz, "Direct answers for search queries in the long tail," in *CHI*, 2012.

[45] T. Yan, V. Kumar, and D. Ganesan, "Crowdsearch: Exploiting crowds for accurate real-time image search on mobile phones," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '10. New York, NY, USA: ACM, 2010, pp. 77–90. [Online]. Available: http://doi.acm.org/10.1145/1814433.1814443

[46] J. D. Ullman, H. Garcia-Molina, and J. Widom, *Database systems: the complete book*. Prentice Hall Upper Saddle River, 2001, vol. 2.

[47] E. F. Codd, "A relational model of data for large shared data banks," *Communications of the ACM*, vol. 13, no. 6, 1970.

[48] ——, *Relational completeness of data base sublanguages*. IBM Corporation, 1972.

[49] S. Chaudhuri, "An overview of query optimization in relational systems," in *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, 1998.

[50] A. G. Parameswaran, H. Park, H. Garcia-Molina, N. Polyzotis, and J. Widom, "Deco: declarative crowdsourcing," in *CIKM*, 2012.

[51] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller, "Human-powered sorts and joins," *Proc. VLDB Endow.*, vol. 5, no. 1, Sept. 2011.

[52] A. G. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom, "Crowdscreen: Algorithms for filtering data with humans," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 2012, pp. 361–372.

[53] A. Parameswaran, S. Boyd, H. Garcia-Molina, A. Gupta, N. Polyzotis, and J. Widom, "Optimal Crowd-Powered Rating and Filtering Algorithms," *PVLDB*, 2014.

[54] A. Das Sarma, A. Parameswaran, H. Garcia-Molina, and A. Halevy, "Crowd-powered find algorithms," in *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, 2014.

[55] V. Polychronopoulos, L. de Alfaro, J. Davis, H. Garcia-Molina, and N. Polyzotis, "Human-powered top-k lists." in *WebDB*, 2013.

[56] S. B. Davidson, S. Khanna, T. Milo, and S. Roy, "Using the crowd for top-k and group-by queries," in *Proceedings of the 16th International Conference on Database Theory*. ACM, 2013, pp. 225–236.

[57] P. Venetis, H. Garcia-Molina, K. Huang, and N. Polyzotis, "Max algorithms in crowdsourcing environments," in *Proceedings of the 21st international conference on World Wide Web*, 2012.

[58] S. Guo, A. Parameswaran, and H. Garcia-Molina, "So Who Won? Dynamic Max Discovery with the Crowd," in *Proc. of SIGMOD*, 2012.

[59] R. Gomes, P. Welinder, A. Krause, and P. Perona, "Crowdclustering," in *Proc. of NIPS*, 2011.

[60] D. W. Barowy, E. D. Berger, and A. McGregor, "AUTOMAN: A Platform for Integrating Human-Based and Digital Computation," University of Massachusetts, Tech. Rep., 2011.

[61] K.-T. Chen, C.-C. Wu, Y.-C. Chang, and C.-L. Lei, "A crowdsourceable QoE evaluation framework for multimedia content," in *ACM Multimedia*, 2009.

[62] R. K. Colwell and J. A. Coddington, "Estimating Terrestrial Biodiversity through Extrapolation," *Philosophical Trans.: Biological Sciences*, vol. 345, no. 1311, 1994.

[63] A. Chao, "Species Richness Estimation," Technical Report, 2005.

[64] I. J. Good, "The Population Frequencies of Species and the Estimation of Population Parameters," *Biometrika*, vol. 40, no. 3/4, 1953.

[65] W. Feller, *An Introduction to Probability Theory and Its Applications*. Wiley, 1968, vol. 1.

[66] P. Flajolet *et al.*, "Birthday Paradox, Coupon Collectors, Caching Algorithms and Self-Organizing Search," *Discrete Appl. Math.*, vol. 39, November 1992.

[67] K. P. Burnham and W. S. Overton, "Estimation of the Size of a Closed Population when Capture Probabilities vary Among Animals," *Biometrika*, vol. 65, no. 3, 1978.

[68] A. Chao, "Nonparametric Estimation of the Number of Classes in a Population," *SJS*, vol. 11, no. 4, 1984.

[69] A. Chao and S. Lee, "Estimating the Number of Classes via Sample Coverage," *Journal of the American Statistical Association*, vol. 87, no. 417, pp. 210–217, 1992.

[70] P. J. Haas, J. F. Naughton, S. Seshadri, and L. Stokes, "Sampling-based estimation of the number of distinct values of an attribute," in *Proc. of VLDB*, 1995.

[71] A. Shlosser, "On Estimation of the size of the dictionary of a long text on the basis of a sample," *Engrg. Cybernetics*, vol. 19, 1981.

[72] M. Charikar, S. Chaudhuri, R. Motwani, and V. Narasayya, "Towards Estimation Error Guarantees for Distinct Values," in *Proc. of PODS*, 2000.

[73] J. Bunge, M. Fitzpatrick, and J. Handley, "Comparison of three estimators of the number of species," *Journal of Applied Statistics*, vol. 22, no. 1, 1995.

[74] J. Heer and M. Bostock, "Crowdsourcing graphical perception: using mechanical turk to assess visualization design," in *Proc. of CHI*, 2010.

[75] T. Shen, A. Chao, and C. Lin, "Predicting the Number of New Species in Further Taxonomic Sampling," *Ecology*, vol. 84, no. 3, 2003.

[76] R. K. Colwell, C. X. Mao, and J. Chang, "Interpolating, Extrapolating, and Comparing Incidence-Based Species Accumulation Curves," *Ecology*, vol. 85, no. 10, 2004.

[77] J. D. Brutlag and T. S. Richardson, "A Block Sampling Approach to Distinct Value Estimation," *JCGS*, vol. 11, no. 2, 2002.

[78] P. B. Gibbons, "Distinct Sampling for Highly-Accurate Answers to Distinct Values Queries and Event Reports," in *Proc. of VLDB*, 2001.

[79] A. Broder, M. Fontura, V. Josifovski, R. Kumar, R. Motwani, S. Nabar, R. Panigrahy, A. Tomkins, and Y. Xu, "Estimating corpus size via queries," in *Proc. of CIKM*, 2006.

[80] K.-L. Liu, C. Yu, and W. Meng, "Discovering the representative of a search engine," in *Proc. of CIKM*, 2002.

[81] Z. Bar-Yossef and M. Gurevich, "Efficient search engine measurements," *ACM Trans. Web*, vol. 5, no. 4, pp. 18:1–18:48, Oct. 2011.

[82] J. Lu and D. Li, "Estimating deep web data source size by capture—recapture method," *Inf. Retr.*, vol. 13, no. 1, pp. 70–95, Feb. 2010.

[83] J. Liang, "Estimation Methods for the Size of Deep Web Textural Data Source: A Survey," cs.uwindsor.ca/richard/cs510/survey_jie_liang.pdf, 2008.

[84] R. Nakatsu and E. Grossman, "Designing effective user interfaces for crowdsourcing: An exploratory study," in *Human Interface and the Management of Information*, ser. Lecture Notes in Computer Science, 2013, vol. 8016, pp. 221–229.

[85] P. Gutheim and B. Hartmann, "Fantasktic: Improving Quality of Results for Novice Crowdsourcing Users," UCB/EECS-2012-112, Tech. Rep., 2012.

[86] J. Gray, P. Sundaresan, S. Englert, K. Baclawski, and P. J. Weinberger, "Quickly generating billion-record synthetic databases," in *Proc. of SIGMOD*, 1994.

[87] A. Parameswaran, M. H. Teh, H. Garcia-Molina, and J. Widom, "An Expressive and Accurate Crowd-Powered Search Toolkit," in *Proc. of HCOMP*, 2013.

[88] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[89] O. Alonso and M. Lease, "Crowdsourcing for information retrieval: principles, methods, and applications," in *SIGIR*, 2011.

[90] O. Alonso, D. E. Rose, and B. Stewart, "Crowdsourcing for relevance evaluation," *SIGIR Forum*, vol. 42, no. 2, 2008.

[91] C. Grady and M. Lease, "Crowdsourcing document relevance assessment with mechanical turk," ser. CSLDAMT '10, 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1866696.1866723

[92] S. Nowak and S. Rüger, "How reliable are annotations via crowdsourcing: a study about inter-annotator agreement for multi-label image annotation," ser. ACM MIR, 2010. [Online]. Available: http://doi.acm.org/10.1145/1743384.1743478

[93] C. Rashtchian, P. Young, M. Hodosh, and J. Hockenmaier, "Collecting image annotations using amazon's mechanical turk," ser. CSLDAMT, 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1866696.1866717

[94] G. Demartini, B. Trushkowsky, T. Kraska, and M. J. Franklin, "Crowdq: Crowdsourced query understanding," in *CIDR*, 2013.

[95] M. Shokouhi and L. Si, "Federated search," *Found. Trends Inf. Retr.*

[96] E. Selberg and O. Etzioni, "Multi-service search and comparison using the metacrawler," in *WWW*, 1995.

[97] J. A. Aslam and M. Montague, "Models for metasearch," in *ACM SIGIR*, 2001.

[98] A. Anagnostopoulos, A. Z. Broder, and D. Carmel, "Sampling search-engine results," in *WWW*, 2005.

[99] V. Hristidis, Y. Hu, and P. Ipeirotis, "Relevance-based retrieval on hidden-web text databases without ranking support," *IEEE TKDE*, 2011.

[100] C. Watkins, "Learning from Delayed Rewards," Ph.D. dissertation, Cambridge University, 1989.

[101] S. Bubeck and N. Cesa-Bianchi, "Regret analysis of stochastic and nonstochastic multi-armed bandit problems," *CoRR*, 2012.

[102] A. Slivkins, F. Radlinski, and S. Gollapudi, "Learning optimally diverse rankings over large document collections," in *ICML*, 2010.

[103] A. Deshpande, Z. Ives, and V. Raman, "Adaptive query processing," *Found. Trends databases*, vol. 1, no. 1, Jan. 2007.

[104] R. Avnur and J. M. Hellerstein, "Eddies: Continuously adaptive query processing," in *Proc. of SIGMOD*, 2000.

[105] J. M. Hellerstein and M. Stonebraker, "Predicate migration: optimizing queries with expensive predicates," in *Proc. of SIGMOD*, 1993.