Otto-von-Guericke-University Magdeburg

Faculty for Computer Science

Department of Technical and Business Information Systems

Master Thesis

An Overview and Classification of current research on Crowdprocessing and Databases

Author:
Xiao Chen
January 17, 2014

Supervisor:
Dr.-Ing. Eike Schallehn

University Magdeburg
Faculty for Computer Science
P.O.Box 4120, D-39016 Magdeburg
Germany

Xiao Chen:
An Overview and Classification of current research on crowdprocessing and databases
Master thesis, Otto-von-Guericke-University Magdeburg, 2014

Acknowledgements

How time flies! At the beginning of 2014, I finished my master thesis finally. I appreciate Dr. Schallehn very much and a simple ‘thanks’ is not enough to express my gratitude. Without Dr. Schallehn, maybe I have not yet started my thesis. His guide and suggestions benefit all my life. I am very thankful to Ziqiang and Siba, thank you for your guide, suggestions, and understanding. At last, thanks for the understanding and the support from my whole family.

Contents

List of Figures

List of Tables

List of Abbreviations

AMT	**A**mazon **M**echanical **T**urk
API	**A**pplication **P**rogram **I**nterface
AI	**A**rtificial **I**ntelligence
CS	**C**rowdsourcing **S**ystem
CDAS	**C**rowdsourcing **D**ata **A**nalytics **S**ystem
DAG	**D**irected **A**cyclic **G**raph
DBMS	**D**ata**B**ase **M**anagement **S**ystem
DDL	**D**ata **D**efinition **L**anguage
DML	**D**ata **M**anipulation **L**anguage
DCL	**D**ata **C**ontrol **L**anguage
GWAP	**G**ames **W**ith **A** **P**urpose
HIT	**H**uman **I**ntelligent **T**ask
HumanGS	**H**uman-assisted **G**raph **S**earch
IMDb	**I**nternet **M**ovie **D**ata**b**ase
TC	**T**ransaction **C**ontrol
SQL	**S**tructured **Q**uery **L**anguage
SCC	**S**trongly **C**onnected **C**omponents
UDF	**U**ser **D**efined **F**unction
UI	**U**ser **I**nterface

1. Introduction

Artificial Intelligence has flourished for many years, but it has some intrinsic limits on certain areas, which could not be eliminated in a very long time, such as object identification[/1/] and emotion analysis. Hence, human intelligence has to be harnessed to better and more efficiently resolve problems in those areas. However, employing skilled people to fulfill such tasks requires a tremendous expenditure and nowadays companies put the economic interest first. Therefore, a new pattern named crowdsourcing arises recurring to the omnipresent Internet[/2/]: Companies, enterprises or other organizations outsource a relatively simple task to the crowd all over the world via the Internet and require only a very low pay or even no pay. In this way, cost is driven down and the objective to maximize the benefit for the task providers can be achieved.

1.1 Motivation

With more and more successful examples recurring to crowdsourcing, such as Wikipedia, Amazon Mechanical Turk (AMT), which is a crowdsourcing marketplace, crowdsourcing has attracted a mass of research, in particular on crowd data sourcing, where the crowd's task is to generate or 'source' data[/2/]. This thesis focuses on crowd data sourcing and attempts to provide an overview of current research on crowd data sourcing and classify them, thus let researchers, who interest in the crowd data sourcing, understand crowd data sourcing quickly and clearly, in turn choose a best fit direction and contribute themselves properly for crowd data sourcing.

1.2 Goal

The general goal of this thesis is to provide an overview of current research on crowd data sourcing and classify them in perspective. To be specific, the following questions are raised from both the crowdsourcing perspective and the database perspective. This thesis aims at answering them:

- **Crowdsourced Databases:** The emergence of something new means that we have to change a lot in the established systems, so does crowdsourcing. Databases have developed a sophisticated mechanism for the traditional data processing, but with the emergency of crowdsourcing, what changes should be made with respect to databases? What are current crowdsourcing databases? What are similarities and differences among them? Are there ideas, which are proposed in one crowdsourced database but may be extended to other crowdsourced databases?
- **Crowd-based data processing:** What is the major current research on crowd-based data processing? Which specific aspects does the research focus on? Can strategies or algorithms for specific crowd-based data processing be extended to other crowd-based data processing?
- **Difficulties in Crowd-based Data Processing:** The ways that people perform a task and the computer performs a task are completely different. In contrast to a computer, people are prone to make mistakes; besides, if a task belongs to a subjective task, different people will have diverse views and understanding. To handle this problem, a microtask is usually assigned to several people. Accordingly, a series of questions should be considered: To how many people should be a same microtask assigned? How to handle people's different responses for one microtask in order to get most-valued answers? In addition, different microtask designs will lead to different performance, what is the best way to design a microtask? Facing these implementation difficulties, what countermeasures have been proposed? What innovative techniques of crowd-based data processing are designed to optimize the crowdsourcing service?

1.3 Structure

The remainder of this thesis is structured as follows:

Chapter 2 In Chapter 2, the background on crowdsourcing and the used terminology are introduced. First, the foundations of crowdsourcing are presented, including a general definition, successful crowdsourcing examples and a conclusion of suitable situations for crowdsourcing. Then, different classifications of crowdsourcing systems on the Internet are summarized, and a new classification is suggested. Finally, the overlapping between crowdsourcing and database is introduced.

Chapter 3 Since some crowd-based data processing relies on crowdsourced databases, in this chapter three crowdsourced databases are first demonstrated. They are CrowdDB, Qurk, Deco, which are presented one by one. The description of three crowdsourced databases is from the perspective of their architecture, data model, query language, query processing and user interfaces. At last, the three crowdsourced databases are compared with each other.

Chapter 4 In Chapter 4, different kinds of crowd-based data processing are introduced one by one. First, crowd-based database queries are presented and they are the core part of this chapter. Next, other crowd-based data processing is only introduced briefly. The current research on crowd-based database queries studies various crowdsourcing issues. Since the overall database query processing flow can be easily implemented with the help of crowdsourced databases, the major research aims at solving different difficulties of specific crowd-based database queries. Therefore, the last part of this chapter concludes the difficulties in crowd-based database queries and the countermeasures to these difficulties in different research. Countermeasures to the same difficulty in different research are compared. Besides, crowd-based database queries and other crowd-based data processing are placed together to be compared.

Chapter 5 The last major discussion is about the innovative optimization techniques of crowd-based data processing. The optimization techniques involve various issues in crowdsourcing, such as the difficulties mentioned in Chapter 4, index techniques in crowdsourcing environments.

Chapter 6 Finally, the conclusion of the whole thesis is given.

2. Background

In this chapter, the background of the thesis is provided. First, crowdsourcing is defined and factors leading to its flourish are explained, then classifications of major crowdsourcing systems on the Internet are provided. At last, the emphasis of this thesis crowd data sourcing is introduced, and the overlapping between crowdsourcing and databases is depicted.

2.1 Foundations of crowdsourcing

At the beginning, people make use of computers mainly for specific domains where relies on the strong calculation power of computers, then at the end of the 20th century computer becomes popular and goes into people's daily life. Many researchers saw the benefits computer has brought to people and hoped to enable computer contributing more, then Artificial Intelligence (AI) was rapidly developed. In recent years, people have to face the reality that they cannot totally rely on computers. Many tasks are really hard for computers even cannot be performed by computers, and they have to be on their own. On the other hand, the Internet rises and breaks the restriction of regions for a task, people all over the world can via the Internet perform a common task. Thus, economic interest is more and more important in today's world. All these issues have facilitated the appearance of crowdsourcing, so to speak that crowdsourcing is a product of the age.

2.1.1 Definition of Crowdsourcing

"Crowdsourcing" was first coined by Jeff Howe and Mark Robinson after the conversation about how businesses were using the Internet to outsource work to individuals in 2005. They concluded that this business phenomenon was like "outsourcing to the crowd" which quickly

gave rise to the portmanteau "crowdsourcing"[3], and then Howe published "The Rise of Crowdsourcing" in June 2006, which gives a definition for "crowdsourcing":

> *"Simply defined, crowdsourcing represents the act of a company or institution taking a function once performed by employees and outsourcing it to an undefined (and generally large) network of people in the form of an open call. This can take the form of peer-production (when the job is performed collaboratively), but is also often undertaken by sole individuals.The crucial prerequisite is the use of the open call format and the large network of potential laborers".*[4]

Since then, a variety of definitions is provided by different researchers according to their own specialities. In order to get a general definition of crowdsourcing, a statistic is made by Enrique Estellés-Arolas and Fernando González-Ladrón-de-Guevara[5]. According to their statistic, there are up to 40 original definitions of crowdsourcing were given through 209 documents from five popular research databases: ACM, IEEE, ScienceDirect, SAGE and Emerald, and they at last conclude a general definition for crowdsourcing:

> *"Crowdsourcing is a type of participative online activity in which an individual, an institution, a non-profit organization, or company proposes to a group of individuals of varying knowledge, heterogeneity, and number, via a flexible open call, the voluntary undertaking of a task. The undertaking of the task, of variable complexity and modularity, and in which the crowd should participate bringing their work, money, knowledge and/or experience, always entails mutual benefit. The user will receive the satisfaction of a given type of need,be it economic, social recognition, self-esteem, or the development of individual skills, while the crowdsourcer will obtain and utilize to their advantage that what the user has brought to the venture, whose form will depend on the type of activity undertaken."*

In this thesis, crowdsourcing is expected to be defined generally; thus this definition is adopted.

2.1.2 Successful crowdsourcing examples

In this section, many successful crowdsourcing examples are enumerated.

Wikipedia: Wikipedia is a well-known free encyclopedia that anyone can edit. Figure 2-1[6] enumerates the number of articles in different languages, which are contributed by people

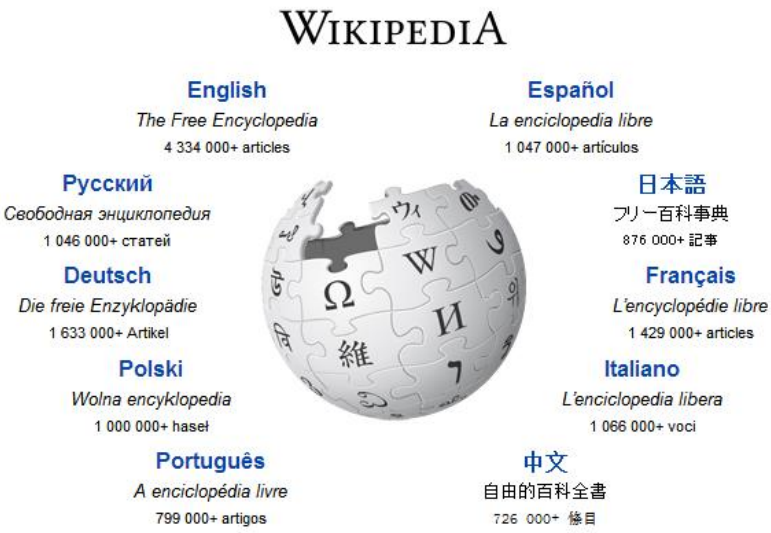

Figure 2-1: Wikipedia[/6/]

from all over the world. In particular, there are already more than 4,334,000 articles in English. In total, there are 287 languages for which official Wikipedia has been created[/7/].

Kaggle: Kaggle is the world's largest community of data scientists. The enterprises or other organizations, such as NASA, which have technical problems, can send the data and the corresponding problem to Kaggle via the Internet. Kaggle publishes this problem, any scientist from all over the world can try to solve the problem and at the same time compete with other scientists. Kaggle ranks scientists according to their contributions. So far, there are at least 30000 scientists in one competition, which submit at least one model to a problem[/8/].

AMT (Amazon Mechanical Turk): AMT is a crowdsourcing marketplace. Individuals or organizations can put tasks on the AMT, which are not efficient or impossible performed by computers, and people from all over the world can perform these tasks via the Internet. There are over 500,000 workers from over 190 countries in January 2011[/9/]. At present, there are more than 330,000 tasks available on it.

IMDb: IMDb is short for Internet Movie Database, which is the world's most popular and authoritative source for movie, TV and celebrity content. People can via the Internet comment and rate movies, TVs or other media. It is one of the 50 most-visited websites. So far, IMDb has 2,692,062 titles and 5,546,740 personalities in its database[/10/].

ESP Game: A "game with a purpose (GWAP) " is a human-based computation technique in which a computational process outsources its certain steps to humans in an entertaining way[/11/]. "ESP" is a successful example of GWAP, in which users are asked to tag images and

get a reward when two users adopt the same label at the same time. In 2006, ESP has been licensed to Google to classify images and offer a better service to Google Images Search users[/12/].

Tagasauris: Tagasauris is founded in New York in December 2012. It provides media annotation services by using crowdsourcing, gamification, machine intelligence, and semantics. Magnum photos cooperate with it to allow the digital archive of Magnum to be searchable. A notable accomplishment is: it helped Magnum photos to find some "lost" photos in the Magnum archive from the movie "American Graffiti"[/13/].

99designs: 99designs is the largest online marketplace for crowdsourced graphic design with over 200,000 designers. Each week there are more than 1,800 new contests, and about two million dollars are paid out to designers each month[/14/].

YouTube: YouTube is a video sharing website, on which users can upload, view, share and rate videos. On YouTube, 60 hours of video are uploaded every minute, over 4 billion videos are viewed a day, over 800 million unique users visit YouTube each month[/15/].

All crowdsourcing examples mentioned above are the most famous ones in a variety of crowdsourcing systems. In section 2.2, classifications of major crowdsourcing systems will be introduced.

2.1.3 Factors leading to the flourish of Crowdsourcing

There are always reasons for everything that happens, just like crowdsourcing; many factors lead to its emergence and prosperity. In the following, these factors will be discussed for one by one.

Computer vs. People:

The electronic digital computer stepped onto the historical stage in the 1940s[/16/] and became mainstream in the 1990s[/17/]. Nowadays computer has almost entered into every corner of our life; it makes our life more colourful and helps us to solve various problems. At the beginning computers are mainly programmed to accomplish the computation tasks, then computers are expected to do more, so a branch of computer science "AI" was developed, which aiming at mimicking the human intelligence. Although computer science has developed for many years,

it still has conspicuous limits on performing a number of tasks. Following are typical areas where people outperform than computer.

- **Object Recognition:** Its task is finding and identifying objects in an image or video sequence[/18/]. Human even little children can effortlessly recognize a multitude of objects according to the knowledge gained along with their grows, even though the objects differ in size, scale, colour, position in image or video or other dimensions. While for computers object recognition is still a big challenge. So far, the appearance-based method and the feature-based method are used for computer object recognition[/18/]. Computers recognize an object without cognitive competence but only based on preset strategies. Facing unpredictable situations, computers often cannot efficiently and accurately accomplish object recognition.
- **Entity Resolution:** Entity resolution means the same as object matching, duplicate identification, record linkage, or reference reconciliation. Its task is to identify entities referring to the same real-world entity[/19/]. Due to different formats, different names, abbreviations or input errors, a same real-world entity is often expressed differently. Human can identify entities easily with the help of their common sense and understanding in most cases. There is a variety of machine-based methods that have been developed. Some of them try to perform better recurring to machine learning, but are still less than satisfaction[/19/ /20/].
- **Data Integration:** Its task is to combine data residing at different sources, and to provide the user with a unified view of these data[/21/]. A variety of methods to automatically integrate data has been developed. However, they typically provide partial or incorrect results. Data integration is more and more imperative because of the more and more large-scale data, and more and more cases require a very high-quality result of data integration. Hence, human refinements based on the results of automatical data integration are necessary and significant[/22/].
- **Nature Language Translation:** Our world nowadays is more and more connectable, people often have dealings with others who speak different languages. Mastering a foreign language is not an easy thing. As a result, people sometimes have to turn to a translator. As we well know, machine translators, such as Google translator, cannot provide a satisfactory result, even sometimes provide a very ridiculous translation.

People who know two-side of languages well could accomplish translation task perfectly.

- **Image description:** Given an image; people can easily describe it in their own word, while computers cannot.
- **Ranking:** People often participate in social activities. After those experiences, they can rate and rank restaurants, films or products. These kinds of behaviours cannot happen for a computer.
- **Subjective comparison:** Computers do not really have a brain; they cannot understand the subjective parameters and cannot have its own feeling for an object. Thus, facing subjective comparison computers are also helpless.
- **Complex Information Retrieval:** Nowadays search engines become more and more intelligent; sometimes people are surprised by the closely matched results. Nevertheless, they cannot handle everything. They are good at processing a single query, but weak at processing very complex queries, for they cannot understand the fact that you are looking for[/23/]. For instance, a user wants to find out the average age of presidents of universities in German. She or he might try to query for 'birth dates of the presidents of all universities in German'. Unfortunately, a search engine cannot answer such a query.
- **Creative work:** Many creative ideas usually make people applaud, while computers are totally stranded with creative works.

Dedicated People vs. Crowd:

So far, the tasks that are inefficiently performed or cannot be performed by computer are enumerated. But there is still a question, why not directly employing dedicated people for these tasks? The reasons for using crowdsourcing instead of dedicated people are two-sided:

- **Economic Interest:** Our Society is full of competition for people and also for companies. Under the serious competition, the payment for dedicated people has to be taken into account. If certain tasks can be fulfilled by crowdsourcing with a very low pay instead of dedicated people, why not adopting this new pattern?
- **The Wisdom of Crowds:** Imagine a crowd of people, for a technical task, the result of the person who is best on it will be better than the average result of the crowd. While for some other kinds of task, such as decision making, the crowd works always better than any individual[/24/].

Objective Conditions:

In fact, long ago there was already crowdsourcing-like pattern, which utilized distributed people to accomplish tasks. However, crowdsourcing flourishes only in recent years; this is because omnipresent internet makes crowdsourcing more comprehensive and feasible. Even more convenient, people can access the Internet through their cell phones at any time.

Performance Evaluation:

Crowdsourcing has developed for many years and attracted a mass of research on it. There was already reaseach, which proved the efficiency of crowdsourcing, as long as crowdsourcing is with the proper control. For example, Steven Komarov, Katharina Reinecke, and Krzysztof Z. Gajos conducted three experiments with a different level of attention from the participants both in the lab and on AMT, which is a crowdsourcing Internet marketplace and will be introduced in detail in Section 2.4. There was no significant difference between the results received from the lab and AMT. In these three experiments, the primary measures of interest were task completion times and error rates[25]; the results proved that crowdsourcing could be a useful pattern to perform a variety of tasks.

All above in this section are the factors why crowdsourcing obtains the attention and the success. Consequently, crowdsourcing has floured nowadays on the Internet. Next, the classification of existing crowdsourcing systems (CS) on the Internet is provided.

2.2 Classification of existing Crowdsourcing Systems

The power of crowdsourcing cannot be ignored, a variety of crowdsourcing systems emerges on the worldwide web and brings the system owner considerable benefit. Broadly speaking, crowdsourcing system is a system that enlists a crowd of humans to help solve a problem defined by the system owners[26]. In this section, many classifications regarding crowdsourcing, which consider different dimensions of crowdsourcing in different fields, are described. Then, a new classification of existing crowdsourcing systems is provided with some specific CS examples. Since each reference has its own descriptions of different dimensions, and in fact, many dimensions with different descriptions are the same, only the dimensions in each reference are first introduced, the explanations of these dimensions are summarized later.

Doan, Ramakrishnan, and Halevy[/24/] gave a global picture of crowdsourcing systems on the web and classified CSs according to the following nine dimensions: nature of collaboration, type of target problem, how to recruit and retain users, what users can do, how to combine contributions, how to evaluate inputs, degree of manual effort, role of human users and standalone vs. piggyback. David, Stefan, Thimo, Robert, and Martin[/27/] distinguished among crowdsourcing processes on four dimensions: aggregation of contributions, accessibility of contributions, remuneration for contribution, and preselection of contributors. Corney, Torres-Sanchez, Jagadeesan, and Regli[/28/] provided a foundation for identifying methodologies or analysis methods for outsourcing on three dimensions: nature of the task, nature of the crowd and nature of the payment[/27/]. Malone, Laubacher, and Dellarocas[/29/] identified the building blocks of collective intelligence approaches with what (goal), who (staffing), why (incentives) and how (structure/process)[/27/]. Piller, Ihl, and Vossen[/30/] analyzed strategies for customer participation in open innovation: stage in the innovation process, degree of collaboration and degrees of freedom[/27/]. Rouse[/31/] tried to clarify the notion of crowdsourcing through nature of the task/supplier capabilities, distribution of benefits and forms of motivation[/27/]. Schenk and Guittard[/ 32 /] understood crowdsourcing from a management science perspective on two dimensions: integrative/selective nature of the process and type of tasks[/27/]. Zwass[/33/] showed a taxonomic framework as a prerequisite for theory building in co-creation research[/27/].
Among these references, some terms mean the same. For example, “how to combine contributions” and “aggregation of contributions”, “performers” and “preselection of contributors”. In summary, the following main dimensions appeared in the references to classify the crowdsourcing system are presented:

Types of Tasks: Certain papers emphasize the complexity of the task, and classify tasks into simple, moderate and sophisticated task[/31/] . Actually, the level of skills that a task requires makes more sense than the level of the complexity. Therefore, in this thesis tasks are classified according to skill levels.

- **Commonsense tasks:** A big part of tasks are the tasks, which do not need specific knowledge and can be performed by most people as long as the contributor has a common sense, such as labelling an image, explaining a word and so on.

- **Moderate tasks:** The skills need contributors to make are not very restricted, usually people do not require a high-level education to get. For example, translating German to English, design a t-Shirt.
- **Skilled tasks:** The most restrict situation is that a task requires professional skills. People must get this skill through systematical and dedicated learning. For instance, writing programs in C# or designing an elaborate ornament for game characters with the dedicated software.

In addition, no matter a task belongs to commonsense tasks, moderate tasks or skilled tasks, it can also be classified into individual tasks or cooperative tasks according to whether they require the collaboration among contributors or not.

Motivation of Contributors: This dimension means that which factors prompt people to contribute. There are mainly four kinds of motivations: money, for fun, altruism, and self-satisfaction.

Qualification of Contributors: This dimension means that whether any contributor can perform the tasks of a crowdsourcing system as they wish, whether a system requires to test the contributors' skill level.

Rights of Contributors: This dimension means that what manipulation a contributor can make to other contributor's contributions. Permission from low to high is nothing, view, rating, and modification.

Aggregations of Contributions: The strategies to aggregate different contributions are still not the same. Some crowdsourcing systems show all contributions without combination; some crowdsourcing systems select a best solution from all contributions; the others integrate all contributions to get a unified solution[/26//27/].

Supervisions on Contributors: Many crowdsourcing systems need to supervise contributors and evaluate their contributions to create a better environment for systems. Generally speaking, crowdsourcing systems, which require higher skilled people, need more supervision.

Stakeholders of Contributions: This dimension means that who benefits from the contributions. Only the system owner or users in the community at well. For example, the Amazon product rating system benefits not only Amazon to improve the service, but also is valuable to other consumers who want to buy such kind of products, while contributions on AMT benefit only the requesters.

Mainstream CSs	Types of tasks	Motivations of Contributors	Qualification of contributors	Rights of contributors	Aggregation of Contributions
encyclopaedia (e.g. Wikipedia)	commonsense or moderate; cooperative.	altruism self-satisfaction	no need	modification	integration
Crowdsourcing Marketplaces (e.g.AMT,CrowdFlower)	commonsense or moderate or skilled; individual or cooperative.	money	no need or qualification	nothing or rating	integration or selection
Rating Systems (e.g. product or film rating)	commonsense	altruism	no need	view	nothing
Game on purpose (e.g. ESP)	commonsense	for fun	no need	nothing	integration
Design systems (e.g.Threadless,99designs)	moderate or skilled	Money; for fun; Self-satisfaction.	no need	view And Rating	selection
Video Sharing (e.g. YouTube)	commonsense	for fun or self-satisfaction	no need	view and Rating	nothing

Table 2-1: Classification of Crowdsourcing Systems

Automatical Degree of Systems: This dimension means that which degree of efforts the crowdsourcing owner needs to do. Some systems have to carefully collect evidences to make sure a malicious user, if someone is reported by other contributors. While some systems like Wikipedia just let users do almost everything, including editing, merging and other manipulations[/26/].

Standalone or Piggyback: In contrast to systems that totally rely on themselves to manage the systems, many systems piggyback on a well-established system. For example, there are systems, which recommend products by searching users' purchases in other stores [/26/].

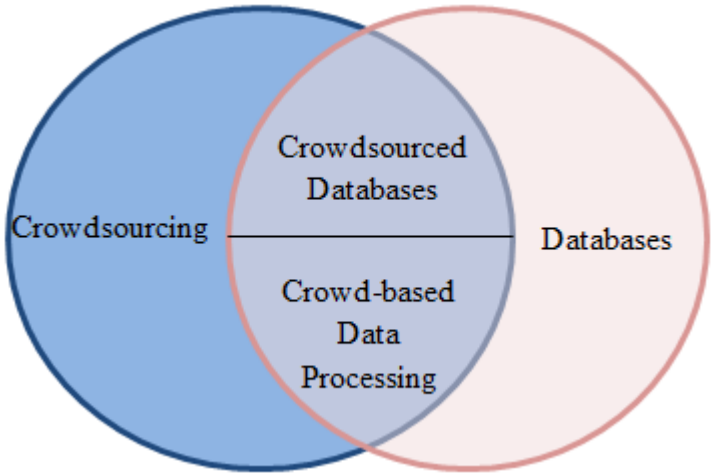

Figure 2-2: Overlapping between crowdsourcing and database

The last four dimensions are not significant for crowdsourcing; therefore the first five dimensions are as our standard to classify the mainstream crowdsourcing systems. They are presented in Table 2-1.

2.3 Overlapping Between Crowdsourcing and Database

In section 2.1 and 2.2, foundations and classifications of crowdsourcing are introduced. As can be seen that crowdsourcing has a very wide scope almost covering the whole computer science. In this thesis, crowd data sourcing is the key point, which focuses on the crowdsourcing research on the database scope. The overlapping between crowdsourcing and databases is two-fold (Figure 2-2). On the one hand, databases should be first adjusted to support the crowd-based data processing; on the other hand, crowd-based technology can help to make better and broader data processing.

2.3.1 Foundations of databases

In this section, foundations of databases related to this thesis are introduced.

Database: A database is a collection of information that is organized so that it can easily be accessed, managed, and updated[/34/].

DBMS: It is short for database management system and is a software system, which is designed to allow the definition, creation, querying, update, and administration of databases[/35/].

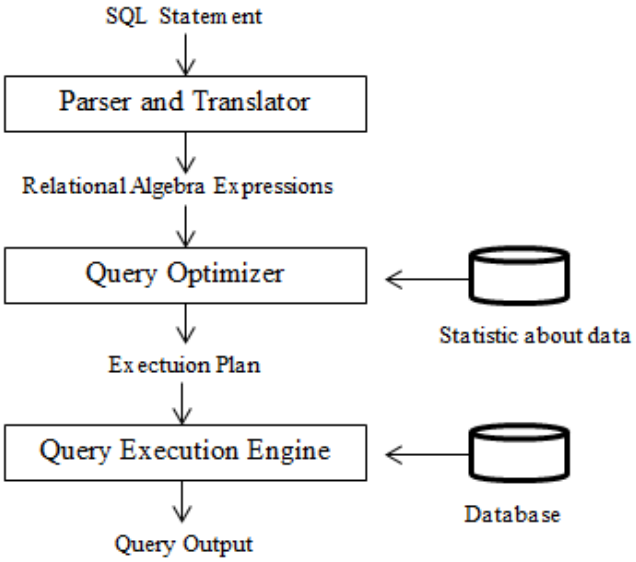

Figure 2-3: Steps in query processing[/36/]

Relational Database: A relational database is the most prevalent database that has a collection of tables of data items, all of which is formally described and organized according to the relational model[/35/] .

Data Model: A data model is a model that describes in an abstract way how data is represented in an information system or a DBMS. The data model in a relational database is called the relational model, in which data is organized in tables (relations) of records (tuples) with columns (attributes). A table can have a primary key, which is the unique identifier of rows. The primary key can be referenced from another table as a foreign key and forces integrity constraints on the data[/36/].

Query Language: A query language is a language to query the data. The Structured Query language (SQL) is the query language used in the relational database, which is based on the operators of the relational algebra and is divided to Data Definition Language (DDL), Data Manipulation Language (DML), Data Control Language (DCL), and Transaction Control (TC)[/36/]. A basic example of SQL is:

```
SELECT column
FROM table
WHERE "conditions"
```

Query Processing: Query processing is the procedure to process a query. It has three basic steps in relational databases[37] (Figure 2-3[38]):

1. Parsing and Translation: Query parser first checks syntax, verifies relations. Then, the query translator translates the query into its internal form, and then the internal form is translated into the relational algebra.

2. Optimization: The query optimizer chooses the one with the lowest cost amongst all equivalent evaluation plans, and the cost is estimated using the statistical information from the database catalog.

3. Execution: The query-execution engine takes a query-evaluation plan, executes that plan, and returns the answers to the query.

2.3.2 Crowdsourced Databases

In order to support crowdsourcing for conventional database systems, current research makes many adjustments based on the conventional databases. In the following, the motivations that traditional databases have to be adjusted are briefly explained:

Data Model: In contrast to computers, people perform the tasks with the cognitive, errors and motivational diversity. However, traditional relational databases lack abstraction for dealing with these kinds of diversity. The traditional data model is too restrained, literal, and has little fault tolerance with the diversity. Thus, in crowdsourced databases the data model is adjusted to adapt the diversity.

Query Language: A crowdsourced database system is a mixed system with computer processors and human processors. Hence, the traditional query language for the computer processor needs to be extended with crowd issues.

Query processing: Overall, the query processing still has three steps: parsing and translation, optimization and execution. Nevertheless, specific implements in each step have to be extended. In the first step, additional operators are created to support the crowd data processing. In the step of the optimization, more issues should be considered to get an optimal query result. For example, the crowd should be paid to perform tasks in most cases; consequently, the optimization on this aspect should be considered. In the last step, the query execution should be associated with AMT or other crowdsourcing services.

The motivation why a traditional database requires changes has been introduced, three crowdsourced databases will be introduced in Chapter 3.

2.3.3 Crowd-based Data Processing

The crowd-based data processing in this thesis focuses on the database queries. The other crowd-based data processing is only briefly introduced. Traditional relational databases can only process database queries, which related data is already stored in the database. For instance, the related data for a query is NULL value in the database, then the query can only be replied with a NULL. Instead, with the help of crowdsourcing, these kinds of limits are broken, although the related data for a query is NULL value in the database, the query can be replied with a right result by invoking the crowdsourcing service.

2.4 Terminology used in this thesis

Most of the research on crowdsourcing are experimented and evaluated on the popular crowdsourcing marketplace "AMT"; thus, the common terminology used in AMT is explained here.

Requesters: The individuals or organizations that have problems or tasks, which need to be solved are called requesters.
HIT: It is short for human intelligent task; it is the smallest task unit to be performed by crowds. A HIT need to be quite simple and can be solved in a relatively short time, for example, a "yes" or "no" question is asked as a HIT.
Workers: People who want to earn money on AMT are called workers.
Assignments: If a worker decides to perform a HIT on AMT, then it means this HIT is assigned to this worker.

3. Crowdsourced Databases

At present, there are the following systems to support the development of complex crowdsourcing: Turkit[/69/], CrowdForge[/39/], CrowdLang framework[/40/], Crowd4U[/41/], CTS system[/42/], CrowdDB[/43/], Qurk[/45/] and Deco[/47/]. Turkit provides a function library for implementing crowdsourcing and uses an imperative programming language to implement crowdsourcing[/42/]. CrowdForge is a MapReduce-like framework for describing complex tasks on AMT (Amazon Mechanical Turk). CrowdLang programming framework engineers complex crowdsourcing systems using Control/data flows[/40/]. Both Crowd4U and CTS system use rule-based language to implement crowdsourcing[/42/]. CrowdDB, Qurk and Deco use SQL-like languages to implement crowdsourcing and are based on the relational database model. Turkit, CrowdForge and CrowdLang framework do not relate to databases. Therefore, they are omitted. Crowd4U and CTS systems are based on databases using rule-based languages, such as the Datalog-like language Cylog in Crowd4U, which is not as widely used as SQL. Therefore, both of them are also omitted in this thesis. In this chapter CrowdDB, Qurk and Deco are described in detail. They are based on the well-known relational database model and their languages are based on SQL (Structured Query Language). The traditional relational database cannot handle certain queries such as incomplete data queries or subjective operation queries among data. These three new emerging crowdsourced databases aim at overcoming these kinds of limitations through combining the ability of the crowd and the traditional database. In order to reach this goal, the existing data model, query language and certain query processing steps in traditional relational databases are modified or extended. In the following, they will be separately introduced and at last a comparison among these three crowdsourced databases and the traditional relational database are made according to their architecture, data model, query language, query processing and user interfaces.

Before introducing these three crowdsourced databases, the economic model of AMT is introduced here, since all three crowdsourced databases connect AMT to obtain the crowd resources. As mentioned above in 2.4, the microtask is called HIT (human intelligent task) on AMT. The requesters have to pay for the workers and AMT at the same time for each HIT. Assume that each worker is paid a fixed amount of money m_c for each HIT and the requester pays AMT a fixed amount of money m_s per worker for each HIT. The paid for each HIT per worker amounts to m_c+m_x.

3.1 CrowdDB[43] [44]

CrowdDB is a crowdsourced database designed by Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. It is designed in order to correctly answer two kinds of queries that the traditional relational database cannot answer: incomplete data queries and subject comparison queries. This idea can be implemented due to the people's following two main capabilities.

Finding new data: The computer processor only knows the data, which is already stored in the database. If a query is about new data, which does not exist in the database, the computer processor cannot obtain new information automatically from elsewhere and only answer the query with a "NULL". While crowds can obtain new information that they do not know at present with the help of search engines or other reference sources. And then crowds can return the right response for a query, even though the database itself cannot answer it.

Comparing data: Crowds are good at making comparisons through their intuition and perception, which is very hard or impossible to encode in a computer algorithm. For example, the entity resolution task, compare "I.B.M." and "International Business Machines" to decide whether they belong to the same entity, people can answer this question easily if they are provided a corresponding context. Another situation is, given some images people can easily evaluate which images are better to represent a specific entity.

In order to reach the two design goals mentioned above, CrowdDB is interacted with AMT to implement crowdsourcing. Due to the participation of people, the whole database architecture needs to be changed.

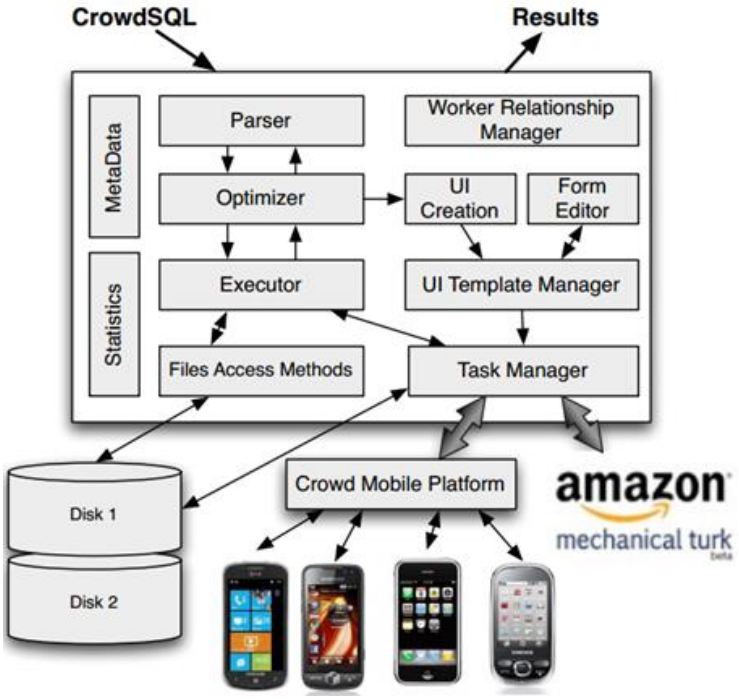

Figure 3-1: Architecture of CrowdDB[38]

3.1.1 Architecture of CrowdDB

Figure 3-1 is the architecture of CrowdDB. So far, the crowd resources for CrowdDB can be obtained by connecting CrowdDB with AMT or the Crowd Mobile Platform. AMT is already introduced above. The Crowd Mobile Platform is a locality-aware platform designed by the designers of CrowdDB. It allows constraining the worker to finite areas. Queries for CrowdDB are expressed by using CrowdSQL. CrowdSQL is a moderate extension of SQL as the query language in CrowdDB. As can be seen from Figure 3-1, the left above part of CrowdDB, which includes three components: Parser, Optimizer and Executor, is designed for the query processing. The query processing procedure is almost the same as the traditional query processing globally. If the related data of a query already exists in local tables, this query will be processed in the traditional way; Otherwise, queries have to invoke the crowd. The invoke of the crowd makes CrowdDB differ from the traditional relational database. The right part in Figure 3-1, which includes Worker Relationship Manager, UI (User Interface)Creation, Form Editor, UI Template Manager and Task Manager, realizes the invocation of the crowd by connecting with a crowd resource platform. UI Creation, Form

Editor and UI Template Manager together are called UI Manager. In the following, these three components are introduced briefly.

Worker Relationship Manager: The relationship between requesters and workers are two-fold. Not only requesters define the required qualification of workers, but also workers prefer to choose the tasks offered by good requesters. Good requesters mean two sides. On the one hand, tasks should be designed properly and easily to understand and perform by workers. On the other hand, the requesters should pay for the tasks appropriately. The relationship between requesters and workers is not static, but evolves over time. Hence, it is significant to manage the relationship between requesters and workers. In this way, CrowdDB could restrain the behaviors of requesters and workers and build an effective work community, in reverse, let workers better serve for requesters.

UI Management: User interfaces are essential for invoking the crowd resources, since crowds need the user interfaces and readable instructions in natural language to guide them how to work on a HIT. The three components in UI Manager: UI Creation, Form Editor and UI Template Manager are responsible for creating, managing and editing user interface templates. CrowdDB extends the DDL (Data Definition Language) of SQL to enable application developers to annotate tables with information that helps in user interface creation. At runtime, CrowdDB automatically creates its UIs based on the annotations and standard type definitions, constrains that appear in the schema. Programmers can also design their own UI forms to override the standard UI when needed.

Task Manager: This component manages the interactions between CrowdDB and the crowdsourcing platforms. It instantiates the UI templates and makes the API calls to post tasks, assess their status, and get the results of the assignments. In order to preload values into the tasks user interfaces or store the crowdsourcing results in database for the reuse, task manager also need to interact with the storage engines.

Since each invocation of crowds requires monetary cost and relatively long time, The results obtained from the crowd can be stored in the database for reuse.

3.1.2 Data Model

The data model of CrowdDB is very similar to the relational model except for some extensions of crowdsourcing elements. The extensions are as follows:

Crowdsourced Columns: There are two kinds of granularities which can be crowdsourced: crowdsourced columns and crowdsourced tables. Crowdsourced columns mean only one or several columns in a table should be crowdsourced, the table itself is like a normal table and the other columns are traditional columns in the traditional database with the same properties. This granularity is often used if only a part of columns need to be searched with the help of the crowd.

Crowdsourced Table: A crowdsourced table means the whole table should be crowdsourced and in CrowdDB a crowdsourced table must have a primary key to ensure that two workers do not input the same new tuple. A crowdsourced table is adopted if all the tuples in database cannot satisfy the users and need more data via crowdsourcing.

Aggregation Strategy: People are prone to make mistakes and often have their own subjective opinions, therefore CrowdDB cannot simply consider a response from one worker as the right result. Usually one HIT is assigned to several workers (usually 3 to 5 workers), consequently there will be several corresponding values for a crowd attribute of the same tuple or several corresponding tuples in a crowd table. In CrowdDB, these responses from different workers are not directly stored in the database. Instead, CrowdDB utilizes majority vote to aggregate responses and stores a unified result in database. As a result, the tables in CrowdDB are very similar with traditional tables.

3.1.3 Query Language

The query language in CrowdDB is called CrowdSQL, which is an extension to SQL. The extension mainly reflects on DDL (Data Definition Language) and DML (Data Manipulation Language) of SQL.

CrowdSQL DDL Extensions: CrowdDB marks Crowdsourced Columns and tables with a special key word CROWD. For example, a new table “department” is generated with a

consideration that the URL is often not provided but is likely available elsewhere, then "URL" is defined as crowdsourced column:

```
CREATE TABLE Department (
university STRING,
name STRING,
url CROWD STRING,
phone STRING,
PRIMARY KEY (university, name));
```

Another example for the crowdsourced table: a new table "professor" is created with a consideration that the professors stored in database is not complete, additional professors' information is expected by the users, then the "professor" table is defined as a crowdsourced table:

```
CREATE CROWD TABLE Professor (
name STRING PRIMARY KEY,
email STRING UNIQUE,
university STRING,
department STRING,
FOREIGN KEY (university, department)
REF Department (university, name) );
```

SQL types and integrity constrains are applied both for the regular tables and the crowdsourced tables. For instance, referential integrity constraints can be defined between two crowdsourced tables, two regular tables, and between a regular and a crowdsourced table in any direction. CrowdDB considers all the columns and tables as the same no matter whether a column or a table is crowdsourced except for one point: a crowdsourced table must have a primary key so that CrowdDB can infer if two workers input the same new tuple.

CrowdSQL DML Semantics:

- **INSERT statements:** CrowdDB introduces a new value CNULL, which means the data is not obtained so far but should be obtained via crowdsourcing when necessary. CNULL values are generated as a side-effect of INSERT statements. Once INSERT

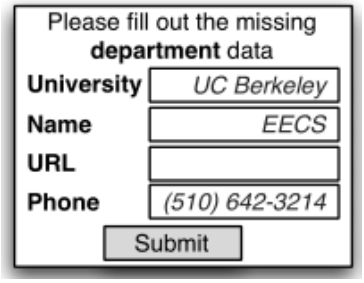

Figure 3-2: Basic Interface for incomplete data[/43/]

statements are executed to a regular table with crowdsourced column, if the value of crowdsourced column is not specified, CNULL as the default value of any crowdsourced column is initiated. Once INSERT statements are executed to a crowdsourced table, all the non-key attributes is initiated with CNULL if not specified. However, the primary key in crowdsourced table has to be specified and is never allowed to be CNULL.

- **DELETE and UPDATE statements:** not changed in CrowdDB.
- **SELECT statement:** If SELECT statement involves queries on the crowdsourced columns or tables, an automatical crowdsourcing procedure will be triggered, including user interface creation, HIT assignment, responses aggregation and the query result generation. A new LIMIT clause is suggested to constrain the number of tuples that are returned as a result of a query so that the time and cost for crowdsourcing in AMT are under a specific budget. Almost all above-mentioned extensions aim at achieving the first goal of CrowdDB: correctly answer the incomplete data queries. In order to achieve the subjective comparison, two new built-in functions are designed: CROWDEQUAL and CROWDORDER. CROWDEQUAL asks the crowd whether two entities are equal. In the CrowdSQL statement, CROWDEQUAL is represented by the symbol"~=". CROWDORDER asks the crowd to rank or order a set of entities. For example:

```
CREATE TABLE picture (
p IMAGE,
subject STRING);

SELECT p FROM picture
```

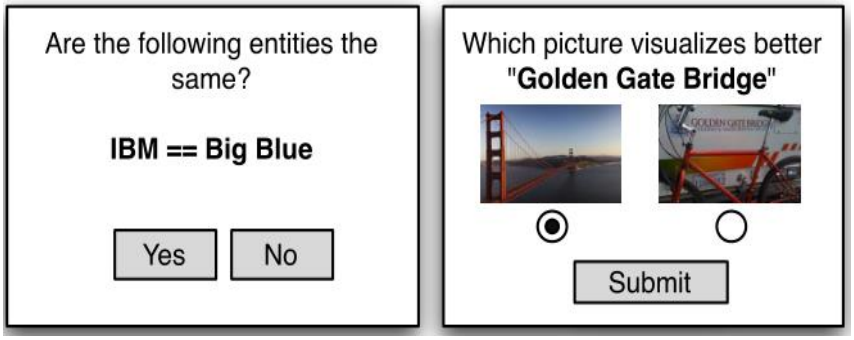

Figure 3-3: Basic UI for CROWDEQUAL and CROWDORDER[43]

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

3.1.4 User Interfaces

User Interface design cannot be ignored in crowdsourcing, it directly relates to the responses quality from workers. According to Figure 3-1, in CrowdDB user interfaces are generated in two steps. At compile-time, CrowdDB creates templates for all the crowdsourced tables and all regular tables with the crowdsourced columns according to the CROWD annotations in the schema, the templates are HTML and JavaScript forms; then these templates can be edited by requesters to provide additional custom instructions and are instantiated at run-time. Some UI examples in CrowdDB are depicted in Figure 3-2, Figure 3-3 and Figure 3-4. Figure 3-2 and Figure 3-3 are the basic interfaces with regard to one single table in CrowdDB. Figure 3-2 is the interface for crowdsourcing missing information. The title of the HTML is the name of the table and the instructions ask worker to fulfil the blanks. In run-time, the templates are instantiated by copying the existing values into the HTML form, the crowdsourced column with CNULL values are left with blanks and asking workers work on them. Sometimes JavaScript is also used to constrain the input contents of blanks. For instance, a select box is generated only allowing worker to choose a value from it for the corresponding attribute. Figure 3-3 is the basic interfaces for CROWDEQUAL and CROWDORDER. The introduction for CROWDEQUAL is a question to ask whether two entities are the same. Workers click “yes“ or “no“ buttons to answer it. The introduction for CROWDORDER is a

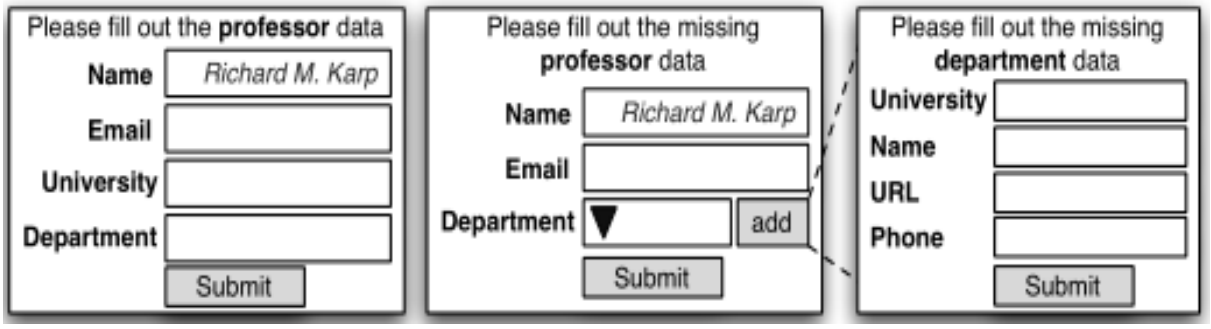

Figure 3-4: Multi-Relation Interfaces[/43/]

question to ask which image or description is better, workers select one image or description to perform the tasks. In order to improve the performance, CrowdDB also supports batching, i.e. one HIT consists of more than one microtask to be solved by workers. Figure 3-4 depicts some multi-relation interfaces, i.e. a crowdsourced table has a foreign key with other tables. If the referenced table is a regular table, then the interface can use a drop-down box or some other functions to list all the possible foreign key values. If the referenced table is also a crowdsourced table, two kinds of interfaces can be chose: the normalized interface (the left one in Figure 3-4) and the denormalized interface (the right one in Figure 3-4). A normalized interface only allows workers to add the foreign key value. A denormalized interface allows worker to add values of all the attributes in the tuple through clicking the "add" button to open a new window of the referenced table.

3.1.5 Query Processing

Query processing in CrowdDB follows the traditional query processing but with some small adjustments in parser and optimization steps.

Parsing and Translation: CrowdDB extends three additional operators called crowd operators. All three crowd operators are designed to instantiate some corresponding tuples to crowdsourcing, then collect and aggregate the responses from different worker. The Aggregation strategy, which has been mentioned above, is majority vote. Majority vote means choosing the answer from majority workers as the final answer. If no majority answer exists, more workers are asked until getting a majority answer. In the case of asking workers to input new tuples and almost each worker inputs their own tuples with different primary keys, the

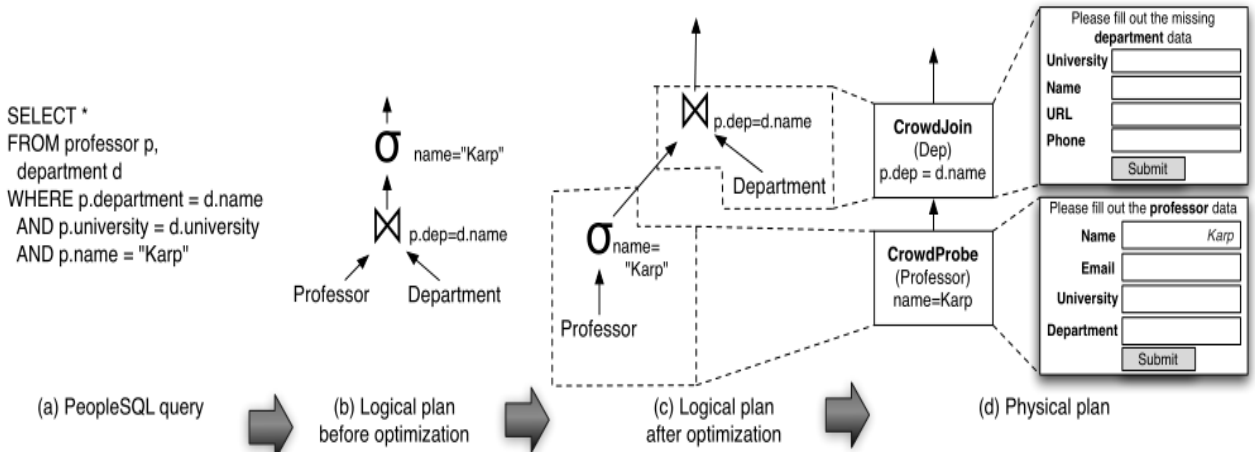

Figure 3-5: CrowdSQL Query Plan Generation[43]

operators will create new tasks with the known primary key obtained from workers and then let several workers input other needed information, at last apply majority vote to decide the final answer. Following their separate intention of operators are introduced:

- **CrowdProbe:** This operator is designed for incomplete data. It aims at obtaining the missing information for crowdsourced tables or columns via crowdsourcing.
- **CrowdJoin:** This operator implements an index nested-loop join over two tables and the inner relation must be a crowdsourced table. This operator creates one or more HITs in order to crowdsource new tuples from the inner relation that matches the tuple of the outer relation.
- **CrowdCompare:** This operator is designed for the two new built in functions CROWDEQUAL and CROWDORDER to compare data. It can be used inside another traditional operator, such as sorting or predicate the evaluation.

All the traditional operators are not changed in CrowdDB.

Optimization: The traditional optimization is cost-based optimization. For instance, some traditional well-known optimization technique: predicate push-down, stop after push-down, join-ordering and determining. CrowdDB absorbs this kind of optimizations and innovates a rule-based optimization due to the crowdsourcing issues. The rule-based optimization is called heuristic. First, heuristic predicts some cardinality of query plans with different operators, and then attempts to reorder the operators to minimize the numbers of HITs. Furthermore, CrowdDB sets a series of rules to balance the cost through choosing the batching size, payment per HIT, restricted time per HIT, proper user interface (e.g.,

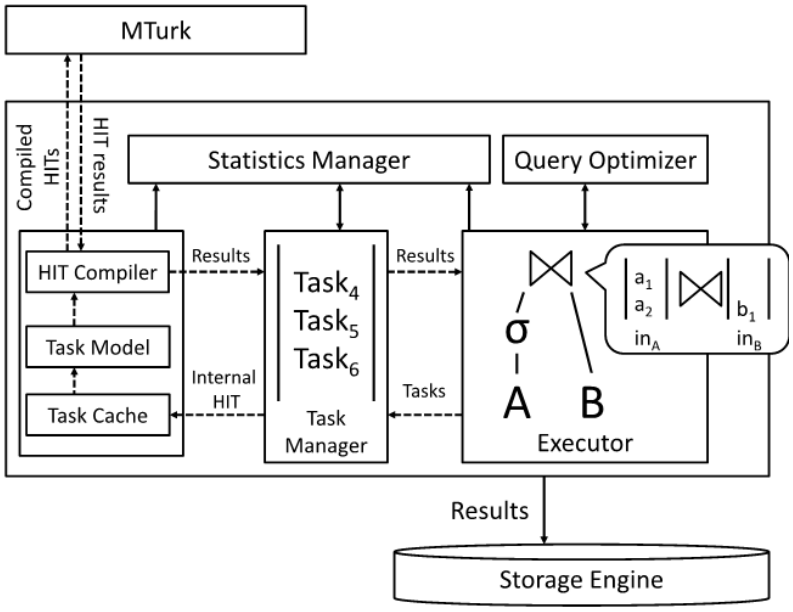

Figure 3-6: Architecture of Qurk[45]

normalized vs. denormalized) and other parameters. In contrast to cost-based optimization, rule-based optimization is very hard to exhaustively take all parameters into consideration and often cannot decide an optimal result.

Physical Plan Generation: Figure 3-5 is an example of the procedure to generate a physical query plan in CrowdDB. A query is first parsed to a logical plan, the logical plan is then optimized using traditional and crowd-specific optimization. Finally, the logical plan is translated into a physical plan which can be executed by the CrowdDB run-time system. Among this procedure, corresponding operators are instantiated. The query in Figure 3-5 is executed by a CrowdProbe operator and a CrowdJoin operator to obtain missing information via crowdsourcing. For brevity, only traditional optimization is showed.

3.2 Qurk[45] [46]

Qurk is designed by Adam Marcus, Eugene Wu, David R. Karger, Samuel Madden, and Robert C. Miller. They view all the HITs as operators which can be invoked as a part of query processing. In order to processing the queries which traditional database cannot process, Qurk is designed as a new database management system with a variety of modifications. Qurk's architecture is first described. So far, Qurk connects only with AMT to obtain the crowd

resources and the requester, which send queries to Qurk, have to pay according to the economic model of AMT.

3.2.1 Architecture of Qurk

Qurk's architecture is depicted in Figure 3-6. As can be seen from Figure 3-6, Query Optimizer in Qurk integrates the function of the traditional query parser and query optimizer. The query is first parsed into a query plan and then adaptively optimized during the query execution in the Qurk query optimizer. Qurk Executor takes query plans from Query Optimizer as input, executes the plan, and then outputs a set of tasks into Task Manager for workers to perform. Task Manager maintains a global queue of tasks and generates internal HIT representations into HIT Compiler according to the tasks one by one. The data in Statistics Manager is used to determine the number of HITs, HIT assignments, and the cost of each task. HIT Compiler transforms the internal HIT representations into HTML form and sends it to AMT, if Task Cache and Task Model cannot respond for the HIT. After workers fulfil the HIT, the results of HITs are sent back to Task Manager. With the purpose of re-use in the future, almost all the results are saved into the Storage Engine to improve the performance and reduce the cost. As an optimization, Qurk caches results in Task Cache. Task model lets Qurk possess the ability of machine learning, if a learning model of a HIT is aware, Qurk trains this model with the results and hopes to perform this kind of HIT automatically in the future. Since computer is good at the multi-tasks processing, it can approach hundreds of thousands of tuples at the same time, while human cannot and usually spends several minutes for a single operation on a tuple encoded in one HIT, Thus, in Qurk different components can operate asynchronously.

3.2.2 Data Model

The data model of Qurk is very similar to the relational model. The one key difference is: Qurk saves all the results in a list for a same HIT obtained from different workers, later the list of results can be reduced according to the aggregation strategies defined by users.

3.2.3 Query Language

The query language used in Qurk is a SQL-based query language with lightweight UDFs to instruct workers to perform HITs. UDF is short for user defined function. The form of UDFs is as follows:

```
TASK FunctionName (type    value) RETURN type of the response:
     TaskType: type of the task
     Text: "texts and images presented to workers'', [(URLify
     (value of the input) | value of the input)]
     Response:  form of the response
```

The italics in UDF will be instantiated for specific HITs. URLify() is a utility function, which converts a database object into a URL. It is used if an image or other media relates to a HIT. If no media relates, this part will be instantiated with the value of the input. The type of a task, the type, and the form of a response are corresponding with each other. For instance, if the type of a task is a "yes" or "no" question, then the type of its response should be "Bool" and the form should be "Choice('YES', 'NO') " , if the type of a task is "Rank", then the type of its response should be "String[]" and the form should be a list of subjects. The UDF templates are stored in Task Manager. Task Manager instantiates the templates according to different tasks and translates the tasks into internal HIT representation forms.
Next, consider an example of crowd join in Qurk. There are two tables: celebrities(name, image), spottedstars(id, image). The requesters want to know all pairs, which stands for the same person. The query should be expressed as follows:

```
SELECT celebrities.name, spottedstars.id
FROM celebrities, spottedstars
WHERE samePerson(celebrities.image, spottedstars.image)
```

The UDFs samePerson is defined as follows:

```
TASK samePerson(Image[] celebs, Image[] spotted)RETURNS BOOL:
```

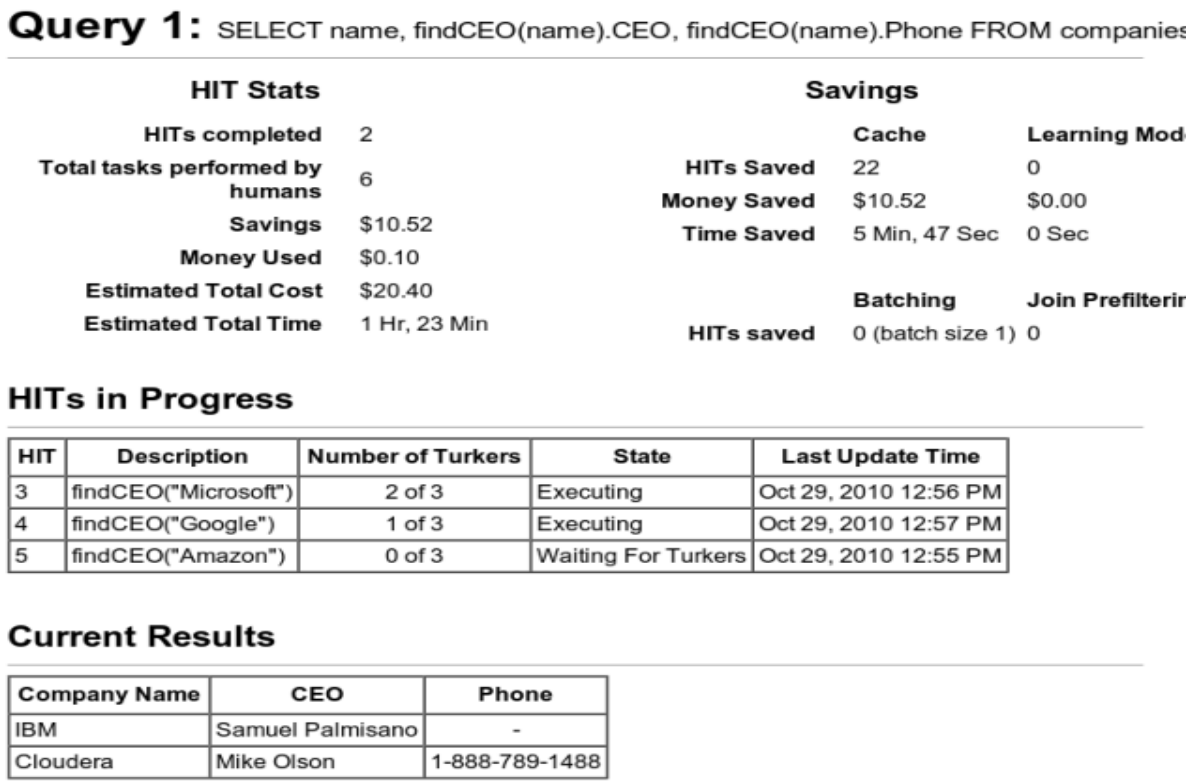

Figure 3-7: Query Status Dashboard in Qurk[45]

```
TaskType:JoinPredicate
Text: ''Drag a picture of any <b>Celebrity</b>in the left
column to their matching picture in the <b>Spotted
Star</b>column to the right.''
Response: JoinColumns("Celebrity", celebs, "Spotted Star",
spotted)
```

3.2.4 User Interfaces

Qurk provides a user interface for requesters called Query Status Dashboard to supervise the HITs, which have been already assigned to workers. Since the requesters have to pay according to the economic model of AMT. Qurk calculates the current budget and estimates for the total monetary cost and time cost, then displays the information on the dashboard. The dashboard also tells the requesters how much benefit can be gained from the query optimizations. Figure 3-7 is an example for the query status dashboard. Figure 3-8 is an example user interface for workers. It corresponds to the above mentioned crowd join example.

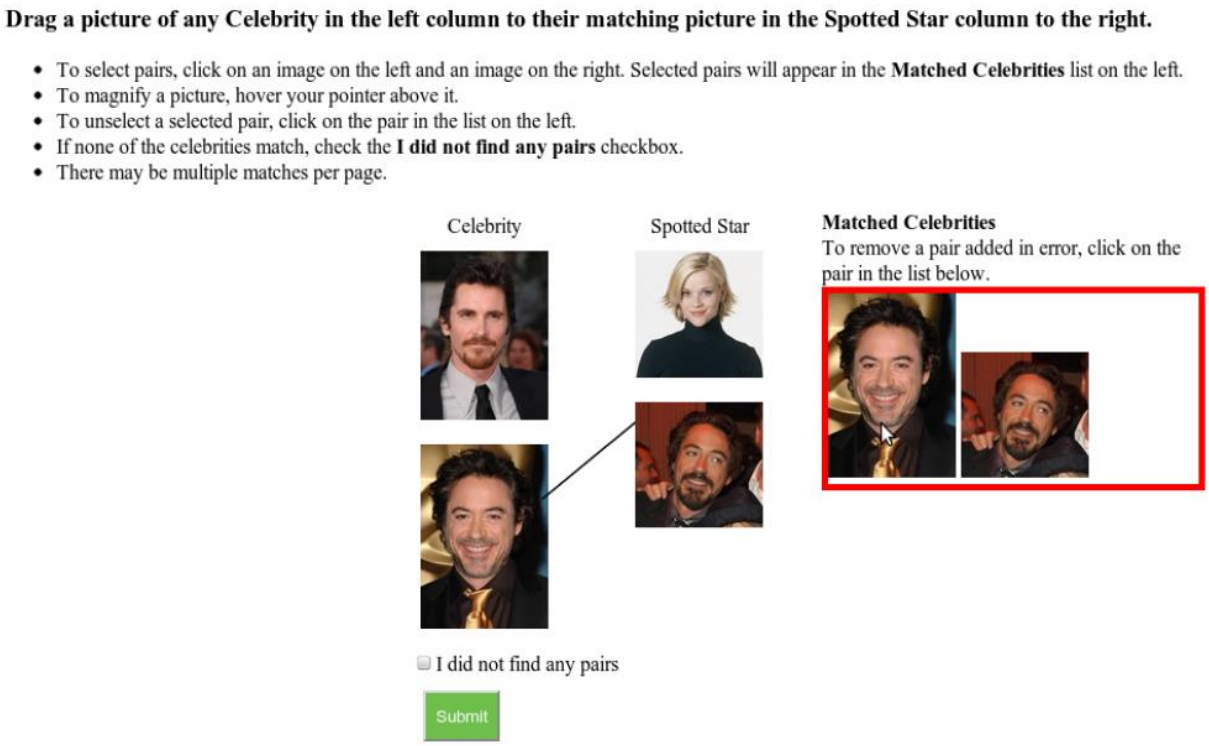

Figure 3-8: Join User Interface[45]

3.2.5 Query Processing

As mentioned in 3.2.1, Qurk integrates the steps of query parsing and optimizing in its query optimizer. In the step of query parsing and translation, instead of creating new operators, Qurk creates UDFs to implement publishing HITs on the AMT and workers can work on the HITs. For the step of query optimization, a Qurk query can be annotated with three parameters: maxCost, minConfidence, and maxLatency. MaxCost constrains the maximum amount of money a requester could pay. The cost of a query can be calculated as follows: multiplying the number of HITs by the number of workers to which a HIT need to be assigned, then multiplying the price of each HIT, the final result is equal to the cost of a query. If the cost exceeds the maxCost, the query cannot be executed. MinConfidence specifies the minimum number of workers provide a same response, then the response can be adopted. MaxLatency constrains the maximum time which a requester can wait for a HIT to be completed. The Qurk designers also raise some ideas for possible optimizations.

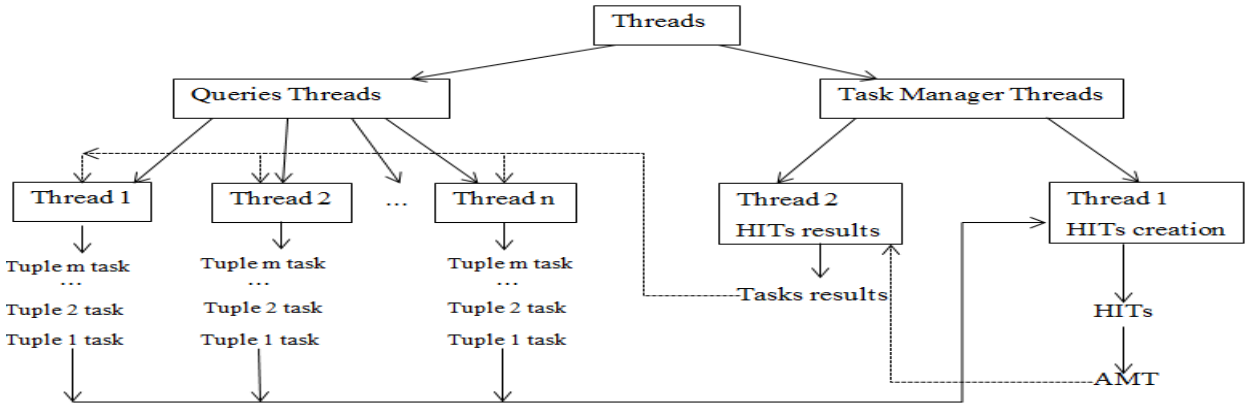

Figure 3-9: The procedure of Qurk query execution

Different Pay according to different HIT runtime: Workers should be paid more for some HITs which need more time to complete.

Input Sampling: Larger table leads to more HITs. Thus Qurk attempts to sample the extreme large table to generate HITs that uniformly cover the whole space.

Batch Predicates: If a filter query contains two predicates on the same table, Qurk can only build one HIT for the two predicates to save money and time.

Different implementations for a query: For example, "Rank" can be implemented through letting workers sort a list of subjects or letting workers score each subject in a list.

Join Prefilter: Before executing a JoinPredicate query, filtering the whole space with necessary conditions, then the comparison of the space will be reduced.

In the traditional relational database, a final query result can be obtained from the query execution engine. The Qurk executor cannot directly provide the final query result; instead it generates a set of tasks according to the query plan. In fact, Executor, Task Manager, HIT Compiler even with AMT together play the same role as the traditional query execution Engine. Qurk is multi-threaded. Each query has its own thread and maintains its own query tree. The thread of each query generates a set of tasks according to the tuples in the table. Then the tasks are enqueued into task manager. There is a specific thread in Task Manager to create a standard HIT through collapsing multiple tasks into one HIT and price HITs according to the instruction provided by the optimizer. There is another thread in Task Manager enqueueing the results of tasks into their corresponding query thread. At last when

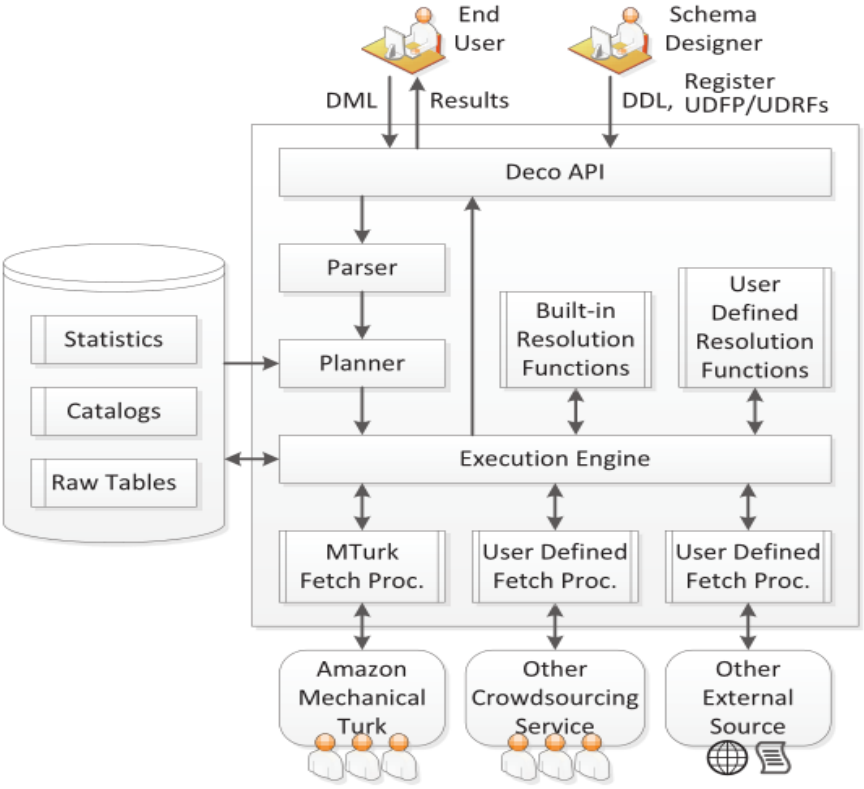

Figure 3-10: Architecture of Deco[/49/]

the root of the query tree contains the complete query result, then the results are stored in the database. Figure 3-9 illustrates this whole procedure.

3.3 Deco[/47/ /48/ /49/ /50/]

Deco is a database system that answers declarative queries posed over the stored relational data, the collective knowledge of the crowd, as well as other external data sources. Deco attempts to implement such a database with transparency, i.e. Deco designers would not like to show the complexities, which is caused by humans providing and processing data, aim at letting users feel the same as accessing a traditional database. The following sections will demonstrate Deco in detail.

3.3.1 Architecture of Deco

Figure 3-10 depicts the architecture of Deco. The Deco prototype is implemented in Python with a SQLite back-end. On the one hand, Deco supports the DDL commands called by schema designer to create tables, resolution functions, and fetch rules. Resolution functions

are the functions to cleanse the inconsistency and uncertainty obtained from AMT, other crowdsourcing service or external files. Fetch rules specify the way that data is obtained from humans. These new concepts in Deco will be expatiated later. On the other hand, Deco also allows end users to use DML to interact with the Deco API, which implements the standard Python Database API v2.0: connecting to a database, executing a query and fetching results. Additionally, Deco provides a command-line interface as well as a web-based graphical user interface (depicted in Figure 3-11). After the Deco API receives a query, in overall the query flow is the same as a flow in a traditional database: parsing, optimizing and executing, but the execution of the query is not only via the database itself but also via AMT or other crowdsourcing services or some other external sources.

3.3.2 Data Model

On the whole, Deco's data model is designed to be general, flexible, and principled. General means the data model can be instantiated to other models that adopted in other crowdsourced databases; Flexible means users can self-define the data cleansing and external access methods; Principled means the data model semantics are sound and precisely-defined. In this section, all the components of Deco's data model are first introduced, then the data model semantics are explained.

Components of the data model

- **Conceptual Relation:** Conceptual relations are the logical relations specified by the deco schema designer and queried by end-users and applications. There are two kinds of attributes in conceptual relations: anchor attributes and dependent attribute-groups. Anchor attributes are the attributes that can identify "entities", while dependent attribute-groups specify the properties of "entities". Formally, anchor attributes and dependent attribute-groups are denoted in a table as follows:

  ```
  table(anchor attribute 1,…, anchor attribute n, [dependent
  attribute-group 1],…, [dependent attribute-group m])
  ```

  Each dependent attribute-group is enclosed with square bracket and there are one or more dependent attributes in a dependent attribute-group, which are independent or dependent with each other. For example:

```
Country(countryname, [language], [capital])
```

All the examples used in this section are based on this example.

- **Resolution rules:** Because of the diversity of the responses from workers or other external files, there is much inconsistent or uncertain data in Deco, thus, Resolution rules are the rules used to cleanse the data in Deco. For each conceptual relation there should be one resolution rule for all the anchor attributes and one resolution rule for each dependent attribute-group specified by the schema designer. The resolution rule for all the anchor attributes should follow this form:

$$\emptyset \rightarrow \mathrm{A}: \mathrm{f}$$

While the resolution rule for a dependent attribute-group should follow this form.

$$\mathrm{A}' \rightarrow \mathrm{D}: \mathrm{f}$$

(A stands for all anchor attributes, A' stands for a subset of all the anchor attributes, D stands for a dependent attribute-group and f is a function.)

F in $\emptyset \rightarrow \mathrm{A}:\mathrm{f}$ means: the resolution function takes a set of tuples of values for all anchor attributes from the conceptual relation as input, cleans the anchor values and outputs the set of tuples of anchor values into a raw table, which will be introduced later. F in $A' \rightarrow D:f$ means the resolution function takes a tuple of a subset of anchor values and a set of tuples of values for the dependent attribute-group as input, cleans the set of dependent values and outputs the set of tuples into a raw table. There are several kinds of resolution functions in Deco:

avg (): Function avg () means calculating the average of the values as the final value. This function is usually used for the values with scores, such as rating.

dupElim (): This function is used to remove the duplicates among the values.

canonicalize(): This function puts the values in a particular form, and can perform "entity resolution" to merge differing pairs that are judged to refer to the same entity.

majority (): This function chooses the majority same value from all the values as the final value.

identity (): This function means directly outputting the input values without any processing.

The schema designers can choose a proper function for each resolution rule according to the specific situation. An example for the above mentioned conceptual relation: Country(countryname, [language], [capital]). The resolution rules may contain:

```
∅ → countryname : dupElim
countryname →language : majority-of-3
countryname → capital : majority-of-3
```

Resolution function *dupElim* generates distinct country values. Resolution function *majority-of-3* generates the majority of three or more languages for a given country.

- **Raw Schemas:** The traditional relational database is the back-end of Deco. The raw schema is the schema stored in the underlying relational database. It is invisible to both the schema designers and the end-users. The raw schema is generated according to the conceptual schema, anchor attributes, and the dependent attribute-group. It contains one anchor table whose attributes are the anchor attributes of the conceptual relation and many dependent tables. Each dependent table is a dependent attribute-group and the related attributes in the resolution rules for this dependent attribute-group. From the perspective of traditional relational database, the raw schema is in fact a Fourth Normal Form (4NF) decomposition of the conceptual schema based on the multivalued dependencies implied by the resolution rules. Users can insert, modify or delete data as they wish in conceptual schemas and these manipulations can easily applied for the raw schema. The raw schema according to the conceptual relation and resolution rules of above examples should be :

```
CountryA(countryname)
CountryD1(countryname, language)
CountryD2(countryname, capital)
```

- **Fetch rules:** Fetch rules can be specified by schema designers to describe the way that data can be obtained from the crowd or other external files. The number of fetch rules is unlimited, schema designers can specify any number of fetch rules. In addition, the fetch rules are unlike resolution rules, which cannot be changed after specified by schema designers. The fetch rules may be added, modified or removed at any time during the lifetime of a database. They are not the permanent schema but only like access methods for data. There is only one following form for fetch rules:

$$A_1 \Rightarrow A_2: P$$

A_1 and A_2 are sets of attributes from a same table, either of them can be ⌀, but cannot be ⌀ at the same time. P is the fetch procedure to access the data from the crowd or other external files. It takes as input a tuple of values for the attributes in A_1 and produces as output zero or more tuples of values for the attributes in A_2. For instance, if Deco is associated with AMT, then P might generate HITs assigned to AMT and collect values for the attributes in A_2. Fetch rules have only one restriction: If A_1 and A_2 contains dependent attributes, then $A_1 \cup A_2$ must contain all the related anchor attributes in the left side of the resolution rules for related dependent attributes. The reason for this restriction will be explained in the data model semantics. Following are the possible forms of the fetch rules:

Fetch rules for obtaining the anchor values: there are four ways to obtain the anchor values:

⌀
Anchor
Dependent
Anchor + Dependent ⇒Anchor: P

Anchor values can be directly obtained by no additional providing for the workers, or obtained by providing workers only other anchor values or dependent values. The latter way, only providing dependent values, is only fit for the situation that the anchor values expected to obtain include all the related anchor values of the resolution rules for dependent attributes. If this condition cannot be satisfied, not included anchor attributes values should be also provided to workers along with dependent attributes.

Fetch rules for obtaining the dependent values: Dependent values can only be obtained by providing workers all related anchor values in corresponding resolution rules.

Fetch rules for obtaining both anchor values and dependent values: The ways are the same as "Fetch rules for obtaining the anchor values", but with more conditions.

⌀
Anchor
Dependent
Anchor + Dependent ⇒Anchor + Dependent: P

The second way is feasible only if the anchor attributes from both left and right sides include all related anchor attributes of resolution rules for the dependent attributes. The third way is feasible only if the anchor attributes include all related anchor attributes of resolution rules for dependent attributes from both left and right sides. The last way is feasible only if the anchor attributes from both left and right sides include all related anchor attributes of resolution rules for dependent attributes from both left and right sides.

Fetch rules for verification: This kind of fetch rules do not intend to obtain data but to verify the reliability of fetch rules. The right side of verification fetch rules is ∅. The fetch rules return yes or no for the reliability of the given attributes. Note that the restriction is also needed in this case.

Possible Fetch rules for the example mentioned above are as follows:

- ∅ ⇒ countryname: Ask for a country name, inserting the obtained value into raw table CountryA.
- countryname ⇒ capital: Ask for a capital given a country name, inserting the resulting pair into CountryD2.
- countryname ⇒ language: Ask for the language used in a contry, inserting the country-language pair into CountryD1.
- language ⇒ countryname: Ask for a country name given a language, inserting the resulting country name into table CountryA, and inserting the country-language pair into CountryD1.
- Countryname, language ⇒ ∅: Ask the crowd whether the pair (countyname, language) is right.

Data model semantics

Deco defines a set of valid instances for the conceptual relations as the semantics of the Deco database at a given time. The valid instances for the conceptual relations can be showed by the following steps: Deco can obtain data from the crowd or other external files according to the fetch rules and store the data into the raw tables, then the raw tables with the newly

obtained data and the original data can be cleansed according to the resolutions rules. At last, all the raw tables are outerjoined into the conceptual relations. These conceptual relations are the valid instances at present. These steps are not really performed to get the valid instances, they are logical in order to define the semantics of the data model. In these steps, any fetch rules can be used several times, this leads to an infinite number of valid instances, but this infiniteness is not a problem, since Deco only delivers the result of a query over some valid instances, not all valid instances. These three steps can be summarized as Fetch-Resolve-Join sequence. Each element will be described in detail.

- **Fetch:** The current raw tables may be extended with the values obtained by invoking any number of fetch rules several times. According to the definition, procedure p takes as input a tuple of values for the attributes in A_1 and outputs zero or more tuples of values for the attributes in A_2. Then, assuming a set of tuples for $A_1 \cup A_2$, if the intersection of these tuples with the attributes in a raw table is nonempty, insert the tuples of the intersection into the raw table and assign a NULL to the attributes in the raw table but not in $A_1 \cup A_2$. Therefore, the restriction for the fetch rules ensures that no anchor attributes of the dependent raw tables have NULL values.
- **Resolve:** After extending the raw tables via fetch rules, the raw tables should be resolved next via resolution rules logically. The resolution rules for anchor attributes can be simply applied to all tuples in anchor raw tables. For dependent attributes, it is a little more complex: the related anchor attributes are first grouped, then the resolution functions are invoked for each group and at last, the outputs of the resolution functions replace the original groups.
- **Join:** At last, conceptual relations are emerged logically through left outerjoining of the corresponding resolved anchor table and dependent tables. The left outerjoin must take the anchor table as its first operand, but the orders of other dependent tables are arbitrary.

3.3.3 Query Language

A Deco query is a relational query over the conceptual relation. Deco defines the query language semantics as the relational answer to a query over some valid instances of the

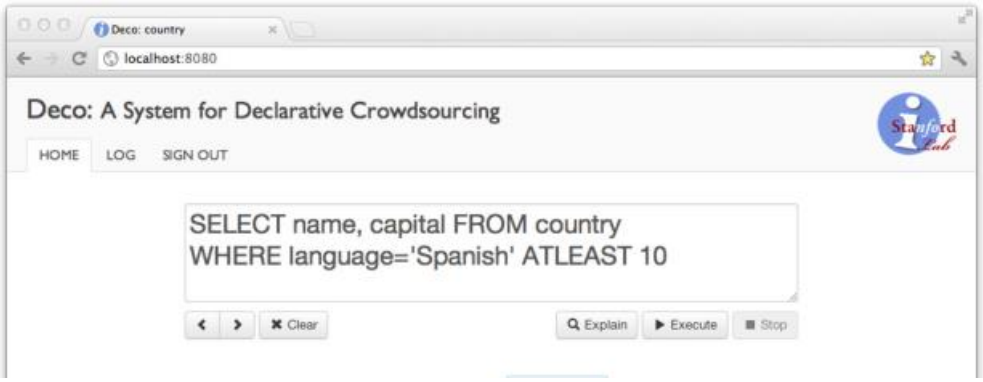

Figure 3-11: Deco web interface[48]

database. As a start, some examples for DDL are demonstrated, then several adjustments for crowdsourcing are depicted.

DDL in Deco:

- **Create Resolution Functions:** Before tables are created, the resolution rules should be first created. Following is an example for function majority.

```
CREATE FUNCTION majority AS
'def majority(tuples, n):
     for t in tuples:
          if tuples.count(t) > 0.5 * max(n, len(tuples)):
          return [t]
     if len(tuples) >= n: return []'
```

- **Create Table:** Once the resolution functions are created, we can create a table; this table means the conceptual relation.

```
CREATE TABLE country (
countryname varchar USING dupElim(1,1),
[language varchar USING dupElim(4,2)],
[capital varchar USING majority(3)])
```

- **Create Fetch Rules:** Fetch rules can be created as follows:

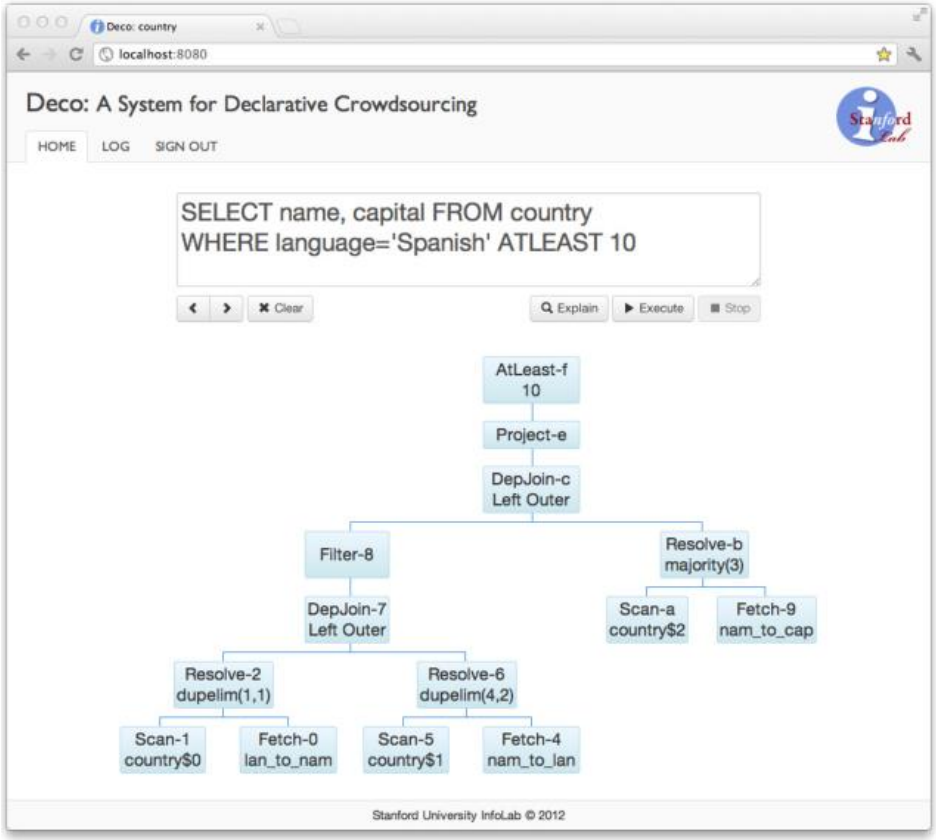

Figure 3-12: Query Plan Visualization[48]

```
CREATE FETCHRULE name to capital ON country
(name TO capital) USING mturk WITH '{"reward": 0.05,
"question": "What is the capital city of ${name}?"}'
```

- **Change Resolution Rules:** Sometimes the original resolution rules are found not feasible. Thus the resolution rules should be changed in following forms:

```
ALTER TABLE country
ALTER COLUMN capital USING majority(5)
```

DML in Deco:

The valid instance of the database can be showed by resolving and outerjoining the current raw tables without invoking any fetch rules. Then a Deco query can be processed directly by Deco without the help of the crowd or other external files, but there are situations that a query cannot get any result only a NULL value according to valid instances. For example,

Figure 3-13: Query Execution[48]

```
SELECT name, address FROM Restaurant
WHERE cuisine='Thai' and rating > 4
```

But at present there are no Thai restaurant's rating is better than 4. To avoid directly returning an empty result, an "AtLeast n" clause is added to specify the least number of query results. If there are fewer results in the current Deco than n, the fetch rules are enforced to obtain data from the crowd and other external files. Other clauses to constrain the performance of Deco are also suggested: MaxTime, MaxBudget, MayFetches specify the maximum time, budget or fetches for a query separately.

3.3.4 User Interfaces

In this section, interfaces for requesters and workers provided by Deco are introduced.

Interfaces for requesters: Deco provides a web interface (Figure 3-11) for requesters to build queries(). After requesters finish the building of the query, by clicking the "Explain" button a query plan can be visualized (Figure 3-12) and then clicking the "Execute" button to execute this query. Assuming that there is not enough data in raw tables for the query, the query has to be transformed to some HITs, which will be published in AMT. After HITs are published in the AMT, the up-to-date results of the HITs can be displayed in the Deco web interface (Figure 3-13). Figure 3-13 shows the current result for the query, which asks for capitals of ten countries, which language is Spanish. Nine capitals haven been already returned from workers. Only the last NULL capital for Peru is still needed to be done.

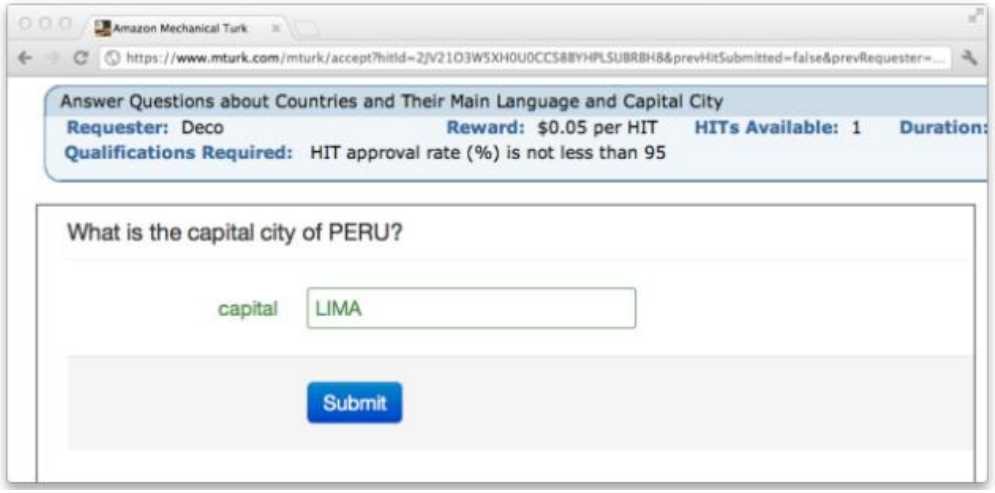

Figure 3-14: AMT worker interface[48]

Interface for workers:

The above Figure 3-14 is the Deco's interface for workers who are answering the capital question.

3.3.5 Query Processing

Analyzing the structure of the whole query processing in Deco seems to be very similar with the traditional query processing: parsing and translation, optimization, and execution. But actually there are some non-negligible changes in Deco to adapt Deco's data model, query language ,and crowdsourcing challenges. Following are the challenges that Deco faces and its corresponding coping strategy.

Monetary Cost vs. Two Phases: Query processing via crowdsourcing requires monetary cost. Hence, Deco is designed to process queries in two phases: materialization phase and accretion phase. The materialization phase checks whether there is sufficient data in raw tables and answers queries according to the existing data if the raw table is not empty, if data in raw tables is not sufficient, then the second phase is performed. The second phase, accretion, invokes fetch rules to obtain more data.

High Latency vs. Asynchronous Pull: Since each HIT published in AMT cannot always be assigned immediately to workers and workers also need time to finish the HIT, the latency to get a query result with the traditional database is very high. The traditional iterator model is

not feasible in the situation with high latency, because getNext calls do not return until data is provided. In Deco, in order to reduce the latency, parallelism is used. Deco realizes the parallelism through asynchronous pull. Query operators do not need to wait for a response after sending a "pull" request asking for the query results. Instead they can pull specific times without any response. The number of the asynchronous pull should be moderate. Too many times pull will lead to work waste and unnecessary monetary cost. This built-in asynchrony makes it possible to allow Deco to ask multiple HITs to the crowd at the same time.

Changing Result Tuples vs. Incremental Push: According to the procedure of crowdsourcing, the result tuples change if there are more results added into raw tables. However, the traditional iterator model does not allow users to modify values in tuples once they are passed up the plan. Therefore, the Deco designers borrow idea from incremental view maintenance to let the output always reflect the current state of the raw tables. This method is called incremental push in Deco. It handles each result of fetch rules as an update to the raw tables and then propagates to the conceptual relation. Push in Deco means respond a new output tuple to the raw table. The incremental push with the asynchronous pull together constitutes a novel push-pull hybrid execution Model for Deco.

After introducing the typical features of the Deco execution model, some extensions for these features in query processing steps will be described next.

Parsing and Translation: Although SQL-like queries are posed over conceptual relations, the query parser and translator translate the queries into execution plans over the raw schema, and the query executor is not aware of conceptual relations at all. There are a variety of new operators created in Deco, such as MinTuples, DLOJoin, Resolve, Scan, Fetch and so on. These operators will be explained in the following procedure to build a query plan.

Because of the diversity of the fetch rules, there may be also different query plans built based on a SQL-like query. The procedure to build a query plan is also depicted with the above mentioned example; a possible query is as follows:

```
SELECT countryname, capital FROM Country
WHERE language='Spanish' AtLeast 8
```

A basic query plan (Figure 3-15) is built according to three fetch rules: $\varnothing \Rightarrow$ countryname (operator 8 in Figure 3-15), countryname $\Rightarrow$ language (operator 11 in Figure 3-15), and countryname $\Rightarrow$ capital (operator 14 in Figure 3-15). Some abbreviations are used in it but are easily understood. The new operators used in the plan are as follows.

- **Fetch and Scan:** The *fetch* operator corresponds to the fetch rules $A_1 \Rightarrow A_2$: P. The *fetch* operator invokes procedure P to get the data for A_1 based on the data in A_2 from its parent operator. It does not wait for the response, so many procedures can be invoked in parallel. After procedure P return the data for $A_1 \cup A_2$, *scan* operator receives the data, passes them up to its parent operator. Then the data is inserted into raw tables.
- **Resolve:** The *resolve* operator corresponds to the resolution rules $A_1 \rightarrow A_2$: f. After the *scan* operator sends the $A_1 \cup A_2$ data into raw tables, the *resolve* operator applies the resolve function on the tuples with the same A_1 value and passes up the resolved data for $A_1 \cup A_2$.
- **DLOJoin:** The *DLOJoin* operator is very similar to the traditional index nested-loop join. The *DLOJoin* operator takes always the anchor attribute raw table as its outer table and dependent attribute raw tables as its inner table. At last it returns the join result tuples.
- **MinTuples:** The *MinTuples* operator according to the clause 'AtLeast n' determines when to terminate the query processing and return the final result into conceptual relations.

For brevity, the query plan in Figure 3-15 is built based on the easy case that no data exists in the raw tables. First the root operator *MinTuples* sends eight *getNext* requests to its child operators. The operators except for the *fetch* operator ignore the request, then a *fetch* operator in the leaves accepts the requests and processes with the fetch rule $\varnothing \Rightarrow$ countryname in parallel. After all the fetches finish, the *scan* operator passes the new data up and resolved by the resolve operator. Through the DLOJoin operator, another fetch rule *countryname* $\Rightarrow$ *language* is triggered to obtain data. Since the resolve function is majority 3, the fact that at least two workers return a same language for a country can make the data efficient. At the same time more fetches on countries and language for a given country are invoked in parallel, after all needed data is returned by fetches, the *scan* operator passes the new data up and the resolve operator resolves the data. Then the DLOJoin operator performs the nested loop join.

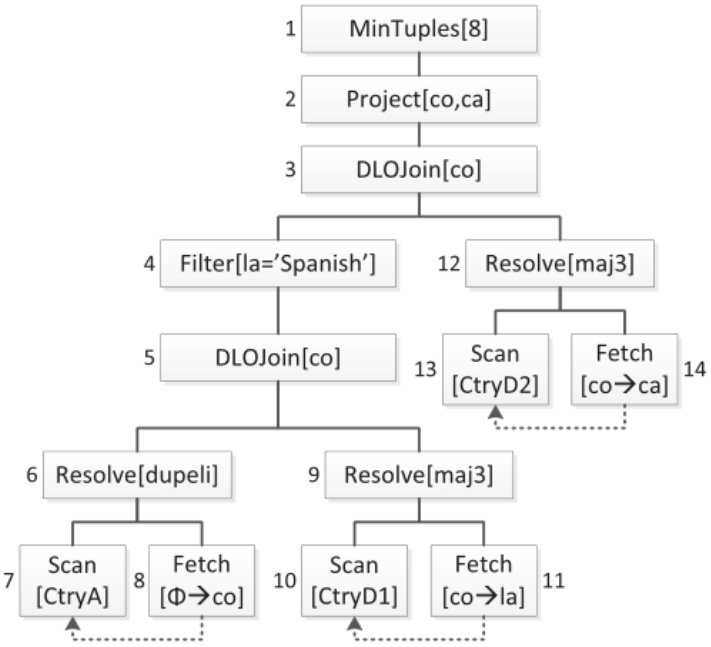

Figure 3-15: A basic query plan in Deco[48]

The join result is then filtered by [language = 'Spanish'], if there are less than eight tuples, more fetches are performed. Next, the higher-level DLOJoin operator triggers the fetch rule *countryname ⇒ capital.* All operators perform the same as described above. At last, the final result with 8 countries' names and their capital names are returned and stored in the conceptual relation.

Optimization: Deco's query optimization is cost-based optimization. But this cost means not the same as the cost in the traditional query optimization. This cost means monetary cost on AMT according to different fetch rules or different join orders. For the example above, there are still other query plans. The query plan depicted in Figure 3-16 is built based on the fetch rules *language ⇒ countryname* (operator 8 in Figure 3-16), *countryname ⇒ language* (operator 11 in Figure 3-16), *and countryname ⇒ capital* (operator 14 in Figure 3-16). Deco's query optimization algorithm takes a query plan and statistics as input, outputs the estimated cost in dollars. Deco is able to count certain information on AMT, for example which fetch rules perform better than others, and saves these statistics for the future query optimization. Assume that each fetch costs 0.01$ and the selectivity factors of predicate language='Spanish' and resolution function majority-of-3 are 0.1 and 0.4, then the cost for the basic plan is: Fetch operator 8 needs 8/0.1= 80 fetches. Fetch operator 11 need 8/0.1/0.4= 200 fetches. Fetch operator 8 need 8/0.4= 20 fetches, then the final cost is 0.01*(80+200+20) = 3$. The cost for the second plan is calculated in the same way, but only costs 0.01*(8+20+20) =0.48$, thus the

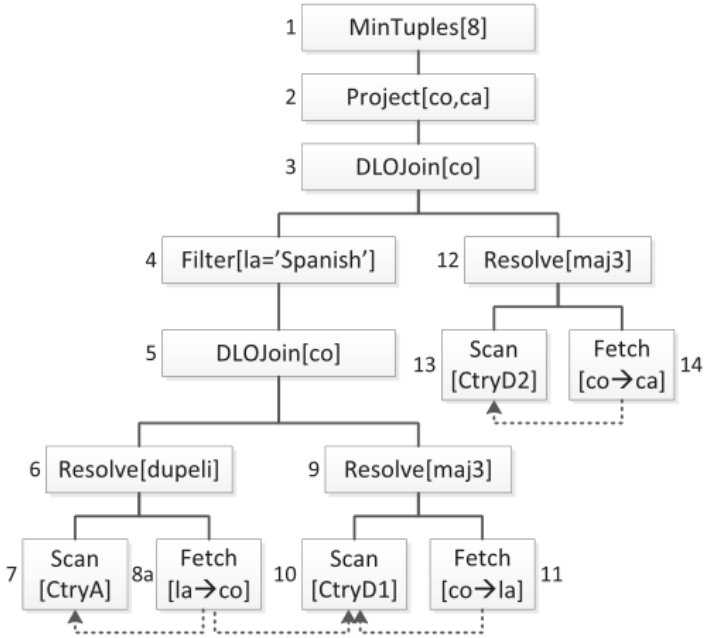

Figure 3-16: Alternative query plan[/48/]

second query plan is chosen to execute the query. In general, Deco enumerates all query plans according to different fetch rules and join order, then evaluate all query plans to get their monetary cost, choose the cheapest one as the final query plan.

Execution: At the beginning of this section, the execution model is already introduced. This part addresses the procedure how HITs are processed in AMT. Once a *fetch* operator invokes a AMT fetch procedure, a form template is instantiated according to the fetch rule. Then the HTML form is sent to AMT and the AMT API is invoked to create a HIT that points to the form server. When a worker accepts any of the HITs, the worker's web browser loads the address of the form server, then the HTML form is sent to the worker.

3.4 Comparison and Conclusion

In this section, the three previously discussed crowdsourced database system CrowdDB, Qurk, Deco and the traditional relational database system are compared based on their architectures, data models, query language, user interfaces design, and query processing steps. At last, a conclusion is made.

dimension \ Databases		CrowdDB	Deco	Qurk
Goal		extend traditinal databases to be able to correctly answer two kinds of queries: incomplete data and	extend traditional databases to be able to answer several queries, more general than CrowdDB	based on traditional databases to solely implement a variety of crowdsourcing tasks, which are viewed as database
Architecture	overall architecture	adapt to a traditional data processing flow	adapt to a traditional data processing flow	cannot adapt to a traditional data processing flow
	Partners	AMT and their own crowdsourcing platform	AMT, other crowdsourcing services	AMT
	human factors management	worker relationship manager	no	no
	UI	UI creatin, form editor and UI template manager	integrated in fetch procedures	integrated in HIT compiler
	storage for future reuse	storage engine	raw tables	storage engine and task cache
	task managemen	task manager	fetch procedures	task manager and HIT
	others	no	no	task model

Table 3-1: Comparison based on the architecture and goal

3.4.1 Goals and Architecture

Goals: Both, CrowdDB and Deco, are designed to realize more data processing besides the traditional database manipulation. CrowdDB is designed only to correctly answer the queries with incomplete data and subjective comparison. While Deco is more general, aiming at fulfilling any task defined by users. In contrast to CrowdDB and Deco, Qurk is designed in specialty to perform a variety of tasks via crowdsourcing based on the traditional relational database model.

Architecture: The different terms in the three architectures caused by different data models will be compared in the data model section. The Architecture comparisons among three databases are based on the overall, their partners, and crowdsourcing management aspects.

- **Overall:** Both, the backend of CrowdDB and Deco, are traditional relational databases. They can work as normal traditional databases. While from the architecture of Qurk, we can see there is no traditional database processing flow, it is solely designed to processing tasks with human operators.

Databases / dimensions	Traditional relational databases	CrowdDB	Deco	Qurk
components	traditional tables with traditional columns	traditional tables with traditional columns, traditional tables with crowdsourced column, crowdsourced tables	raw tables, conceptual relations with anchor attributes and dependent attribute-groups, fetch rules and resolution rules	traditional tables with traditional columns
integrity constraints	Domain constraints, Key constraints, Not Null constraints, Referential integrity constraints, Semantic integrity constraints	all constraints of the traditional relational database && must have a primary key for a crowdsourced table	no constraints on the raw tables, all constraints of the traditional relational database on the conceptual relations	all constraints of the traditional relational database except for allowing multi-values for an attribute
aggregation strategy	x	majority voting	resolution rules(majority voting, remove dumplates, average,entity resolution	retain multi-values in the database or defined by users
granularity to be crowdsourced	x	a column or a whole table cannot be changed by end users	any tables or any column with proper fetch rules, fetch rules can be changed easily by end users	any level of granularities(any table, any column)
Flexibilty to face the change of aggregation function	x	have to crowdsource again and then apply the new aggregation function	no need to crowdsource again, only apply the new resolution rules to the data stored in the raw tables	if multi-values still exist in the database, no need to crowdsourced again. Otherwise have to crowdsource again

Table 3-2: Comparisons based on the data model

- **Partners:** At the beginning of the design of CrowdDB, it only supports AMT as its crowdsourcing platform. So far, CrowdDB can also be associated with their own mobile platform. The mobile platform lets the crowdsourcing service even more conveniently realize with cellphones almost anytime and anywhere. Qurk is designed only associated with AMT to achieve the crowdsourcing task. Deco intends originally to associate with different crowdsourcing marketplaces, services or any external files. However, Deco's query model is described mainly based on AMT, the interaction with other files or services is not instantiated, only a good thought.
- **Crowdsourcing Management Aspects:** In CrowdDB, human factors are emphasized and the worker relationship manager is designed to manage the relationship between workers and requesters. In the long run, a proper management can lead to a better performance and accuracy, thus this is necessary to have such a manager; while there is no worker relationship manager in Deco and Qurk. UI component is explicitly showed as a separate part in the architecture of CrowdDB, but implicitly integrated in

dimensions \ Databases		Traditional relational databases	CrowdDB	Deco	Qurk
overall		SQL	SQL-like	SQL-like	SQL-based with lightweight UDFs
DDL		create table	new CROWD key word for crowdsourced tables and columns	first create resolution functions, then create tables with square brackets for dependent attribute-groups && Create fetch rules	create tables && additional UDFs descriptions
		alter table, delete table	not changed	additional alter resolution rules and fetch rules&& only raw tables can be deleted by schema designers, conceptual relations cannot be deleted by end-users.	not changed
DML	INSERT	NULL value as the default value	CNULL values as the default value	NULL value as the default value	NULL value as the default value
	SELECT	basic clauses	additional built-in functions: CROWDEQUAL(~=) and CROWDORDER && LIMIT clause	AtLeast clause, MaxTime clause, MaxBudget clause, MaxFetches clause	UDF names embedded in SQL clauses

Table 3-3: Comparisons based on the query language

Qurk's HIT compiler and Deco's fetch procedure. In CrowdDB, the task manager is responsible for the interaction with the crowdsourcing platforms. In Qurk, the task manager and the *HIT* compiler work together to generate and manage the HITs. In Deco, the fetch procedures are used to interact with the crowdsourcing platforms or some external files. All three databases store certain crowdsourcing results for future reuse; CrowdDB and Qurk store the results in the storage engine, particularly, Qurk still has a task cache to cache certain results temporarily, which may be used again in a short time. Deco stores the results directly in the raw tables due to its special data model. Besides, Qurk has the task model, which uses a kind of machine learning technology for the sake of the highest degree of automation. The comparisons in this section is summarized in Table 3-1.

3.4.2 Data Model

In the traditional relational database data is organized in tables (relations) of records (tuples) with columns (attributes). A table can have a primary key, which can be referenced from another table as a foreign key. CrowdDB designs special crowdsourced tables and

crowdsourced columns to mark that these tables or only these columns should trigger the crowdsourcing services, particularly the crowdsourced table is forced to have a primary key. The data for an attribute obtained from crowdsourcing is first cleansed into a single value by majority voting, then stored in the database. The data model in Qurk is very similar to the relational model, since Qurk is designed to process only crowdsourced queries, i.e. all of the queries need the help of the crowd, no special tables or columns are created to trigger the crowdsourcing services. There is only one difference with the relational model: Qurk allows to store all the different values for the same attribute obtained from the crowd into the database, then values for an attribute may be multi-valued. These multiple values can be aggregated by the different strategies defined by users. Deco is designed to be general, flexible and principled, thus it has a most complex data model. There are two kinds of tables: conceptual tables and raw tables. The raw tables are the physical tables really stored in the database, there are no integrity constrains on it, all the data obtained from the crowd or external files according to the fetch rules can be stored in the raw tables. The raw tables can be integrated into conceptual relations according to the resolution rules. The conceptual relation is a logical table which is as the same as the table in the traditional database. The attributes in the conceptual relation are classified into anchor attributes and dependent attribute-groups. The comparisons based on the data model are showed in Table 3-2.

3.4.3 Query Language

Overall, the query languages in three crowdsourced databases are based on SQL. CrowdDB adds a new key word “CROWD” to indicate that a query needs to be crowdsourced. Qurk embeds UDF names in SQL clauses to trigger the crowdsourcing services. Deco has the most complex data model. Therefore, the DDL in Deco is extended not only for tables but also for resolution rules and fetch rules. Deco adds “AtLeast” clause to restrict the least number of records that should be returned. If the number of satisfied records is not enough in local, crowdsourcing services will be triggered. Besides the aspects mentioned above, there are some other characteristics in three crowdsourced databases. The specific comparisons are depicted in Table 3-3.

dimension \ Databases	Traditional Relational Databases	CrowdDB	Deco	Qurk
operators	traditional operators	traditional operators &&new operators: CrowdProbe,CrowdJoin, CrowdComopare	traditional operators && new operators: fetch, scan,DLOJoin,resolve, MinTuples,filter,project	UDFs for filter, incomlete data, ranking and join
query optimization	cost-based optimization	traditional optimization && rule-based optimization (heuristic)	list a variety of ideas to indicate the factors can be optimized.	traditional optimization for the materialization phase && monetary cost-based optimization for accretion phase.
query execution	a query plan->a query result; a literation execution model	1. traditional queries are processed in the traditional way; 2. crowdsourced queries: a logical plan after optimization ->a physical plan(HITs on AMT) ->HITs results->majority voting->a query result	a query plan->a set of tasks->a set of HITs->published on AMT->HITs results-> tasks results->a query result; different components operate asynochronously	1.queries only need materialization phase processed in the traditional way; 2. queries only need accretion phase: a query plan->HITs->HITs results ->raw tables->Resolution Rule->a query result; 3.queries need two phases:1 + 2; hybrid push-pull execution model;

Table 3-4: Comparisons based on the query processing

3.4.4 User Interfaces

UI for workers: Communication is important among people, thus the user interfaces for workers as the intermediary of the communication between the requesters and the workers are non-negligent. In the references of CrowdDB, basic UIs and multi-relation UIs for all the queries are separately depicted. While in the references of Qurk and Deco, only one example UI is depicted. However, we can also see that all the three databases intend to make the UIs for workers easy to understand and humanized, which is the basic criterions for UIs for workers.

Other UIs: In Qurk and Deco, there are other UIs for the requesters introduced. Qurk provides the query status dashboard to let the requesters supervise the HITs, which have been already assigned to workers. Deco provides a web UI for the requesters to build a query, visualize a query plan, and check the status of the current HITs.

3.4.5 Query Processing

All of the three crowdsourced databases include the three traditional steps for the query processing: parsing and translation, optimization and execution. The comparisons based on the three steps are depicted in Table 3-4.

3.4.6 Conclusion

In general, CrowdDB and Deco aim at building a better database to be able to fulfil more data processing, while Qurk recurs to the traditional database to build a complete workflow for the crowdsourcing data processing. CrowdDB is able to answer incomplete data queries and subjective comparison queries through some simple extensions based on the traditional database. While Deco is intended to be general, flexible and principled, hence it has a more complex data model and a more complex query processing procedure. On one hand, each crowdsourced database has their own nice elements: such as worker relationship manager in CrowdDB, query status dashboard in Qurk, and the web interface to visualize the query plan in Deco. These elements can be involved in all databases easily and play a role for all the databases. On the other hand, all crowdsourced databases are still facing a series of challenges brought by crowdsourcing, these challenges will be described in chapter 5.

4. Crowd-based Data Processing

In this chapter, a variety of crowd-based data processing is discussed. Since this thesis focuses on the research between crowdsourcing and databases, the extensional database queries, which are processed via crowdsourcing, are demonstrated in detail. Other crowd-based data processing such as data integration, information extraction, and information retrieval is briefly presented.

4.1 Crowdsourced Database Queries

In this section, the main current research on database queries is demonstrated. Some of them briefly implement the specific database queries in crowdsourcing environments based on the crowdsourced databases introduced in chapter 3. Some of them focus on strategies to design proper questions or aggregate worker responses to the specific database queries. Others focus on designing algorithms to solve database queries based on different metrics.

4.1.1 Filter

Filter is a very common kind of query via crowdsourcing. It asks the crowd to determine whether the items from a given item set satisfy a given set of properties. Each property is called a filter. For instance, given a set of pictures, a filter can be "picture shows a cat." In crowdsourcing, each item or multiple items in the given item set generate one question, which is assigned to the preset number of workers. In chapter 3.2, query processing based on Qurk has been introduced. Filter query processing based on Qurk is addressed in **/45/.** A filter query is expressed by the SQL-based query language in Qurk as follows:

```
SELECT *
```

```
FROM table
WHERE FunctionName(type of the input   value of the input)
```

The UDF (User Defined Function) may be like this:

```
TASK FunctionName(type value) RETURN Bool:
     TaskType: Question
     Text:"Does...:(<src='%s'>)contain...?"|"Is...:(<src='%s'>)
     ...?",URLify(value of the input)|value of the input
     Response: Choice("YES", "NO")
```

"Question" in the "TaskType" field means: workers are asked to answer a question. The "Response" field specifies that workers will be given a choice, such as a radio button with two possible values "YES""NO". The "Text" field shows the question which will be supplied to workers. Once the filter query is asked, a HIT (Human Intelligent Task) in the HTML form is created on AMT (Amazon Mechanical Turk) based on the filter UDF, then the HIT can be performed by different workers.

Next, two approaches to implement the crowdsourced filter queries in /51/ are depicted and an example is used for ease of understanding. Consider a table (id, name, picture), suppose the filter is to identify the photos of males. This research in /47/ also relies on the crowdsourced database Qurk. The corresponding SQL-based filter query is depicted as:

```
SELECT id, name
FROM photos
WHERE gender(picture) = 'male'
```

The first form of a UDF gender() may be like this:

```
TASK gender(field) TYPE Generative:
```

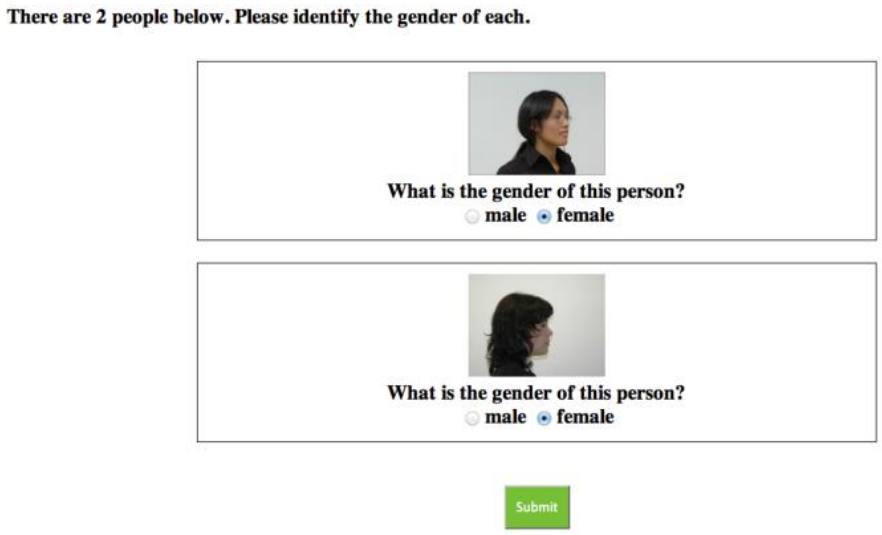

Figure 4-1: Label-based interface[/47/]

```
ItemPrompt: "<table><tr><td><img src='%s'><td>What is the
gender of this person? </table>", tuple[field]
Response: Choice("Gender", ["male","female"])
BatchPrompt: "There are %d people below. Please identify
the gender of each.", BATCHSIZE
Combiner: MajorityVote
```

The *gender* UDF instances the filter templates with the following changes: Instead of asking a "yes" or "no" question to workers, The *gender* UDF asks workers Wh-Questions: *What is the gender of this person*. In order to improve the performance, the batch optimization is utilized in it. The *BATCHSIZE* specifies the number of questions in one HIT, for instance, the *BATCHSIZE* in Figure 4-1 is two. Additionally, the aggregation method is directly defined as majority vote. The reference /47/ names this basic approach as label-based. Since if a photo satisfies a property, then this property may be considered a label for this photo. The corresponding interface for workers is depicted in Figure 4-1. The label-based approach asks one question for each photo. The other approach presented in /47/ is called the counting-based approach. The counting-based approach means that instead asking workers whether an item satisfies a property or not, it shows several items to workers and asks the workers how many items satisfy a property. Therefore, the UDF is modified as follows:

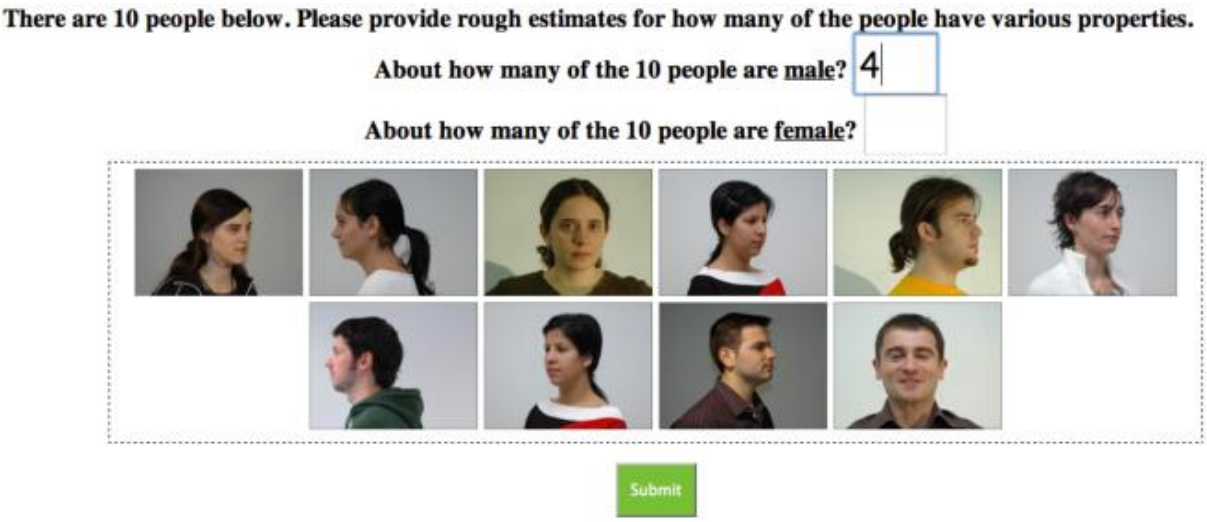

Figure 4-2: Counting-based interface/47/

```
TASK gender(field) TYPE Generative:
     ItemPrompt: "<table><tr> <td><img src='%s'>… <td></table>",
     tuple[field]
     PropertyPrompt: "About how many of the %d people are %s?",
     BATCHSIZE, PROPERTY
     Response: number
     CountPrompt: "There are %d people below. \Please provide
     rough estimates for how many of the people have various
     properties.", BATCHSIZE
     Combiner: MajorityVote
```

CountPrompt provides the overall instructions for workers. The corresponding interface for the counting-based approach is depicted in Figure 4-2. In /47/, these two approaches are respectively experimented on AMT and each of them are experimented with two instances. One instance is the picture example introduced above, the other instance is letting workers label or count twitter texts into three categories: Information Sharing, where a user links to a piece of information; Me Now, where a user says what they are doing now, and Question to Followers, where a user poses a question. The conclusion is a little surprising: For the picture instance, the counting-based approach can achieve commensurate accuracy to the label-based approach using an order of magnitude less HITs. For the text-based instance, the label-based

approach obviously has better accuracy than the counting-based approach/47/. Therefore, if the filter is applied to picture items, the counting-based approach may be used to improve the performance. Otherwise, the label-based approach is suggested to achieve a good accuracy although with poorer performance.

As mentioned before, crowdsourcing is flourish in recent years. One of the most important factors is that crowdsourcing is able to fulfil certain tasks as well as experts but with a lower monetary cost. Thus, there are usually budget constraints and a specific accuracy rate. In this situation, good strategies are needed to fit these problems to crowdsourcing. The query "filter" can be divided into three subclasses: the single filter, multiple filters and finding. A single filter is the simplest filter to judge if an item satisfies a single property or not. Multiple filters are to judge if an item satisfies several properties at the same time or not. Finding is a special filter, which aims to determine whether there are more than 'n' items in a given item set satisfying one or more properties. In /48/, the filter query is formally defined according to different metric and certain strategies are developed for the single filter. The strategies for the multiple filter queries are quite hard to solve in contrast to a single filter. Therefore, only the strategies for the single filter are provided in /48/. The reference /49/ defines finding queries formally and develops a series of algorithms to solve the finding problem in crowdsourcing environments. In the following, they are introduced respectively.

4.1.1.1 The Single Filter/52/

Formal Definitions: Given a set of items I, where |I| = n. V is used to express whether an item satisfies the filter (V = 1) or not (V = 0). The selectivity S of a filter means the probability that V = 1 over the whole item set, and it may be estimated through sampling a small number of items or obtained from a prior history. Since for some questions people are prone to have a high false positive rate, while for other questions people are prone to have a high false negative rate, the error rate should be expressed from the both false rates. Thus the false positive rate is: Pr[answer is yes | V = 0] = e_0; the false negative rate is Pr[answer is no | V = 1] = e_1; A *strategy F* is a computer procedure, which takes one item as input, then ask one or more workers questions on this item and at last outputs either "Pass" or "Fail". A *Pass* means this item satisfies the filter, while a *Fail* represents the opposite. A good strategy should be a strategy that makes few mistakes and without asking too many questions. The current state of an item can be represented as a pair (x, y). X represents the number of "yes"

answers that already were obtained from workers, y stands for the number of "no" answers that were obtained already. Therefore, a strategy can be visualized in a two-dimensional grid. Each state of the strategy can be represented as a pair (x, y), means the corresponding numbers of the already obtained "yes" or "no" answers. The input state of the strategy for any item is (0, 0). The sum of 'x' and 'y' in the output state of the strategy for any item is always the number of the workers that were assigned the corresponding question of this item. Figure 4-3 depicts an example of a strategy. The green points called continue points mean the answers for this question should be continued to collect (F(x, y) = Cont), the blue and red points called terminate points mean all the answers have been completely collected and the strategy has made a decision. The blue points means Pass (F(x, y) = Pass), while the red points means Fail (F(x, y) = Fail). The white points are called unreachable points, since the strategy cannot reach any white points. The basic requirement of a strategy for the filter queries should be uniform, complete and terminating. Uniform means, the same strategy is applied to each item from the given item set. Complete means, the strategy should tell what to do next at any reachable point. Terminating means, no matter in what sequence of "yes" or "no" answers are obtained, the strategy can terminate in the finite steps. Besides, the trends of a strategy should be fully predicted in order to be able to plan beforehand with constraint budget or accuracy.

Metrics: In crowdsourcing environments, the metrics to judge a strategy usually involve error and cost. In order to calculate the error rate and cost, two quantities are first defined: $P_1(x, y)$ is the probability that the strategy reaches point (x, y) and the item satisfies the filter (V = 1); $P_0(x, y)$ is the probability that the strategy reaches point (x, y) and the item does not satisfy the filter (V = 0). $P_1(x, y)$ and $P_0(x, y)$ can be calculated according to the following equations based on the value of e_1 and e_0:

$$P_0(x,y) = \begin{cases} P_0(x-1,y)(1-e_0)+P_0(x,y-1)e_0 & if\ F(x,y-1) = Cont \wedge F(x-1,y) = Cont \\ P_0(x,y-1)e_0 & if\ F(x,y-1) = Cont \wedge F(x-1,y) \neq Cont \\ P_0(x-1,y)(1-e_0) & if\ F(x,y-1) \neq Cont \wedge F(x-1,y) = Cont \\ 0 & if\ F(x,y-1) \neq Cont \wedge F(x-1,y) \neq Cont \end{cases}$$

$$P_1(x,y) = \begin{cases} P_1(x-1,y)(1-e_1)+P_1(x,y-1)e_1 & if\ F(x,y-1) = Cont \wedge F(x-1,y) = Cont \\ P_1(x,y-1)e_1 & if\ F(x,y-1) = Cont \wedge F(x-1,y) \neq Cont \\ P_1(x-1,y)(1-e_1) & if\ F(x,y-1) \neq Cont \wedge F(x-1,y) = Cont \\ 0 & if\ F(x,y-1) \neq Cont \wedge F(x-1,y) \neq Cont \end{cases}$$

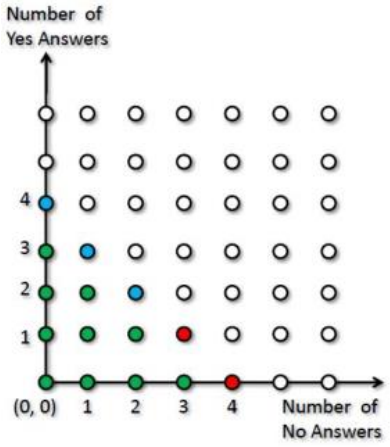

Figure 4-3: Representation of a strategy[48]

Therefore, the first metric *error* is defined as follows: The errors for any termination point are:

$$E(x,y) = \begin{cases} \dfrac{P_0(x,y)}{[P_0(x,y)+P_1(x,y)]} & if\ F(x,y) = Pass \\ \dfrac{P_1(x,y)}{[P_0(x,y)+P_1(x,y)]} & if\ F(x,y) = Fail \\ 0 & else \end{cases}$$

Then the expected errors of all termination points are:

$$E = \sum_{(x,y)} E(x,y) * [P_0(x,y) + P_1(x,y)].$$

The second metric *cost* is defined much easier than the metric *error*. C(x, y) for any termination point is its number of questions, i.e. x + y, then the expected cost of all termination points is:

$$C = \sum_{(x,y)} C(x,y) * [P_0(x,y) + P_1(x,y)].$$

Problem Statement: Given input parameters S, e_0, e_1, the problem to search for a good strategy can be stated in a variety of ways. In order to make the strategies terminate, assume that at most *m* different workers can be assigned to the same question. In the following, some variants for the problem statements are presented.

Problem 1: Given an error threshold τ and a budget threshold per item m, find a strategy F that minimizes C under the constraint E < τ and ∀(x, y) C(x, y) < m

Problem 2: Given an error threshold τ and a budget threshold per item m, find a strategy F that minimizes C under the constraint ∀(x, y) E(x, y) < τ and C(x, y) < m.

Above two problem statements are defined to get a minimum cost under different constrains. The error constraint in "Problem 1" requires the average errors less than τ, while in "Problem 2" the errors for each termination point should be less than τ. Instead of minimizing the overall cost, "Problem 3" minimizes the maximum cost for any given point.

Problem 3: Given an error threshold τ, find a strategy F that minimizes the maximum value of C(x, y) (over all points), under the constraint E < τ.

Another two variations are based on minimizing the errors. The difference between them is , there is an additional constraint on the overall cost in "Problem 5".

Problem 4: Given a budget threshold per item m, find a strategy F that minimizes E under the constraint ∀(x, y) C(x, y) < m.

Problem 5: Given a budget threshold per item m and a cost threshold α, find a strategy F that minimizes E under the constraints ∀(x, y) C(x, y) < m and C < α.

Strategies: There are two kinds of strategies: deterministic strategies and probabilistic strategies. The deterministic strategies can determine a direct conclusion for a given point (F(x, y) = *Pass*, *Fail* or *Cont*), while the probabilistic strategies provides *Pass*, *Fail* and *Cont* each with a probability for a given point, for instance, F(x, y) may be (Pass 0.3, Fail 0.1, Cont 0.6) in the probabilistic strategy. In the deterministic strategy, F(x, y) can only be (0, 0, 1) for green points, (1, 0, 0) for blue points and (0, 1, 0) for red points. In the following, the deterministic strategy is first introduced as a start point, but the probabilistic strategy is often also needed in uncertain crowdsourcing environments and will be introduced later. "Problem

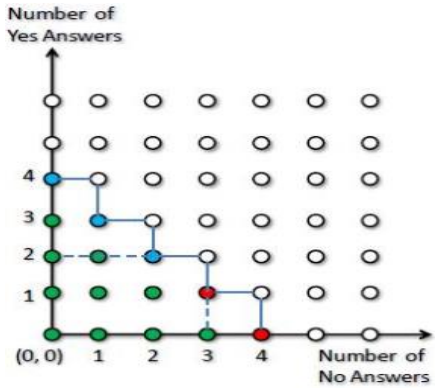

Figure 4-4: A ladder shape and a normal shape[/46/]

1" is as an example to find the best strategy. The algorithms to find answers to other problem statements are similar.

- **Deterministic Strategies:** There are three kinds of algorithms to find the best deterministic strategy called naive2, naive3, ladder shapes.
  - Naive3: This algorithm examines strategies with all possible assignments (3^{m^2} assignments) of *Pass*, *Fail*, and *Continue* for each point and calculate the overall cost of all strategies and then choose the best one.
  - Naive2: This algorithm first divides the points into termination points and non-termination points and then examines strategies with all assignments (2^{m^2} assignments) of Pass or Fail for each termination point, at last, choose the cheapest one.

  These two algorithms are named due to the number of the assignments 3^{m^2} and 2^{m^2}. Since evaluating cost and error for each strategy takes time $O(m^2)$ using the recursive equations, their corresponding complexity is $O(m^2 3^{m^2})$ and $O(m^2 2^{m^2})$. In practice, considering all possible 3^{m^2} or 2^{m^2} strategies is not feasible sometimes, hence, another algorithm considering only a subset of the total possible strategies is suggested.
  - Ladder shapes: A shape is defined by a connected sequence of horizontal or vertical segments on the grid, beginning at a point on the y-axis, and ending at a point on the x-axis, along with a special point, called a decision point. The decision point is to divide the termination points into *Pass* termination points or *Fail* termination points. Each shape corresponds to a strategy. If a shape

satisfies the following constrains, then the shape is called a ladder shape and its corresponding strategy is the best strategy. The constraints are before the decision point, the flat segments always go right, and the vertical segments always go up in the grid. After the decision point, the flat segments always go left and the vertical segments always go down. Then the real shape of segments before the decision point looks like a rising ladder and looks like a downward ladder after the decision point. A ladder shape and a non-ladder shape are depicted in Figure 4-4. The decision point is (3,2). The shape with the segments connected by the points (0, 4)(1, 4)(1, 3)(2, 3)(2, 2)(3, 2)(3, 1)(4,1) (4, 0) is not a ladder shape because the segments (1, 4) to (1, 3), (2, 3) to (2, 2) go down and the segment (3, 1) to (4, 1) goes right. The shape with the segments connected by the points (0, 2)(1, 2)(2, 2)(3, 2)(3, 1)(3, 0) is a ladder shape. The corresponding strategy is the best strategy. The exhaustive proof why a ladder shape corresponds to the best strategy can be found in /46/.

- **Probabilistic Strategy:** The intrinsic difference between probabilistic strategies and deterministic strategies is the points in the deterministic strategies are divided into termination points and non-termination points, while the points in the probabilistic strategies have no termination or non-termination, all the points are with probabilities to terminate or continue. Hence, the metric for the error and cost has to be changed:

$$\forall (x,y); x + y \leqslant m :$$

$$E(x, y) = b(x, y) * \min\left(\frac{P_0(x,y)}{[P_0(x,y)+P_1(x,y)]}, \frac{P_1(x,y)}{[P_0(x,y)+P_1(x,y)]}\right)$$

$$E = \sum_{(x,y);x+y\leq m} E(x,y) * [P_0(x,y) + P_1(x,y)].$$

b(x, y) is the probability the strategy terminate at that point. The error at a certain point is simply the probability that the strategy terminates at that point, times the smaller of the two error probabilities, since the strategies should choose *Pass* or *Fail* with a lower probability. Following is the method to calculate the cost in the probabilistic strategy:

$$\forall (x,y); x + y \leqslant m :$$

$$C(x, y) = b(x, y) \times (x + y)$$

$$C = \sum_{(x,y);x+y\leq m} C(x,y) * [P_0(x,y) + P_1(x,y)].$$

The cost at a given point is the total number of questions times the probabilities the strategy terminates at the given point. P_0 and P_1 is calculated in the probabilistic strategies as follows:

$\forall(x, y); x + y \leqslant m :$

$P_0(x, y) = e_0 \cdot P_0(x, y-1) \cdot a(x, y-1) + (1 - e_0) P_0(x-1, y) \cdot a(x-1, y)$

$P_1(x, y) = e_1 \cdot P_1(x-1, y) \cdot a(x-1, y) + (1 - e_1) P_1(x, y-1) \cdot a(x, y-1)$

(a(x, y) is the probability that the strategy continues in the point(x, y), a(x,y) +b(x,y) = 1)

Only the algorithm complexity is given in /46/: $O(m^4)$, the specific algorithms for choosing the best probabilistic strategy is not given, but definitely it can be implemented by calculating the minimum cost under an error threshold.

4.1.1.2 Finding /53/

The finding query can be considered as a special type of the filter query. The filter query is to find all satisfied items in an item set, while the finding query is to find a specific number of satisfied or not satisfied items from an item set. As examples, a travel website may want to identify 15 photos containing "Neuschwanstein" from a dataset of 100,000 travel photos. Of course, a finding query can be answered first by applying a filter strategy to find all satisfied items, then based on the filter result get the specific number of items from all satisfied or not satisfied items. Nevertheless, in this way it leads to such high costs and very low performance. Hence, studying particular strategies to solve the finding query is necessary. In this section, the study in /49/ for the finding query is introduced in detail.

Formal Definitions: Given a set of items I, where |I| = n, the goal is to find k_1 or more items that satisfy the filter, and k_0 or more items that do not satisfy the filter. In common crowdsourcing cases, finding k_1 satisfied items is the only need. P denotes the property and O denotes the output condition (k_1, k_0). For each item one question may be asked to workers on that item, workers may answer "yes" or "no" to reflect their thoughts. A tuple (I, (x, y)) represents the current state for the item I. x means the obtained "yes" and y means the obtained "no". If x and y are not both zero, then I is called a partially evaluated item. There are two scenarios for the finding query: one is the deterministic scenario and the other is the uncertain scenario. In the deterministic scenario, assuming workers can always answer the questions without errors. Then as long as x = 1 or y = 1 for the item I, the item is confirmed to

satisfy the property or not. In the uncertain setting, workers may make mistakes. A finding algorithm, denoted *A*, takes a data set of items as input and outputs the k_1 satisfied items and k_0 not satisfied items. If the algorithm finds k_1 satisfied items and k_0 not satisfied items, then it is said to return a *correct solution*. The algorithm A maintains a set of the partially evaluated items $S = \{I_i, (x_i, y_i)\}$, which means the item I_i receives x_i yes and y_i no. As the algorithm asks more questions to workers, either the already existing items' information is updated or new items' information is added into *S*. A finding algorithm proceeds in phases. In each phase, the algorithm performs the following three steps:

- Item Selection: In this step, algorithm A picks a multiset of $Q = \{I_i\}$.
- Questions: Then algorithm A in parallel assigns multiple questions on the items in Q to workers. The newly obtained answers are added to the set of the partially evaluated items S.
- Test for Solution: If the set of the partially evaluated items S satisfies the condition of (k_1, k_0), then the algorithm terminates. Otherwise, a new phase will repeat these three steps.

Metrics: Assuming that each question takes one unit of cost to answer and that a batch of questions asked in parallel needs one unit of time to be finished. The performance of an algorithm can be quantified by two metrics: latency and monetary cost.

- Latency T: The latency of an algorithm is defined as the total number of the phases T of the algorithm. It ignores other computation time such as testing whether sufficient items are picked, since such computation is automatically performed and can be ignore compared to the human tasks. In practice, the answers from different workers are not returned at the same time. A small number of answers may be returned much later. However, the common algorithms do not wait for the late answers and just ask a new batch of questions. Therefore, the latency of an algorithm is T.
- Monetary Cost C: The total monetary cost of an algorithm is the total numbers of questions asked in all phases. Let x_i denote i questions are asked in the *i-th* phase. Then the total cost is:

$$C = \sum_{i=1}^{T} x_i$$

Note that the worker errors E are not explicitly considered, since the errors do not exist in the deterministic setting and in the uncertain scenario, the errors are implicitly captured. If a question for an item is assigned to multiple workers, then by a proper strategy to infer the final answer from workers' different responses the error rate is acceptable in most cases.

Problem Statement: In general, there are the following forms of algorithms to solve the finding query based on the assumption that all workers do not make mistakes.

- Sequential: In each phase only one item is picked. Suppose on average there are 20% items in the item set which satisfy a property and K_1= 10, then 10/0.2 = 50 phases are needed to find the 10 items. Note that this algorithm is cost-optimal, because it only asks as many HITs as strictly necessary.
- Parallel: There is only one phase in this algorithm. In this sole phase, all items are picked to be asked. This algorithm needs extremely heavy monetary cost.
- Hybrid-1: K_1 items are picked in the first phase. As examples, pick 10 items in the first phase, then x_1 items are returned as satisfied items. Then pick 10- x_1 items in the second phase, x_2 items are returned as satisfied items, then pick 10- $x_1 – x_2$ items in the third phase and so on until 10 satisfied items are found. In this way, the expected number of phases is much smaller than 50, while the number of HITs is as same as the sequential algorithm.
- Hybrid-2: (K_1/Selectivity items) are picked in the first phase, hoping to get K_1 satisfied items directly from the first phase. For instance, 10/0.2 = 50 items are picked in the first phase, then x_1 items are returned as satisfied items. Then pick (10- x_1)/0.2 items in the second phase and so on, until 10 satisfied items are found. In this way, the expected number of phases is less than the number for algorithm Hybrid-1, while the expected number of HITs is bigger.

Next, different problems are formally defined according to the metrics:

> Problem 1(Sequential): Given a problem(P,O), design an algorithm A that asks one question in each phase and returns the correct solution incurring the least monetary cost Copt(I)for each problem instance I.

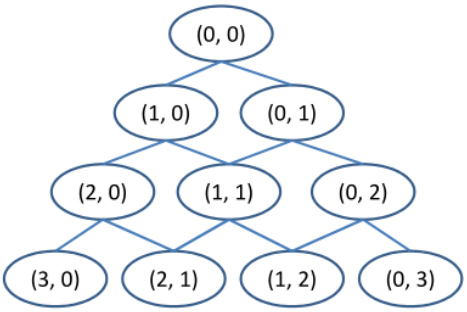

Figure 4-5: Possible States of the partially evaluated items set S[49]

$C_{opt}(I)$ means the optimal monetary cost. The sequential algorithm is trivial for the deterministic scenario, but is significant for the uncertain scenario.

> Problem 2(Cost-optimal Max-parallel): Given a problem(P,O), design an algorithm A that for each problem instance I, Algorithm A returns a correct solution with C(I) = $C_{opt}(I)$ and no other algorithms A′(which for every instance I, returns a solution and has C(I) = Copt(I))has lower latency.

> Problem 3(α -multiplicative-approx, Max-parallel): Given a problem(P,O), design an algorithm A that for each problem instance I, Algorithm A returns a correct solution with C(I) ≤ α $C_{opt}(I)$ and no other algorithms A′(which for every instance I, returns a solution and has C(I) ≤ α $C_{opt}(I)$)has lower latency.

> Problem 4(α -additive-approx, Max-parallel): Given a problem(P,O), design an algorithm A that for each problem instance I, Algorithm A returns a correct solution with C(I) ≤ α + $C_{opt}(I)$ and no other algorithms A′(which for every instance I, returns a solution and has C(I) ≤ α + $C_{opt}(I)$)has lower latency.

The problems described so far capture instance-specific guarantees, i.e., the algorithms for these problems are problem instances-oriented and aim at guaranteeing their cost or latency for each instance relative to the sequential algorithm. The next problem is to design an

algorithm to evaluate its expected monetary cost and latency, in this way, the algorithms can be compared relative to each other on the cost and latency.

```
Problem 5 (Expected Monetary Cost and Latency): Given an algorithm
for a problem(P,O), find its expected monetary cost and latency.
```

Deterministic Algorithms: In this part, the algorithms for the deterministic situation are presented based on the three steps introduced in *formal definitions* part. The deterministic situation means workers do not make mistakes. In this case, each item may be asked at most once, then the right answer can be returned. In the *item selection* step, the multiset of Q that contains the picked items can be simply represented using x, which indicates the number of the newly picked items in each phase. Additionally the set of the partially evaluated items *S* can be represented using a pair (*a*, *b*). *a* is the current number of already verified items that satisfy the property, b is the current number of already verified items that do not satisfy the property. Figure 4-5 depicts all possible states for *S* on asking up to three questions. Assuming that $k_1 = 2$, $k_0 = 0$, then the states (2,0), (2,1) and (3,0) satisfy the output condition, although (3,0) is not necessary. This output condition $k_1 = 2$, $k_0 = 0$ is assumed to the following algorithm examples.

- **Algorithm *OptCost*** for problem 2: The *OptCost* algorithm proceeds as follows: Assume that the current *S* is (*a*, *b*). Then in the next phase, $x_m + 1$ items are picked and each item generates one question, where x_m is the largest x that satisfies the following formula:

$$\forall\, i,j \geq 0, 0 \leq i + j \leq x, (a + i < k_1) \vee (b + j < k_0) = \text{True}$$

- **Algorithm *α –MultApprox*** for problem 3: The *α –MultApprox* algorithm proceeds as follows: Assume that the current S is (*a*, *b*), then the number of the picked items so far is $y = a + b$. In the next phase, this algorithm picks $\alpha * (y + x_m + 1) - y$ items. x_m is defined the same as in *OptCost* algorithm. Therefore, at each phase, this algorithm asks $(\alpha - 1) * (y + x_m + 1)$ more HITs than in *OptCost* algorithm.
- **Algorithm *α –AddApprox*** for problem 4: The *α –MultApprox* algorithm proceeds as follows: Assume that the current S is (*a*, *b*). In the next phase, this algorithm picks $x_m + \alpha + 1$ items. x_m is defined the same as in *OptCost* algorithm. Therefore, at each phase, this algorithm asks *α* more HITs than in *OptCost* algorithm.

In this thesis the proofs that these three algorithms are able to respectively solve problem 2, 3, 4 are omitted, they can be found in /49/. Suppose that the current state is (1, 3), the output condition $k_1 = 2$, $k_0 = 0$ and $\alpha = 2$. Then in the next phase, only 1 question is asked using algorithm *OptCost*, 6 questions are asked using algorithm α *–MultApprox* and 4 questions are asked using α *–AddApprox* algorithm. We can see from the procedures of the α *–MultApprox* algorithm and α *–AddApprox* algorithm, α *–AddApprox* algorithm picks always the same number of items each phase, while α *–MultApprox* algorithm picks more items at each phase than its former phase. Thus, a hybrid algorithm can be adopted according to the specific situation, i.e. at some phases α *–MultApprox* algorithm is applied and at the other phases α *–AddApprox* algorithm is applied.

Uncertain Worker Answers: This part considers the case in which humans may make mistakes.

- **Settings:** In this case, the worker error rate and the selectivity of the property for the item set are needed to know in advance. Both of them can be estimated using a golden standard sample. The expert evaluate the sample to get the selectivity and workers evaluate the sample to get the worker error rate. As an easy case, the output condition is set to be $(k_1, 0)$, that is, the algorithm needs only to find the k_1 satisfied items. The case that both k_1 and k_0 are not 0 can be considered as two finding queries respectively with the output condition $(k_1, 0)$ and $(0, k_0)$ respectively.
- **Strategies:** In suncertain scenarios, different responses may be returned to the same item. Thus, a strategy to infer the fact that an item satisfies the property or does not satisfy the property is required. There are many possible strategies, such as majority vote: choose the majority answer as the final answer or rectangle strategy: as long as one of the number of “yes” or “no” answers reaches n (n is defined by people), then the first reached answer is the final answer. The algorithms apply to all possible strategies.
- **Expected Cost Computation:** Given a tuple(I, (x, y)), $C_{next}(x, y)$ is defined to be the expected cost to find the next item that satisfies the property. There are two different situations:

- The item I is confirmed as the next item: Suppose that the probability for this situation is Pr_1 and the number of the HITs needed to confirm I satisfies the property is N_1.
- A new item picked from the item set is confirmed as the next item, since the cost to confirm I is higher than picking a new item. Suppose that the probability for this situation is Pr_2 and the number of the HITs needed to confirm a new item satisfies the property is N_2, then the total expected cost is $(C_{next}(0, 0) + N_2)$.

By combining the two situations, $C_{next}(x, y)$ is expressed as follows:

$$C_{next}(x, y) = Pr_1 * N_1 + Pr_2 * (C_{next}(0, 0) + N_2)$$

Next, the strategies that can be represented as *shape* introduced in *the single filter* part as examples to indicate how to get the value $C_{next}(x, y)$. Given a strategy, $C_{next}(x, y)$ is computed using the following recursive dynamic programming algorithm for all the points within the strategy. The computation begins at the boundary of the strategy and proceeds towards (0, 0). Consider a point(x, y), $P_{yes}(x, y)$ is the probability to get a "yes" answer at (x, y). $P_{no}(x, y)$ is the probability to get a "no" answer at (x, y). Then:

$$P_{yes}(x,y) = Pr(V=1 \mid (x,y)) * Pr(YES \mid V=1) + Pr(V=0 \mid (x,y)) * Pr(YES \mid V=0)$$

$$P_{no}(x,y) = Pr(V=1 \mid (x,y)) * Pr(No \mid V=1) + Pr(V=0 \mid (x,y)) * Pr(No \mid V=0)$$

These two expressions are independent of the strategy. Instead they depend on the point(x, y), the selectivity of the property, and the worker accuracy. At each point(x, y), either questions for a new item is asked or more questions for the current item is asked. For the latter case, if another "yes" answer returns, then the expected cost in the next point of (x, y) is $C_{next}(x + 1, y)$, otherwise $C_{next}(x, y + 1)$. Thus:

$$C_{\text{next}}(x,y) = \begin{cases} \min \begin{cases} C_{next}(0,0) \\ P_{yes}(x,y) * C_{next}(x+1,y) + P_{no}(x,y) * C_{next}(x,y+1) + 1 \end{cases} \\ 0, \quad if\ pass\ is\ returned \\ C_{next}(0,0), \quad if\ fail\ is\ returned \end{cases}$$

If the first case is used, a new item is picked and the strategy returns to the original point (0,0). Otherwise, additional questions are asked. To compute $C_{next}(x, y)$ on each point(x, y), the equation is recursively unfolded until the base case (0,0). $C_{next}(0, 0)$ is calculated by an approximate decision procedure: $C_{next}(0, 0)$ should be between 0 and

m. m is the boundary of a strategy. The procedure begins to attempt with a value between 0 and m for $C_{next}(0, 0)$, until the value approximates the result of this formula ($P_{yes}(0, 0) * C_{next}(1, 0) + P_{no}(0, 0) * C_{next}(0, 1) + 1$). Then $C_{next}(0, 0)$ is approximately this value.

- **Algorithms to solve the problems in the uncertain setting**:
  - **Algorithm *OptSeq* for problem 1:** Before the main process of the algorithm, $C_{next}(x, y)$ for all points in the given strategy is computed. Then a priority queue containing items with the corresponding $C_{next}(x, y)$ is formed, in each phase an item with the smallest $C_{next}(x, y)$ is picked and a question is asked to a worker. After the answer is returned, $C_{next}(x, y)$ for this item is updated. Note that at the beginning all items are with the same $C_{next}(o, o)$, then in the first phase, the algorithm picks a random item. Whenever an item is confirmed to a *Pass* or a *Fail*, it is removed from the queue.
  - **Algorithm *UncOptCost* for problem 2:** This algorithm is based on the *OptSeq* Algorithm. Assume that so far there are already n items confirmed to satisfy the property. Then instead in each phase only one item is picked in *OptSeq*, $(K_1 - n)$ items with lower $C_{next}(x, y)$ are picked. If the number of questions leading the item to be confirmed as *Pass* is smaller than Fail, the questions leading the item to be confirmed as Pass are asked and vice versa.
  - **α -cost approximation Algorithms ($\alpha > 1$) for problem 3 and 4:** A α -cost approximation algorithm asks more questions at each phase than the *UncOptCost* Algorithm. There are two ways to increase the number of questions at each phase: either increasing the number of picked items and maintaining the number of questions asked on each item or increasing the number of questions asked on each item and maintaining the number of picked items. Algorithm α-Expand implements the first idea and Algorithm α-Multiply implements the second idea.
    - **Algorithm α-Expand:** Given an $\alpha > 1$, there is a master instance, which is the same as the instance in the *UncOptCost* Algorithm; additionally there are $\alpha - 1$ slave instances, they mimic the master instance, i.e. the master instance asks a_i questions on the i´th item, the

slave instance asks a_i questions on its i´th item as well. The master instance has another task, it has to keep track of the total number of *Pass* items that have been found in all instances. As long as the *Pass* items are sufficient, all instances stop at once.

- **Algorithm α-Multiply:** This algorithm picks the same number of items in each phase as algorithm *UncOptCost*, but asks α times questions of algorithm *UncOptCost* for each picked item.

In this thesis the proofs that these four algorithms are able to solve problem 1, 2, 3 respectively are omitted, they can be found in /49/.

So far, the introduced algorithms are used to solve the problems that provide instance-specific guarantees. Next, the algorithms to solve problem 5 are introduced to provide the expected cost and latency guarantees and enable the comparison among different algorithms. The algorithms are based on the assumption that the output condition asks for k_1 satisfied items, and that humans do not make mistakes.

Problem 5:

In order to distinguish the expected cost and latency from the instance-specific cost and latency, the expected cost and latency are denoted as EC and ET. In the following, EC and ET of completely sequential, completely parallel and intermediate algorithms are analyzed.

- **Completely Sequential Algorithm:** As introduced above, this algorithm means asking one question each phase, until K_1 satisfied items are found. EC and ET in this algorithm are both equal to K_1/s. s is the selectivity of a property.
- **Completely Parallel Algorithm:** As introduced above, this algorithm means asking all items questions at a single phase. Then ET = 1, EC = n. n is the number of items in the given item set.
- **Intermediate Algorithm:** There are certain algorithms speeding up the completely sequential algorithm by asking more than one question each phase to decrease the number of phases. For instance, in the first phase, k_1 questions are asked and $s*k_1$ satisfied items are found. Then in the second phase, (k_1-s*k_1) questions are asked, $s(k_1-s*k_1)$ satisfied items are found and in the i-th phase, $k_1(1-s)^{i-1}$ questions are asked, this number should be smaller than one, then we can say in the i-th phase k_1 satisfied items haven been found. Therefore,

$$\mathrm{i} > 1 - \frac{\log k_1}{\log(1-s)}$$

and the total number of questions are

$$\sum_{j=1}^{j=i} k_1(1-s)^{j-1} = k_1 \frac{1-(1-s)^i}{s}$$

This instance can be generalized by asking α times questions at each phase, then

$$\mathrm{i} > 1 - \frac{\log \alpha k_1}{\log(1-\alpha s)}$$

and the total number of questions:

$$\sum_{j=1}^{j=i} k_1(1-s)^{j-1} = k_1 \frac{1-(1-\alpha s)^i}{s}$$

However, in practice, the obtained number of satisfied items at each phase is actually less than the expected number, to ensure with a very high probability the sufficient number of satisfied items can be obtained at each phase, the number of questions at each phase are scaled up by multiplying a factor β. Then the expected cost and latency are guaranteed to be lower than the following ET and EC:

$$\mathrm{ET} \leq 2 - \frac{\log(\alpha k_1)}{\log(1-\alpha s)} \text{ and } \mathrm{EC} \leq \beta k_1 \frac{1-(1-\alpha s)^i}{s}$$

4.1.2 Sorting[/54/]

Sorting is a familiar operation in databases, it is implemented by the SQL clause "ORDER BY". It orders the tuples in a table according to the values for one or more specific attributes. But the ORDER BY clause used in the traditional database can organize data only in an alphabetic or a numeric order. With the help of the crowdsourcing, the sorting can be extended also in a more meaningful order. For instance, given some photos about a restaurant, asking the crowd to sort them according to the fact, which photo describes the restaurant better, and then the better photos may be showed in its website. In this section, the sorting ideas based on the above introduced crowdsourced database Qurk in crowdsourcing environments are demonstrated.

A sorting query in Qurk can be expressed in its SQL-based query language:

```
SELECT column
FROM table
ORDER BY FunctionName(column name)
```

The corresponding UDF may be like:

```
TASK FunctionName(type value) RETURNS (String []):
     TaskType: Rank
     Text: "Rank the following list of ...  by ..." or "please
     rate the following… and give your own opinion"
     Response: List: column name
```

"Rank" in the "TaskType" field shows that the HIT is letting workers rank a list of subjects. The "Text" field provides two description examples, which stand for two ideas to design a HIT (Human Intelligent Task). The first idea called the comparison-based sort is in a direct way, given several subjects, letting the crowd order them on request and return the ordered subjects. The other idea called the rating-based sort means given several subjects, letting the crowd rate each subject and return the corresponding scores. In order to let readers easier understand, an example is used to more visually explain the ideas for the crowdsourced sorting query. Suppose there are a table of images of squares of different sizes: *squares(label text, img url).* Then the following SQL-based query is used to sort these squares by their areas:

```
SELECT squares.label
FROM squares
ORDER BY squareSorter(img)
```

The UDF "squareSorter" is defined as follows:

```
TASK squareSorter(field) TYPE Rank:
     SingularName: "square"
     PluralName: "squares"
     OrderDimensionName: "area"
```

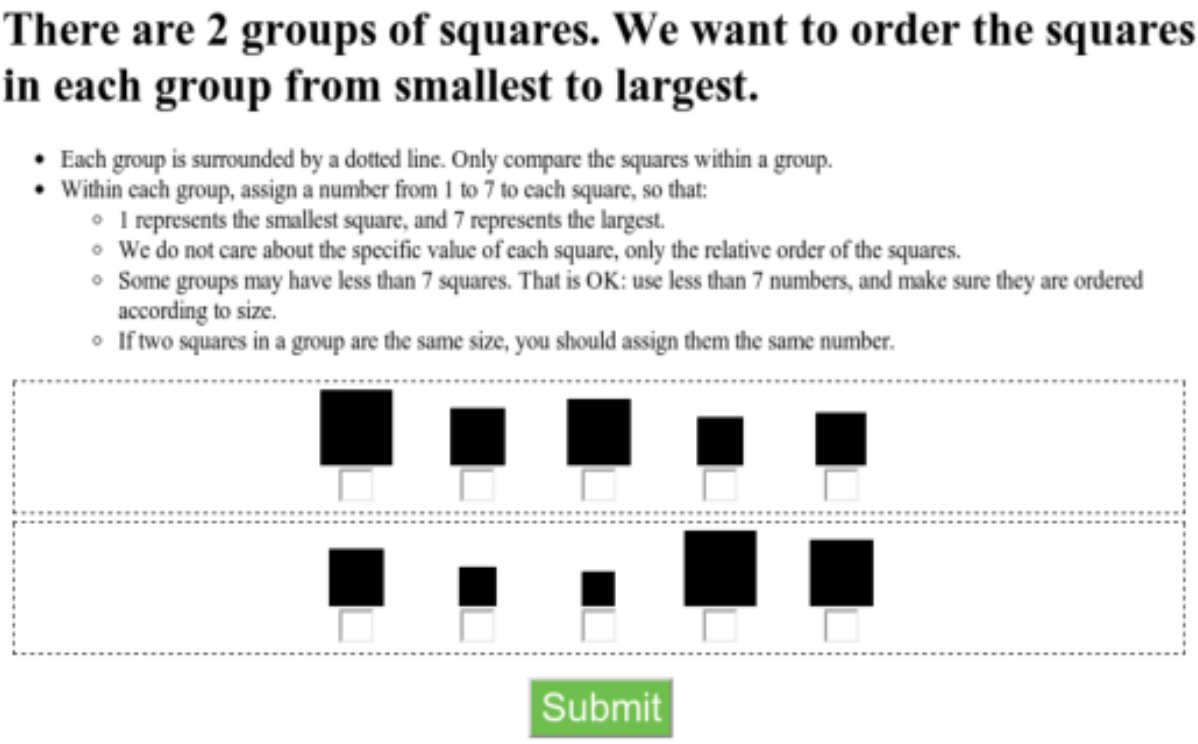

Figure 4-6: Comparison-based sort[/54/]

```
LeastName: "smallest"
MostName: "largest"
Html:"<img src='%s' class=lgImg>",tuple[field]
```

Comparison-based Sorting: A naive comparison-based sorting is given two subjects for each HIT to ask the workers to compare, but consequently there will be too many HITs, hence, the designers put S subjects in one HIT and ask the crowd provide the order of the S subjects on requests. Assume that there are N tuples of squares in the dataset. In this way, the total number of HITs are reduced from $\binom{N}{2}$ to $\frac{N(N-1)}{S(S-1)}$. After the workers submit their order of S subjects, the order is transformed into $\binom{S}{2}$ pairwise comparisons and then the pairwise comparisons are integrated into an order of the entire dataset. The interface of the above mentioned example in the comparison-based way is depicted in Figure 4-6. This example asks the crowd to order two groups of squares according to their areas in one HIT, each group is with 5 squares.

Rate-based Sorting: In contrast to the comparison-based sorting, the rating-based sorting needs much less HITs, the number of HITs can also be reduced by batching. Batching means in each HIT more than one subjects are asked to be rated by workers. The interface of the above mentioned example in the rate-based way is depicted in Figure 4-7. Although the rate-based sorting needs much less HITs than the comparison-based sorting, it has a disadvantage in contrast to the comparison-based sorting, the sorting order obtained by the rate-based sorting is not as good as the sorting order obtained by the comparison-based sorting according to the experiments on AMT. Therefore, the designers utilize the advantages of both methods and design a hybrid sorting.

Hybrid Sorting: The hybrid sorting first adopts the rate-based sorting and generates a list L. Each item in the list has an average rate calculated from the ratings from different workers and a corresponding standard deviation. Then the hybrid sorting algorithm attempts to identify the subsets in L that may not be accurately ordered and adopts the comparison-based sorting to order them again. According to the experiments on AMT, the hybrid sorting can reach a similar accuracy of the comparison-based sorting and at the same time costs only one-third of the comparison-based sorting.

4.1.3 Top-K

Top-K queries can be solved based on the sorting ideas in the above section by adding a LIMIT clause[/54/]. For instance,

```
SELECT squares.label
FROM squares
ORDER BY squareSorter(img)DESC
LIMIT K
```

4.1.4 Maximum[/55/]

As the same as the top-k queries, the maximum queries can also be solved by adding a LIMIT clause based on the result of the sorting result[/54/]. For instance,

```
SELECT squares.label
```

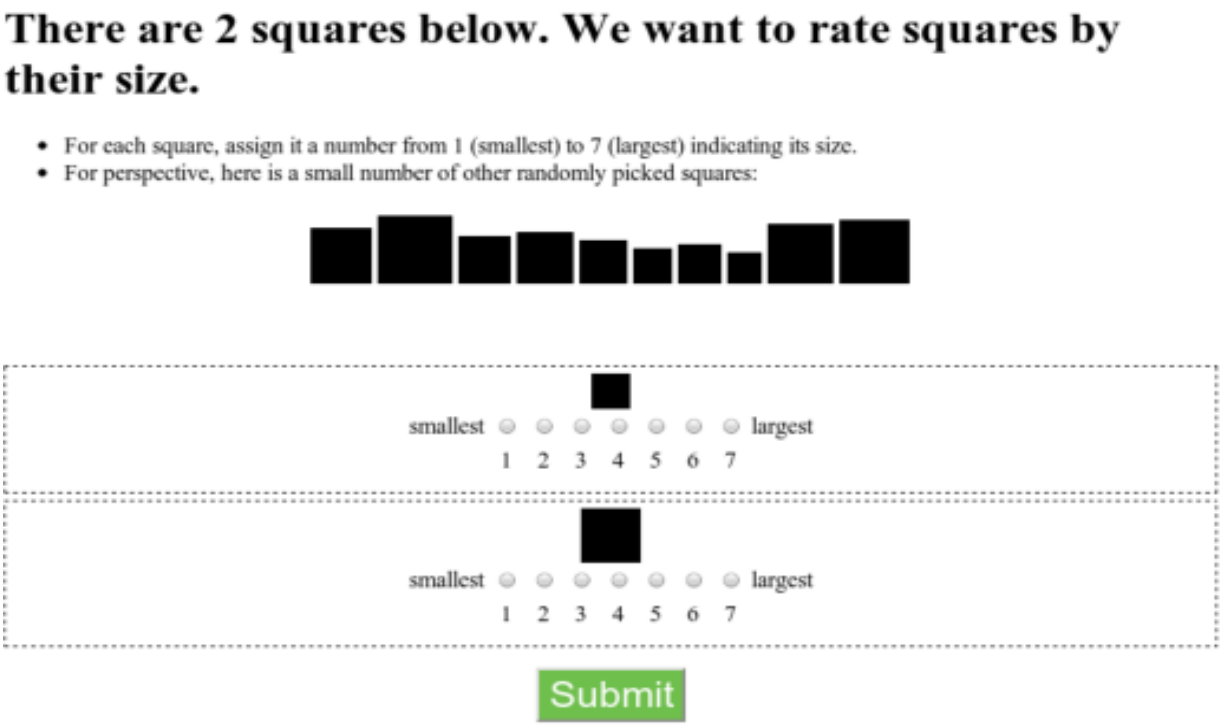

Figure 4-7: Rate-based Sort/54/

```
FROM squares
ORDER BY squareSorter(img)DESC
LIMIT 1
```

Beyond that, some current researches specially aim at processing maximum queries in crowdsourcing environments. In /55/, the authors studied two problems: the judgment problem and the next votes problem based on the premise of the pairwise comparison and fixed worker errors. In general, the judgment problem means among the responses that were already obtained from different workers, what is the current best estimate for the overall maximum? Next votes problem means: given the current state of the responses, if the requester wants to invoke more votes, which votes are the most efficient ones to invoke? In the following, these two problems will be described in detail.

Judgment Problem:

- **Formal Definitions:** Given a set O of n objects$\{o_1, \ldots, o_n\}$, where each object o_i is associated with a latent quality c_i ($c_i \neq c_j$). If $c_i > c_j$, then we say that c_i is greater than c_j. π denotes a permutation, $\pi(i)$ denotes the rank of object o_i in π and $\pi^{-1}(i)$ denotes the

$$W = \begin{pmatrix} 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 3 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Figure 4-8: The Matrix Representation[55]

objects of the *i-th* position in π. If $\pi(i) < \pi(j)$, then o_i is ranked former than o_j in permutation π, i.e. $c_i > c_j$. Since $c_i \neq c_j$, then $\pi(i) \neq \pi(j)$, the permutation is a true permutation and denoted as π^*. Pairwise votes are used in order to find the maximum object. In a vote, workers are shown two objects o_i and o_j, and are asked to vote which one has greater c? The results of already obtained votes can be represented in a matrix W. W is a n * n matrix, W_{ij} is the number of obtained votes that o_j is greater than o_i, thus W_{ii} is always equal to 0. The results of already obtained votes can also be represented as a directed weighted graph $G_v = (v, a)$. The vertices in the graph denote the subjects and the weight for the arc from o_i to o_j denotes the number of already obtained votes that o_j is greater than o_i. For example, Figure 4-8 is the matrix representation for a set O with four objects. Figure 4-9 is the corresponding graph representation. Object 1 in Figure 4-9 is called A, object 2 is B, and so on. For instance, there is an arc from vertex B to vertex C with weight 2 because $W_{2,3} = 2$, and there is a reverse arc from C to B with weight 1, because $W_{3,2} = 1$.

- **Problem Statement:** The judgment problem is formally defined as follows:

  `Given W, predict the maximum object in O, π*-1(1).`

- **Maximum Likelihood:** Given a vote matrix W, the maximum likelihood algorithm computes the probability that each object is the maximum object in O, then the object with the highest probability is the maximum. Assuming that the average worker accuracy p is known, the probability that o_j is the maximum is computed by the following equation:

$$\mathrm{P}(\pi^{-1}(1) = \mathrm{j} \mid \mathrm{W}) = \frac{\sum_{d:\pi_d^{-1}=j} P(W|\pi_d)}{\sum_l P(W|\pi_l)}$$

  Π is a random variables over the set of all n! possible permutations, assuming π_d is one possible permutation. The numerator of the right part in the above equation

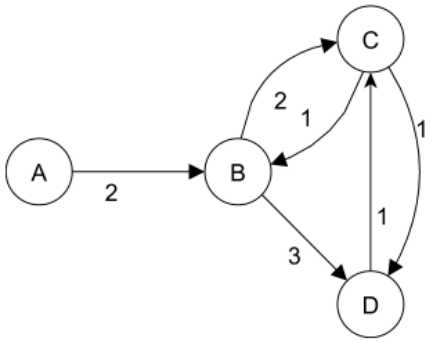

Figure 4-9: The graph representation/55/

$\frac{\sum_{d:\pi_d^{-1}=j} P(W|\pi_d)}{\sum_l P(W|\pi_l)}$ means the sum of the probabilities that o_j is the maximum in each possible permutation π_d. Each probability $P(W|\pi_d)$ can be computed using the following equation:

$$P(W|\pi_d) = \prod_{i,j:i<j} f_{\pi_d}(i,j)$$

$f_{\pi_d}(i,j)$ is the probability that o_j is greater than o_i in π_d and it can be calculated using the following equation:

$$f_{\pi_d}(i,j) = \begin{cases} \binom{W_{ij}+W_{ji}}{W_{ij}} P^{W_{ji}}(1-p)^{W_{ij}} & if \pi_d(i) < \pi_d(j) \\ \binom{W_{ij}+W_{ji}}{W_{ji}} P^{W_{ij}}(1-p)^{W_{ji}} & if \pi_d(j) < \pi_d(i) \end{cases}$$

The maximum likelihood is the optimal feasible solution to the Judgment Problem. Nevertheless, in /55/ calculating the probabilities that each object is the maximum is proved with a computation complexity of #P-Hard, i.e. it is quite inefficient to find the maximum object by the maximum likelihood algorithm. However, note that the maximum likelihood algorithm can induce an optimal result; it can be used as a baseline to estimate other efficient algorithms.

- **Heuristic Strategies:** In general, a solution to a judgment problem is based on a scoring function s. The scoring function calculates the scores for all the objects, the "confidence" that an object is the true maximum is represented by the scores. The scoring function of the above introduced maximum likelihood algorithm is equation 1.

In the following, four heuristic strategies are provided to solve the judgment problem based on their respective scoring functions.

- **Indegree Strategy:** Note that the Indegree strategy is a heuristic, although it can only provide an approximate maximum instead of an optimal maximum, it is much more efficient than the maximum likelihood. From its name "indegree" some ideas of this strategy can be seen: The weight W_{ij} on each in-arc of o_j represents the number of votes that indicate o_j is greater than o_i. The scoring function in this strategy is by calculating the sum of the probabilities of o_j is greater than all the other objects:

$$S(j) = \sum_i l_{ij}$$

l_{ij} is the probability that o_j is greater than oi based on the local W_{ij} and W_{ji}, it can calculated in the following equation:

$$l_{ij} = \frac{\binom{W_{ij}+W_{ji}}{W_{ji}} P^{W_{ij}}(1-p)^{W_{ji}}}{\binom{W_{ij}+W_{ji}}{W_{ji}} P^{W_{ij}}(1-p)^{W_{ji}} + \binom{W_{ij}+W_{ji}}{W_{ij}} P^{W_{ji}}(1-p)^{W_{ij}}}$$

If there is no arc between two vertices o_i and o_j or the arcs between o_i and o_j is symmetry, then l_{ij} and l_{ji} are equal to 0.5. For instance, assuming p = 0.55, the corresponding l_{ij} of the above matrix (Figure 4-8) or the graph (Figure 4-9) is*:* $l_{AC} = 0.5$, $l_{CA} = 0.5$, $l_{AD} = 0.5$, $l_{DA} = 0.5$, $l_{CD} = 0.5$, $l_{DC} = 0.5$, $l_{AB} = 0.599$, $l_{BA} = 0.401$, $l_{BC} = 0.55$, $l_{CB} = 0.45$, $l_{BD} = 0.646$, $l_{DB} = 0.354$. Then S(A) = 1.401 ,S(B) = 1.403, S(C) = 1.55, S(D) = 1.65, at last the maximum object is D. If p is changed, the conclusion that D is the maximum will remain the same.

- **Local Strategy:** It is simple to get the maximum by computing the Indegree scoring function, but the score of an object o_j only depends on the votes that contains o_j directly. The local strategy fills the gap. Consider the votes two steps away from o_j. This strategy first computes the wins and losses for each objects. Wins and losses of an object is defined as follows:

$$wins(j) = \sum_i W_{ij}\,;\ losses(j) = \sum_i W_{ji}$$

For instance, the wins and losses of vertices B and C of Figure 4-9 are: wins(B) = 3, losses(B) = 5; wins(C) = 3, losses(C) = 2.

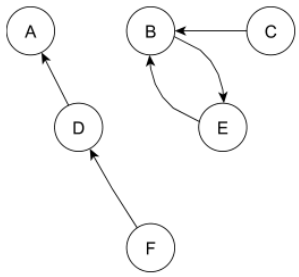

Figure 4-10: Example for the next votes problem[/55/]

The wins minus the losses of an object o_j can reflect the overall support for o_j, the local strategy also considers the strength of the contrary objects o_i which o_j was compared against. If $W_{ji} > W_{ij,}$ then o_j is rewarded the wins of o_i, in reverse if $W_{ji} < W_{ij}$, then o_j is penalized the losses of o_i, i.e. the scoring function is:

$$S(j) = wins(j) - losses(j) + \sum_i [1(W_{ij} > W_{ji}) wins(i)] - \sum_i [1(W_{ij} < W_{ji}) losses(i)]$$

According to this scoring functions, the scores of the vertices in Figure 4-9 are computed: s(A) = 0−2−5 = −7, s(B) = 3−5−3 = −5, s(C) = 3−2+3 = 4, and s(D) = 4−1+3 = 6. Therefore, D is the maximum.

- **PageRank Strategy:** Actually, both the Indegree strategy and the local strategy consider only from the local votes. While the PageRank strategy is a global heuristic. The scoring function of the PageRank is actually an iteration procedure. The initial PageRank of all the objects is $1/n$, then the PageRank of each object is iterative by the following equation:

$$PR_{t+1}(j) = \sum_i \frac{W_{ij}}{d^+(i)} PR_t(i)$$

$d^+(i)$ is the out degree of vertex I, that is $d^+(i) = \sum_i W_{ji}$, If $d^+(i)=0$, vertex i is called a sink vertex. For each iterative, each vertex i transfers its current PageRank proportionally to the other vertices j which is voted that may be greater than i. The proportion is equals to $\frac{W_{ij}}{d^+(i)}$, which means the percent j contributes to i's out degree. Considering the strongly connected components (SCC) in the graph, after a sufficient number of iteration, the SCC will terminate, that is, the PageRank will concentrate in the SCC, the PageRank of all the other vertices outside of the SCC will be zero. In practice, assuming K

times iteration is performed and it is sufficient to get the period of nearly all vertices in the terminal SCC. Then the PageRank scoring function is the average PageRank for each vertex over its period. Consider again the Figure 4-9, there are two SCCs in the graph: (A) and (B, C, D). At the beginning, each vertex is with a PageRank 0.25. After the first iteration and the second iteration, both the PageRanks of four vertices are (0, 0.375, 0.35, 0.275). After 20 times iteration, the PageRanks oscillates around (0, 0.217, 0.435, 0.348). After a large number of iteration, the period is found to be 1. Then the average PageRanks of four vertices is calculated and is equal to (0, 0.217, 0.435, 0.348). Finally, C is concluded as the maximum.

- **Iterative Strategy:** The iterative strategy has a general framework: All vertices are first put into one set, then eliminating the vertices from the set by comparing the result obtained by a metric, the metric and the number of the eliminated vertices each time can be defined as needed. Then iterating this procedure, until only one vertex is left. The scoring function in the iterative strategy is the iteration number in which the vertex is removed from the set. For instance, the iterative strategy is performed with the metric of the value of wins(i) minus losses(i) and each time two vertices are removed. At the first iteration, wins(A) – losses(A) = -2, wins(B) – losses(B) = -2, wins(C) – losses(C) = 1, wins(D) – losses(D) = 3, then A and B are removed. At the second iteration, wins(C) – losses(C) = 0, wins(D) – losses(D) = 0, thus C or D is the maximum with the same probability. If like this, the metric and the number of the eliminated vertices each time can be changed in order to determine only one vertex as the maximum.

Note that excluding the maximum likelihood and Indegree strategies, the other three strategies do not need the knowledge of the worker accuracy. The above introduced five algorithms are experimented in /55/, the following conclusions are obtained:

- The maximum likelihood has better performance than the other four heuristics.
- PageRank is a poor heuristic when worker accuracy is low. However, with a high worker accuracy and a low number of votes, PageRank is the best among the four heuristics.

- Over various worker accuracies, Iterative is the best heuristic, followed by PageRank, Local and Indegree.

Next Votes Problem: Sometimes the initial votes matrix is not sufficient to get a constraint accuracy, then more votes are needed to improve the prediction of the maximum. Therefore, the next votes problem is studied and strategies to select and evaluate additional votes are suggested. In the following, next votes problem is formally defined.

- **Formal Definition:** A potential vote between o_i and o_j is represented as a unordered pair$\{o_i, o_j\}$. Each pair is called a vote multiset. If the number of additional votes is constraint to b, then a multiset Q contains all possible b vote multisets and the repetition of votes are permitted. Q is represented as Q = $\{\{o_i, o_j\}\{o_j, o_k\}\ldots\{o_k, o_l\}\}$. Each vote multiset corresponds to an answer multiset tuple($\{o_i, o_j\}, o_x$). o_x is the object that a worker votes that o_x is the greater one in this vote multiset. Then the corresponding answer multiset is represented as A(Q) = $\{(\{o_i, o_j\}, o_x)(\{o_j, o_k\}, o_x)\ldots(\{o_k, o_l\}, o_x)\}$. Note that for a vote multiset of b additional votes, there are 2^b possible answer tuples.
- **Problem Statement:** The next votes problem is formally defined as follows:

  Given b, W, select b additional votes and predict the maximum object in O, $\pi^{*-1}(1)$.

- **Maximum Likelihood:** The purpose of making additional votes is to improve the "confidence" of the chosen maximum. Therefore, the maximum likelihood strategy for the next vote problems computes the "confidence" of the maximum of all possible obtained answer multisets *a* based on all possible vote multisets Q, then choose the vote multiset with the highest "confidence" of the maximum. The sum of the "confidence" of the maximum of all possible obtained answer multisets based on the vote multiset Q is computed as follows:

$$\sum_{a \in A(Q)} max_i P\left(\pi^{-1}(1) = i,\ a \bigwedge W\right)$$

  The new obtained answer tuples *a* is combined with the initial matrix W, the result of the combination is the current vote matrix ($a \wedge$ W)**.** The maximum likelihood strategy is an exhaustive strategy and it can provide an optimal solution to the next votes

problem. Nevertheless, its computational complexity is as the same as the maximum likelihood strategy for the judgment problem: #P-hard and it requires the knowledge of worker accuracy p. Therefore, a general framework to select and evaluate additional votes is developed to efficiently solve the next votes problem.

- **General Framework:** The general framework has three steps:
  1. Computer all objects with an introduced scoring function for the judgment problem according to the initial vote matrix W.
  2. Select a batch of b votes to request.
  3. Evaluate the new matrix W´ ($a \wedge W$) with a scoring function to predict the maximum in O.

  Next, four heuristic vote selection strategies to implement the step 2 are presented: Paired, Max, Greedy, and Complete Tournament strategies. For ease of understanding, an example in Figure 4-10 is used to explain the four strategies. Assuming two additional votes are allowed and the iterative strategy is used in step 1 to score all objects. The result of wins() minus losses() is as the metric and in each iterative one object is removed from the set, then the scores can be: A = 6, B = 5, D = 4, E = 3, C= 2, F = 1 and the corresponding rank is {A, B, D, E, C, F}.

  - **Paired Vote Selection:** This strategy restricts that each object can be involved in at most one additional votes. The objects ranked in the front have the priority to be involved in the additional votes. Therefore, {A, B} and {D, E} are the additional votes.
  - **Greedy Vote Selection:** This strategy first computes all the possible votes score by the product of the scores of the two objects. In the example, {A, B} and {A, D} have the highest score 30 and 24, thus these two pairs are the additional votes.
  - **Complete Tournament Selection:** This strategy first chooses K objects with the highest scores, K is the largest number that satisfies $\frac{K(K-1)}{2} \leq b$. Each of the K objects is compared to other objects in the K objects. Then the remaining r = b - $\frac{K(K-1)}{2}$ votes contains the (K + 1)th object and one of the K objects. The selection of the remaining votes is based on the product of the scores of the

Figure 4-11: Join HIT interface/48/

two objects. In the example, K is equal to 2, then 2 objects with the highest scores A and B are chosen and the pair {A, B} is one of the two votes. The other vote is {A, E} with a higher product 30.

- **Max Vote Selection:** In this strategy, the current top-ranked object is asked to be compared with the rank second object, rank third object, until rank *(b + 1)th* object. Since b is equal to 2, then in this example {A, B} and {A, D} are the additional votes.

The four heuristic strategies and the maximum likelihood strategy are experimented in /55/ and the following conclusions are obtained:

- The maximum likelihood strategy outperforms the other strategies with the maximum likelihood scoring function. However, when the maximum likelihood strategy is scored by other heuristic strategies, the prediction performance is similar to the others.
- The Complete Tournament and Greedy strategies are significantly better than the Max and Paired strategies. The paired strategy is suitable for certain initial states where many objects have the same score.

4.1.5 Join/Entity Resolution

The ability to "join" two or more tables is one of the most powerful features of relational databases. Loosely speaking, a join query is a query in which data is retrieved from more than one table[/56/]. It allows establishing connections among data contained in different tables and comparing the values contained in them[/57/]. Nowadays the comparison is not as simple as before, since there are more and more different expressions for the same entity, more cases with the comparisons among different media and more cases that need human's subjective comparison. Then the join queries with certain comparisons are extremely difficult for the traditional databases. Therefore, crowdsourcing is used to help processing join queries. In this section, the overall flow to process the join queries via crowdsourcing based on Qurk is first demonstrated. Next the entity resolution via crowdsourcing is focused on. Entity resolution is the basis of the comparison; it aims at clustering records that refer to the same real-world entity.

Join Queries Based On Qurk[/54/]

Join queries are expressed by the SQL-based query language in Qurk as follows:

```
SELECT table1.column1 table1.column2 FROM table1 table2
WHERE FunctionName(table1.column1, table2.column1)
```

The UDF may be like this:

```
TASK FunctionName(type  value, type  value) RETURNS BOOL:
     TaskType: JoinPredicate
     Text: "Drag ...of any...in the left column to their
     matching object in...to the right."
     Response: DragColumns("...", ... ,"...", ...)
```

" JoinPredicate" in the "TaskType" field shows that the HIT is letting workers join two table-valued arguments. The "Text" field provides a description example. The "Response" field

constrains workers to drag subjects and give the response that contains two columns labelled by the two tables.

For ease of understanding, suppose that there are two tables that are needed for a join query: photos(img url) and celeb(name text, img url). Then the join query is:

```
SELECT c.name
FROM celeb c JOIN photos p
ON samePerson(c.img,p.img)
```

The UDF *samePerson* to implement the equijoin task may be like this:

```
TASK samePerson(f1, f2) TYPE EquiJoin:
     SingluarName: "celebrity"
     PluralName: "celebrities"
     Text:”Is the same celebrity in the image on the left and
     the image on the right?”
     LeftPreview: "<img src=’%s’ class=smImg>",tuple1[f1]
     LeftNormal: "<img src=’%s’ class=lgImg>",tuple1[f1]
     RightPreview: "<img src=’%s’ class=smImg>",tuple2[f2]
     RightNormal: "<img src=’%s’ class=lgImg>",tuple2[f2]
     Response:Choice(“YES”, “NO”)
     Combiner: MajorityVote
```

The samePerson UDF is much more specific than the UDF templates above. Its type is equijoin, means the join query is to find two photos from two tables with an identical celebrity. Its aggregation method is set to MajorityVote, which is the most popular method to aggregate different responses. F1 and f2 are the fields that are used to generate one of several different join user interfaces. The basic idea of the interfaces is to ask users to compare pairs of elements from the tuple1 and tuple2. Figure 4-11 is a sample interface. The join HIT interface asks a worker to compare objects from two joined tables, after the HIT results return, Qurk evaluates the result to infer whether the two objects satisfy the join condition. In Qurk,

batching optimization is used to improve the performance. There are two kinds of batching methods called the naive batching and the smart batching. In the naive batching, several pairs of objects are displayed vertically in one HIT, each pair is followed by two buttons "yes" and "no". After a worker finished all pairs' comparison, the worker can click the "submit" button to submit his or her answer. The naive batching interface is depicted in Figure 4-12. In the smart batching, two columns of photos are displayed, the left column contains photos from the *photos* table and the right column contains photos from the *celeb* table. The workers are asked to select a matching pair of photos by double clicking the pair, then the pair will be displayed in the right part of the interface. The smart batching interface is showed in Figure 4-13. In order to save space, the photos in the two columns are displayed in the small size. If the mouse hovers over a photo, the photo will be displayed in full size. Suppose that the two tables are with |R| and |S| records respectively, then the simple join (Figure 4-11) needs |R|*|S| HITs, the naive batching (Figure 4-12) needs $\frac{|R|*|S|}{b}$ HITs (b is the batch size) and the smart batching (Figure 4-13) needs $\frac{|R|*|S|}{r*s}$ HITs (r is the number of photos in the left column, s is the number of photos in the right column). Therefore, the batching technique can save monetary costs and reduce the whole latency in overall.

In /54/, another optimization is introduced, that is join pre-filter. The pre-filter is implemented with a POSSIBLY keyword that indicates that this filter may help the join query. For instance:

```
SELECT c.name
FROM celeb c JOIN photos p
ON samePerson(c.img,p.img)
AND POSSIBLY gender(c.img) = gender(p.img)
AND POSSIBLY skinColor(c.img) = skinColor(p.img)
```

Qurk only asks the workers to compare the objects with the same property of the pre-filter. The property is obtained also via crowdsourcing. For instance, the gender UDF may be as follows:

```
TASK gender(field) TYPE Generative:
```

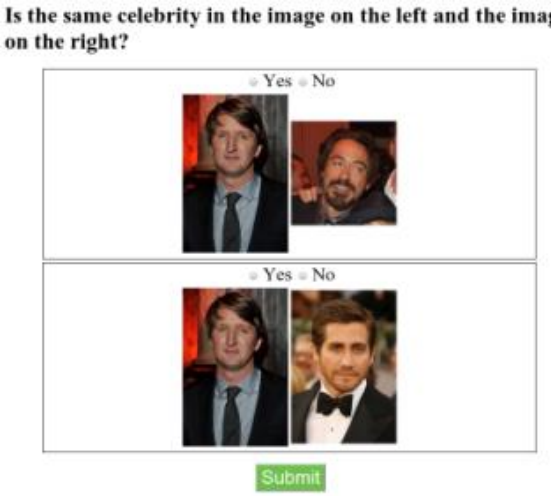

Figure 4-12: Naïve batching interface/48/

```
Prompt: "<table><tr> <td><img src='%s'> <td>What is this
person's gender?</table>", tuple[field]
Response: Radio("Gender",["Male","Female",UNKNOWN])
Combiner: MajorityVote
```

The above introduced techniques are experimented in /50/. The batching optimization provides a very considerable performance benefit. Both the naive batching and the smart batching can maintain acceptable accuracy and reduce the number of HITs. The smart batching with 5 and 5 columns is still acceptable, while for the naive batching, 25 batch size is too big, therefore smart batching is the better choice if there are too large numbers of records in the two joined tables. The pre-filter optimization can also achieve a much better performance, if the pre-filter is chosen properly. /58/ suggests to choose the Prefilter also via crowdsourcing, in order to further save the monetary cost to employ an expert to choose the join pre-filter and benefit from the wisdom of crowds. The rest of this section focuses on the entity resolution via crowdsourcing.

Hybrid Methods for Entity Resolution via Crowdsourcing

Entity resolution is to cluster the records that refer to the same real-world entity. The most two common cases are: 1. Given a set of media (such as photos, videos), find two or more media which belongs to the same person or other objects. 2. Given a set of different

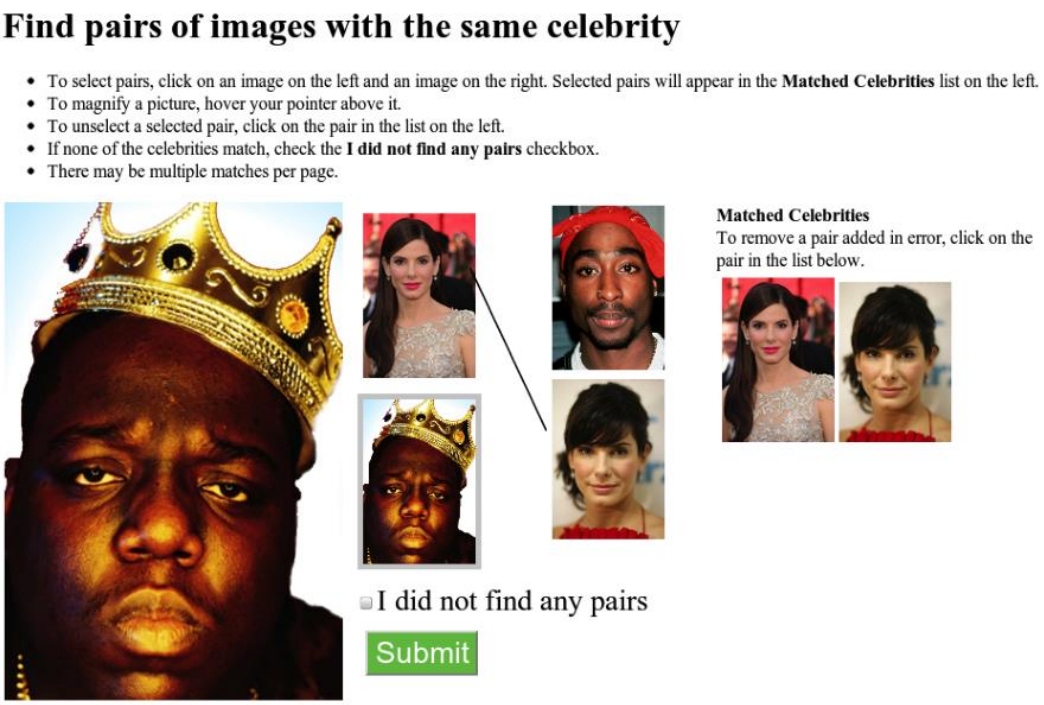

Figure 4-13: Smart batching interface[/48/]

expressions, find the expressions which describe the same objects. The above example belongs to the first case. Next, the two researches are based on the second case, taking different expressions as examples to cluster them. Computer algorithms to solve the entity resolution are improved, but the accuracy of the result is still far from perfect. Crowdsourcing brought a new thinking in entity resolution to get higher accuracy but the requesters, which need the entity resolution result, have to spend money on it. Although in the former part: join queries based on Qurk, batching optimization and pre-filter optimization are suggested to reduce the monetary cost, the optimization is only a palliative, i.e. if the set of the different expressions is very large, the monetary cost spent on crowdsourcing is unacceptable. Hence, some researchers suggest adopting a hybrid human-machine approach to solve the entity resolution problem. That is, at first letting the computer algorithms do the entity resolution, then crowds are used to verify only the algorithm results which have a high probability to be wrong. /59/ and /60/ are two researches on the hybrid approach to solve the entity resolution. Next, the two researches are demonstrated respectively and then are compared.

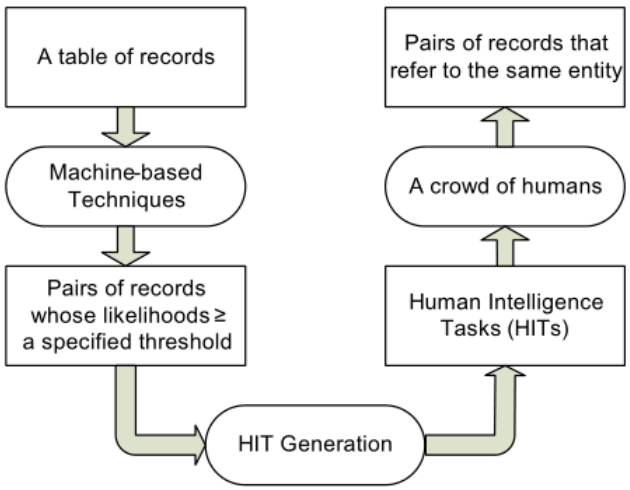

Figure 4-14: Hybrid human-machine Workflow[/59/]

4.1.5.1 Research in /59/

Hybrid Workflow: Figure 4-14 depicts the hybrid workflow in /59/. The current machine-based techniques are mainly divided into two categories: similarity-based and learning-based. The similarity-based approach is using a similarity function to calculate the similarities for each pair of records, then the similarity score beyond the threshold indicates the two expressions refer to a same entity. The learning-based approach models the entity resolution problem as a classification problem. A pair of records is represented as a feature vector in which each dimension is a similarity score of the records on some attribute. In the hybrid workflow, the machine-based technique adopts the similarity-based approach. An example of using the hybrid workflow to cluster records is depicted in Figure 4-15.

HIT Generation: There are two basic methods to generate the HITs: pair-based and cluster-based. The pair based approach is very similar to the naïve batching in the former part. Figure 4-16 Shows an example of the pair-based user interface. Generating pair-based HITs is straightforward. Given a set of pairs R, then $\lceil \frac{|R|}{k} \rceil$ pair-based HITs are generated (k is the batching size). Cluster-based method seems like the smart batching in the former part. However, this research focuses on the entity resolution, not the join queries, therefore in one HIT a set of records are together given instead of two columns of records from two tables. Figure 4-17 depicts the cluster-based user interface. We can see that all records are listed in a table and there is a drop-down list at the front of each record. Workers can label each record. Each label corresponds to a background colour, after workers select a label for a record, the

record's background colour will changed correspondingly. The generation of the cluster-based HITs is not as simple as the pair-based HITs. If too few records are contained in one HIT, then the monetary cost will be very high, since one payment unit is for one successfully completed HIT. If too many records are contained in one HIT, then the worker accuracy will be very low. Hence, the cluster-based HIT generation should be considered. This problem is formally defined as follows:

Given a set of pairs of records, R, and a cluster-size threshold, k, the cluster-based HIT generation problem is to generate the minimum number of cluster-based HITs, $H_1, H_2, \cdots, H_h$, that satisfy two requirements: (1) $|H_l| \leq k$ for any $l \in [1, h]$, where $|H_l|$ denotes the number of records in H_l; (2) for any $(r_i, r_j) \in R$, there exists $H_l (l \in [1, h])$ with $r_i \in H_l$ and $r_j \in H_l$.

In /59/, it is proved that the cluster-based HIT generation problem is NP-hard. Hence, a two-tiered approach is designed to solve the problem. In order to adopt this approach, a set of pairs of records are first converted into a graph. Each record is a vertex and each edge in the graph represents that the two records are in a pair. For instance, Figure 4-18 is an example of the conversion. After a set of pairs of records is converted into a graph, the graph is divided into two groups according to the cluster-size k: large connected components and small connected components. If there are more vertices than *k* in a component, this component is a large connected component. Otherwise, the component is a small connected component. Since a large connected component has more vertices than the number of records in a cluster-based HIT, it has to be partitioned to several small connected components. This is the top tier in this approach. The bottom tier is to pack all the small connected components into the minimum number of cluster-based HITs. The top tier is implemented as follows: In each large connected component, the vertex with the maximum degree is formed into a small connected component *scc*. Then the other vertices that connect it with an edge are placed into a set *conn* and the indegree and the outdegree of each vertex are calculated. Each time, one vertex in *conn* is picked with the maximum indegree into *scc*, if more than one vertices are with the same indegree, the vertex with the maximum outdegree is picked and the set *conn* is updated as long as a vertex is picked out. This procedure repeats until the vertices in *scc* reach k or the

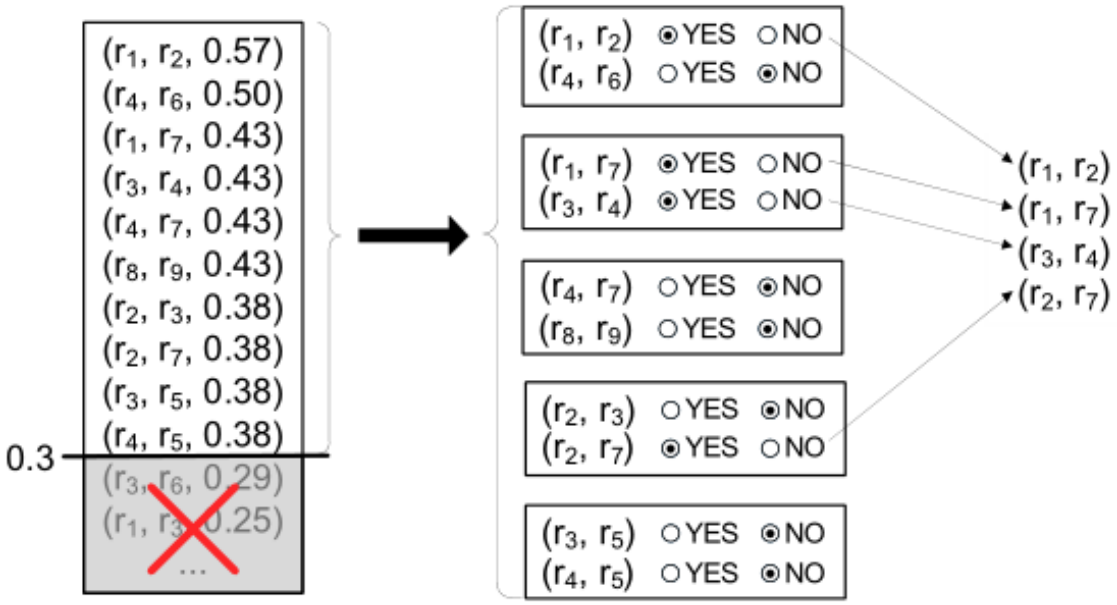

Figure 4-15: An example of using the hybrid workflow[59]

set *conn* is empty. This top tier of the above mentioned example (Figure 4-18) is depicted inFigure 4-19. The cluster size is assumed to be 4, then the component above is a large connected component. This large connected component is partitioned into three small connected components {r_3, r_4, r_5, r_6}, {r_1, r_2, r_3, r_7}and {r_4, r_7}. Next, the bottom tier implementation is introduced. As above mentioned, the bottom tier packs the small connected components into the minimum number of cluster-based HITs. This problem is actually a variant of the one-dimensional cutting-stock problem. This problem is a NP-hard, but it can be formulated as an integer linear program. The formulation procedure is as follows: Let P = {a_1, a_2,…,a_k} be a pattern of a cluster-based HIT, a_j is the number of the small connected components that contains j vertices. Since the cluster size is k, then $\sum_{j=1}^{k} j * a_j \leq k$. All feasible patterns are collected into a set A = { p_1, p_2,…,p_m}, where p_i = {a_{i1}, a_{i2},…,a_{ik}}, i ∈ [1, m]. Each cluster-based HIT must correspond to a pattern. Let x_i denote the number of cluster-based HITs which correspond to the *i-th* pattern. Then the problem is formulated as the following integer linear program:

$$\texttt{Min} \sum_{i=1}^{m} x_i$$

$$\texttt{s.t.} \quad \sum_{i=1}^{m} a_{ij}\, x_i \geq c_j, \forall j \in [1,k], x_i \geq 0, integer$$

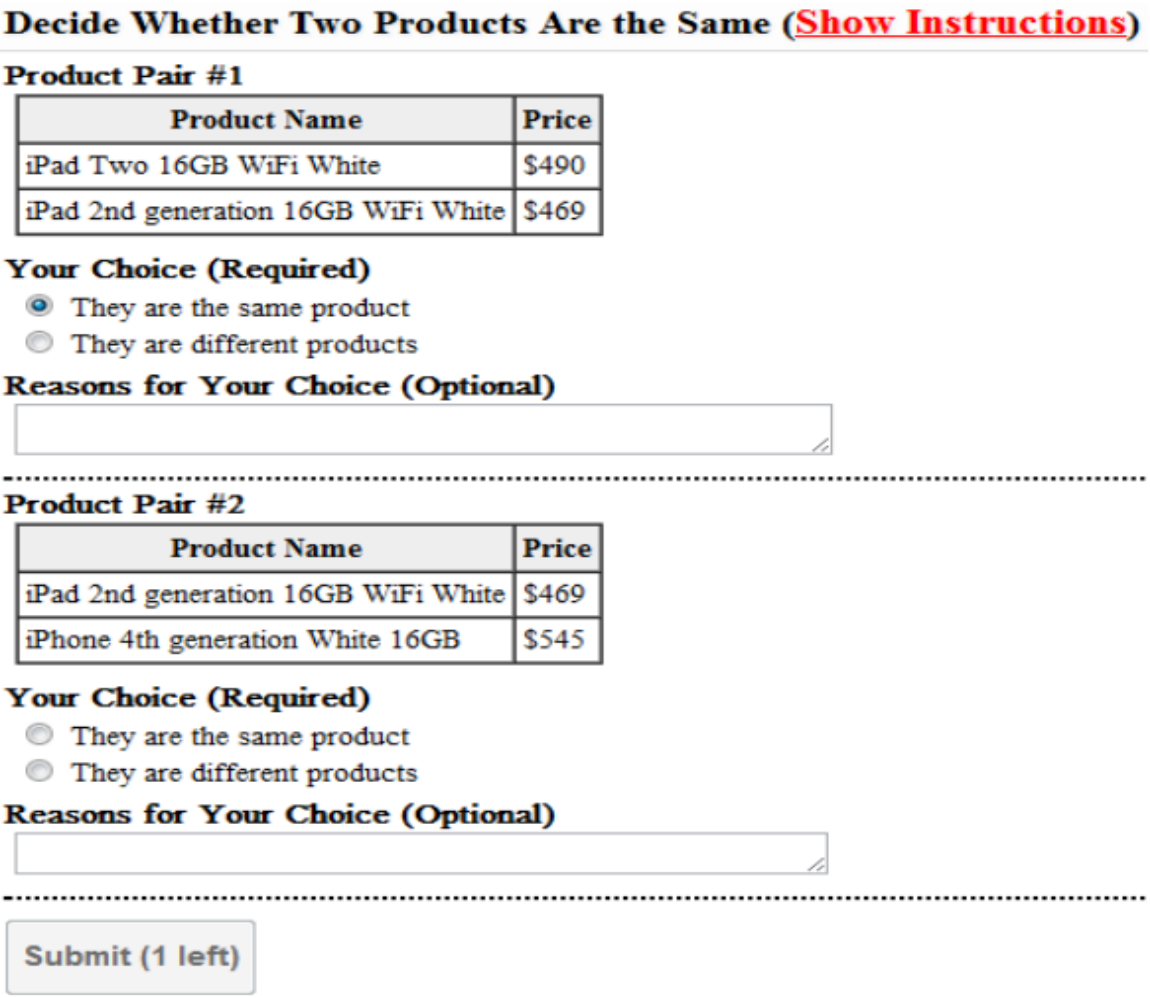

Figure 4-16: Pair-based user interface[/59/]

This integer linear program can be solved by using column generation and branch-and-bound[/61/]. This technique is very efficient, since it is not necessary to find all the patterns at the beginning. Instead it begins with only a few patterns, and more patterns are generated as needed. At each iteration, a brand-and-bound tree is built to find the optimal integer solution.

4.1.5.2 Research in /56/

Hybrid Workflow: The hybrid workflow in /54/ is depicted in Figure 4-20. It starts with the machine-based technique based on a similarity function to get pairs with similarity scores. This is called pairwise analysis phase. Then a portion of pairs are chosen to be identified by the crowd. Next, the identified pairs and the other portion of pairs directly obtained by the pairwise analysis phase beyond a specific threshold are analyzed by the global machine-based

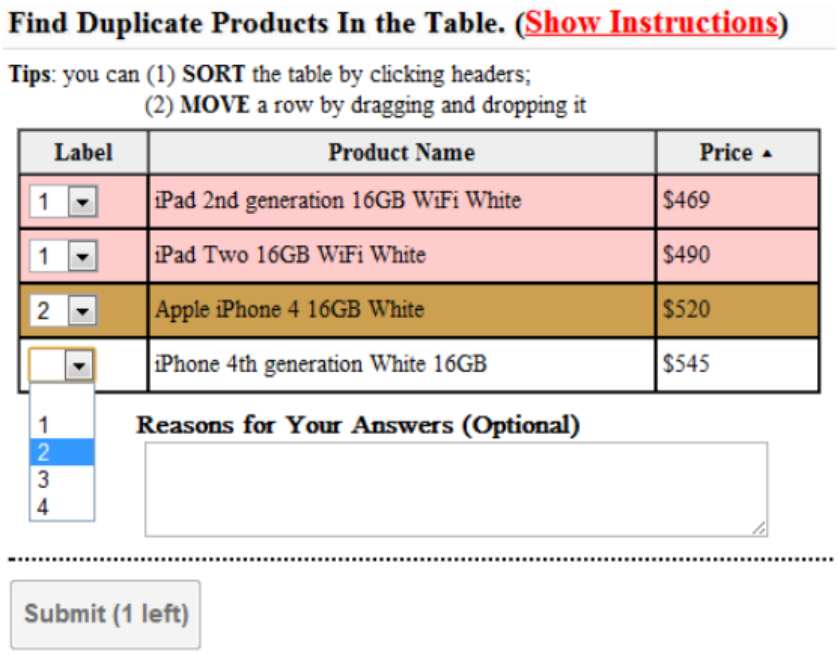

Figure 4-17: cluster-based user interface[/59/]

technique, such as transitive closure. This is called global analysis phase, which takes as input a set of record pairs, each with a similarity value, and clusters the records that are deemed to represent the same entity. At last, the final pairs are output.

Metrics: Traditionally, in order to evaluate the entity resolution accuracy, a gold standard data set is used. The gold standard data set means that its correct answers are known in advance. The gold standard data set is trained as the sample, an algorithm is proceeded on the gold standard data set, and then the accuracy for the algorithm can be obtained. There are two kinds of correct answers for the entity resolution. Let *C()* denote the set of all pairs that obtained after the pairwise analysis phase. *S()* denotes the set of all pairs obtained after the global analysis phase. One correct answer for *C()* is denoted as *C*()* and the other correct answer for *S()* is denoted as *S*()*. Note that given *C*()*, *S*()* can be obtained by the global analysis. But given an independent *S*()*, *C*()* may not be obtained. Hence, the gold standard used in /60/ is the *S*()* based on *the C*()*. The metric used to evaluate the final output O based on *S*()* is the F_1 metric. It is the harmonic mean of two values, P and R, i.e. $\frac{2*P*R}{P+R}$. P is the fraction of the pairs in O for which *S*()* holds, R is the fraction of all pairs in *S*()*.

Question Selection: In /60/, the problem which pairs should be contained in questions that are asked to the crowd in order to get the best final result is studied.

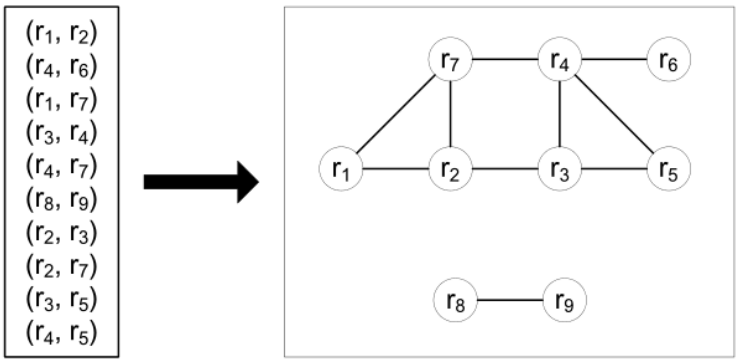

Figure 4-18: Convert a set of pairs into a graph[/59/]

- **Match Probability:** The goal of the entity resolution is to find pairs of records that refer to the same entity. Thus, the questions with a higher probability that crowds answer them with "yes" should be selected to ask the crowd. The probability can be inferred by the similarity scores of the pairs. The inference is based on the training data, which is the gold standard data. Assume that the actual data that needs to be resolved has the same characteristics with the training data. Then the needed probability can be obtained in this way.
- **Possible World:** There are two kinds of possible worlds: *C* and *S*, which stand for the pair sets with a probability. C possible worlds mean, the pair sets after the pairwise analysis phase, S possible worlds mean, the pair sets after the pairwise analysis phase and the global analysis phase. For instance, Figure 4-21 is an example of *C** and **S* possible worlds. The pairs (a,b),(a,c),(b,c) are represented in the graphs. Each record is a vertex, and the edge with a weight denotes the similarity score of the pair F(x,y) (Figure 4-21 above left). Then the weight is converted to the probability P(x,y) (Figure 4-21 above right) that workers answer a pair with "yes" according to the inference obtained by the training data. Then the *C* possible worlds are showed. The probability of each C possible world is calculated of multiplying the probabilities of all pairs and the probabilities of no pairs between records:

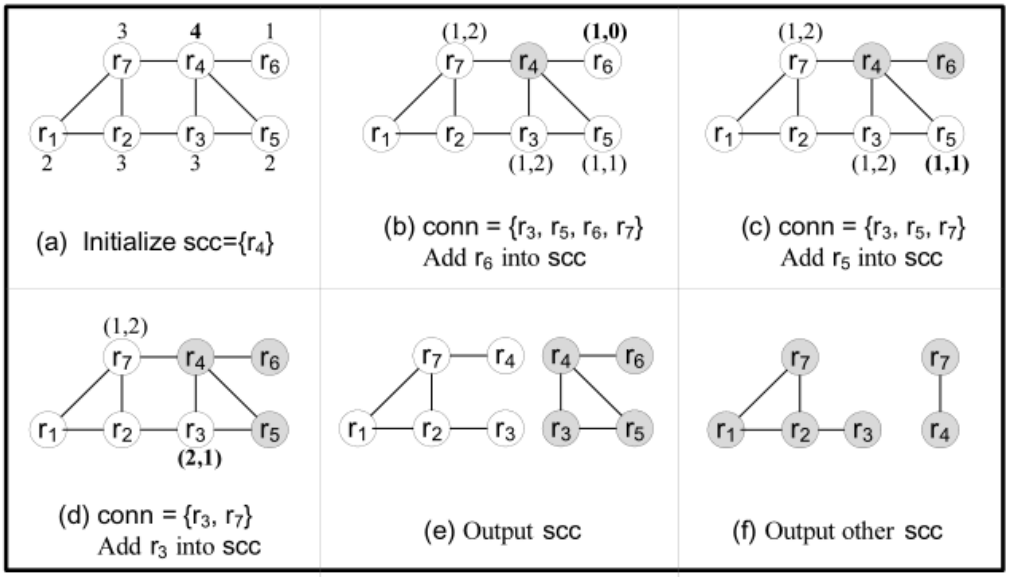

Figure 4-19: Top tier implementation procedure[/59/]

$$\mathrm{P}(\mathrm{P},\mathrm{C}) = (\prod_{a.b\in C} P(a,b))\,(\prod_{a,b\notin C}(1-P(a,b)))$$

The *S** possible world in Figure 4-21 is obtained by the transitive closure. From the possible worlds how good an entity resolution output is expected to be can be evaluated. The above mentioned metric $F_1(O, W_s)$ is used to calculate the accuracy of the output in *S* possible worlds. The global analysis algorithm is denoted as $E(R, W_c)$. Then $ACC(O,W_s)$ replaces $F(O, E(R, W_c))$ to represent the accuracy of the output O in S* possible worlds. Given a set of C* possible worlds L of the probability graph P and a global analysis algorithm, the expected accuracy of an entity resolution result O against a random possible world W∈ L is as follows:

$$\mathrm{E}\big(\mathrm{Acc}(\mathrm{O},\mathrm{W}_s)\big) = \sum_{W_c\in L} p(P,W_c) * Acc(O,W_c)$$

For instance, suppose the result O is {a, b}. The accuracy of O based on the C* possible worlds in Figure 4-21 is calculated as follows:

$$0.12*\mathrm{Acc}(\mathrm{O},\mathrm{W}_{c1})+0.48*\mathrm{Acc}(\mathrm{O},\mathrm{W}_{c2})+0.08*\mathrm{Acc}(\mathrm{O},\mathrm{W}_{c3})+0.32*\mathrm{Acc}(\mathrm{O},\mathrm{W}_{c4})=$$

$$0.68*\mathrm{F}_1(\mathrm{O},\mathrm{W}_{s1})+0.32*\mathrm{F}_1(\mathrm{O},\mathrm{W}_{s2})=0.68*\frac{2*1*1/3}{1+1/3}+0.32*\frac{2*1*1}{1+1}=0.66$$

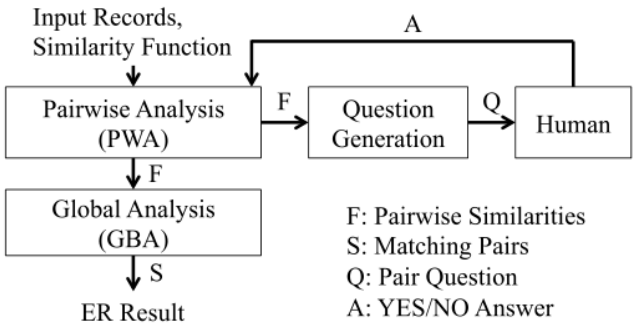

Figure 4-20: Hybrid human-machine workflow[/60/]

- **Choosing the best question:** Using the information in the S* possible world, the benefit of asking a question to the workers can also be calculated. Suppose that the question on the pair(x,y) is asked to workers, denoted as Q(x,y). Assume that the workers do not make mistakes. If the question is answered with "yes", then the similarity score for pair(x,y) F(x,y) = 1, otherwise, F(x,y) = 0. Then the weight of the edge for pair(x,y) in the probability graph is changed to be 1 or 0 and the corresponding possible worlds should be inferred again. Then the expected accuracy to ask a question on the pair(x,y) is calculated by the following equation:

$$\mathrm{E}(\mathrm{Acc}(\mathrm{O},\mathrm{W}_s)|A(x,y)) = \mathrm{P}(\mathrm{x},\mathrm{y}) * \mathrm{E}\left(\mathrm{Acc}\left(\mathrm{O}_{1,}\mathrm{W}_{s1}\right)\right) + \left(1-\mathrm{P}(\mathrm{x},\mathrm{y})\right) * \mathrm{E}(Acc(O_2,W_{s2}))$$

$$\mathrm{A}(\mathrm{x},\mathrm{y}) = \begin{cases} 1, & the\ question\ on\ the\ pair(a,b) has\ already\ been\ answered \\ 0, & else \end{cases}$$

For instance, the case in Figure 4-21 is used as examples. The expected accuracy to ask the question Q(b,c) is calculated as follows: Assume the threshold is 0.7. The situation that a worker answers Q(b,c) with "yes" is first considered. The output O_1 in this situation is {(a,b)(b,c)(a,c)}, since P(b,c) = 1, there is only one S* possible world{(a,b)(b,c)(a,c)}, then $\mathrm{Acc}\left(\mathrm{O}_{1,}\mathrm{W}_{s1}\right) = 1$. The other situation is a worker answers Q(b,c) with "no", then there are two S* possible worlds {(a,b),(b,c),(a,c)} with the probability 0.2 and {(a,b)} with the probability 0.8. The output O_2 is{(a,b)}. Then $\mathrm{Acc}\left(\mathrm{O}_{1,}\mathrm{W}_{s1}\right) = 0.2 * \frac{2*1*1/3}{1+1/3} + 0.8$. Hence, $\mathrm{E}(\mathrm{Acc}(\mathrm{O},\mathrm{W}_s)|A(b,c)) = 0.6 * 1 + 04 * 0.9 = 0.9$. In the same way, $\mathrm{E}(\mathrm{Acc}(\mathrm{O},\mathrm{W}_s)|A(a,b)) = 0.66$,

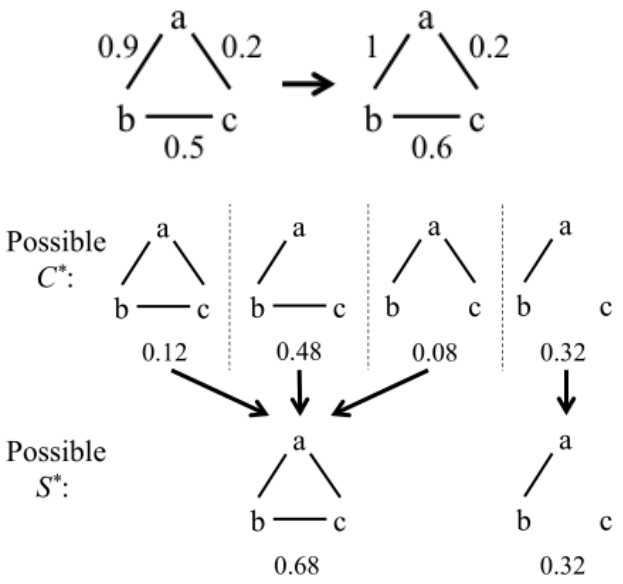

Figure 4-21: Match graph and possible worlds

$E(\mathrm{Acc}(O, W_s)|A(a,c)) = 0.76$. Hence Q(b,c) should be selected to reach the highest expected accuracy. This computing of E[Acc(O,W)] is #P-hard in general. If the exhaustive computation is proceeded in the same way as the computation in the above example, the time complexity is exponential. Hence, two approximation algorithms are provided to reduce the exponential runtime to the polynomial runtime: GCER (General Crowd Entity Resolution) and half algorithm.

- **GCER Algorithm:** The GCER algorithm is based on the accurate computation procedure showed in the former part. It reduces some redundant computation and produces an approximation result within polynomial time. In the first step, the algorithm prunes the questions, which probabilities are higher than the high threshold *h* or lower than the low threshold *l*. This step does not reduce the complexity but reduce the actual runtime. Then the Monte-Carlo method is used to reduce the time complexity. Instead of generating all C*possible worlds, a random set of the C* possible worlds is selected. Then, the corresponding S* possible worlds are obtained and the accuracy is calculated. By repeating the sampling and the calculation of the accuracy, an average accuracy is produced and it is very close to the real accuracy. The approach to decide the right number of samples is explained with detail in /62/. Although the Monte-Carlo method reduces the complexity into polynomial, the

number of questions to ask still increases in proportion to the edges in the probability graph (Figure 4-21 above right). The expected number of samples can be reduced to a constant by sharing the samples among different questions and different answers. The expected accuracy is only calculated for the questions with a probability between *l* and *h* instead of 0 and 1 ($0<l<h<1$). At last, the incremental entity resolution method is used. That is, some redundant computation can be removed by only resolving records that may be "influenced" by the current question. For example, suppose the global analysis algorithm performs a transitive closure on matching records in R = {a,b,c,d,e,f}. If the S* possible world is {(a,b),(c,d),(e,f)} and the current question is Q(a,c), then the global analysis algorithm is only rerun on the set {a,b,c,d} instead of the entire R.

- **Half Algorithm:** Half algorithm is a simple heuristic that chooses the optimal question by picking the edge in the probability graph, whose probability is close to 0.5. This algorithm is intuitively designed to select the most uncertain pairs to be verified by workers. In this way, a largest benefit may be achieved.

4.1.5.3 Comparison

The hybrid workflow in /59/ does not contain the global analysis phase suggested in /60/. This phase is needed to improve the accuracy. The problems that are addressed are totally different. In /59/, the main problem is how to design HIT with the already chosen pairs. The pairs beyond a threshold are simply chosen to ask the workers. In /60/, the main problem is, which pairs should be selected to improve the accuracy. The way to design the HITs is not addressed. It realizes the HITs in the simple pair-based approach. The combination of the two approaches may realize a better hybrid human-machine entity resolution solution. In addition, there are another research /63/, which focuses on the transitive relations of the pairs in order to crowdsource the minimum number of pairs to fulfill the whole join query.

4.2 Other Crowd-based Data Processing

In this section, further approaches on some other crowd-based data processing are briefly demonstrated.

Data Integration: Data integration means transforming the heterogeneous data sources into a consistent logical schema according to the specification of mappings among the data sources/22/. So far, a variety of algorithms or heuristics are studied to carry out the specification of such mappings. However, the results, which are obtained by automatically generating the mapping, are still far from perfect. Therefore, people can participate in the data integration procedure in order to get a better result. The new data integration is in general realized in the following way: The mappings are first carried out by the automatical algorithms or heuristics, then crowds can provide feed-back in a pay-as-you-go fashion on results of the automatical mappings. On the one hand, such feedback can refine the mappings and then improve the data integration quality; on the other hand, such feedback reduces the monetary cost of potentially employing expert staffs. Examples and implementation details can be found in /22 //64//65//66//67/.

Information Retrieval: Nowadays search engines are more and more intelligent. Sometimes people will be surprised by the close matched results. But they cannot handle everything. They are good at processing a single query, but weak at processing very complex queries, because they cannot understand the fact you are looking for. For instance, if a search request is “the ages of the mayors of all cities in Germany”, the search result by a search engine will not be satisfied. Since people can understand people well, crowdsourcing can be used to analyze very complex unstructured queries, and furthermore let people enjoy their search experience. In /68/, CrowdQ is suggested. It is a novel system that aiming to solve complex queries and get a corresponding result with the participation of the crowd. In addition, it can also store those kinds of complex queries. If there are similar queries later, the results can be utilized directly and automatically without each time’s participation of the crowd. In general CrowdQ uses a hybrid method to solve complex queries. More details can be found in /68/.

Iterative tasks: The above introduced crowd-based data processing in this section does not need the cooperation among the workers. They are relatively simple. There are some crowd-based data processing belongs to the iterative task, which need the cooperation among the workers in an iterative way. For example, the image or video description. The first phase is the initial phase, the workers are asked to describe an image or a video in short, simple sentences. Then the descriptions are collected preparing for the next phase. The next phase is the vote phase, the workers are asked to rate the descriptions and may provide their refinement based on the descriptions. In like manner, writing tasks, changing grammatical

tense or handwriting recognition can be performed on AMT. Turkit is a new toolkit for deploying iterative tasks to AMT, with a familiar imperative programming paradigm that effectively uses the workers as subroutines/69/. The concrete description can be found in /69/.

4.3 Comparison and Conclusion

In this chapter, the current research on a variety of crowd-based data processing is introduced. Wherein the database queries are focused on. In this section, they are compared with each other in various dimensions, and at last, the crowdsourcing issues that they have addressed are concluded.

Crowdsourcing Alone or Hybrid Approach: The crowd-based entity resolution, data integration, and information retrieval adopt the hybrid human-machine workflow. The reason that these three kinds of data processing does not adopt the crowdsourcing alone is the monetary cost that crowdsourcing requires. The hybrid processing flow becomes a compromise to achieve the expected quality with the acceptable monetary cost. Other crowd-based data processing is fulfilled via crowdsourcing only.

Cooperation of the workers: Most of the crowd-based data processing does not need the cooperation among the workers. Only the iterative tasks require the cooperation of workers. This kind of task requires usually repeated refinements based on the former result version, such as image description, since the single process cannot achieve the expected result.

Difficulties in Crowdsourcing: The focuses of the above introduced database queries vary. Table 4-1 summarizes the research focuses. The term "microtask" is used to replace HIT for the sake of universality. It means the smallest unit to be asked to workers. The overall data processing flow can be easily implemented with the help of crowdsourced databases. Therefore, the vast majority of the research in Table 4-1 aims at solving certain difficulties in crowdsourcing. Overall, difficulties in crowdsourcing are mainly reflected with the output quality, the monetary cost, and the relatively large latency. In the following, the difficulties and the corresponding measures that addressed in Chapter 3 and this chapter are concluded.

Database queries		Research Focus
The General Filter	filter in /39/	Overall data processing flow
	filter in /45/	Microtasks form Design
	filter in /46/	Response Aggregation
		Cost,latency and quality Optimization
Finding	finding in /47/	Cost,latency and quality Optimization
Sorting	Sorting in /48/	Microtasks form Design
		Overall data processing flow
Maximum	Maximum in /49/	Resonse aggregation
		Question Selection
Join	join in /48/	Microtasks form Design
		Overall data processing flow
	Entity Resolution in /53/	Microtask form Design
	Entity Resolution in /54/	Question Selection

Table 4-1: Research focuses summary

4.3.1 Quality Control

In order to tackle the difficulties of the output quality, three major segments in crowdsourcing should be paid attention to. They are the microtask design, the microtask assignment, and the response aggregation. The microtask as the communication media between requesters and workers is a precondition of the acceptable output quality. Only if the microtask is properly designed; the crowd is able to fulfill the microtasks with acceptable quality. Therefore, the microtask design should be considered. Crowds are not machines and have so many uncertainties that the answers returned from the crowd cannot directly be guaranteed. while machines can get the right result according to the preset algorithms, if there is no exception. Hence, the accuracy of the answer to a microtask cannot rely on one individual response from one worker. At most cases, one microtask is assigned to several workers. The number of workers that a microtask is assigned to should be considered to balance the accuracy and the monetary cost. After responses for the same microtask are returned from workers, the proper strategies to aggregate all responses into the final result are needed to be studied case by case. Next, the approaches aiming at solving the difficulties of output quality, which has presented in Chapter 3 and Chapter 4 are concluded.

Microtaks Types	Forms
Filter	Label-based Form
	Counting-based Form
Sorting	Comparison-bsed Form
	Rate-based Form
Join	Naïve-batching form
	Smart-batching form
Entity Resolution	Pair-based form
	Cluster-based form

Table 4-2: Microtask form designs summary

4.3.1.1 Microtask Design

The microtask design consists of two aspects: form design and question selection. The question selection focuses on the contents of a microtask.

Form Design: In Chapter 4, the microtask form designs are addressed four times. They are the filter, sorting, join and entity resolution form designs. Table 4-2 summarizes the microtasks forms. Among the four kinds of microtasks, sorting is a special case, since sorting is to order the objects, it is possible to rate each object and then compare the rate scores. The other three microtask designs are essentially the same. Their basic designs are to place each object group in one question. The object group for filter is one object, while the object group for join and entity resolution includes a pair of objects. Their upgraded designs are to place a cluster of objects in a microtask. Note that the comparison-based form of sorting amounts to the upgraded designs of the other three crowd-based data processing, that is cluster-based form. As a conclusion, for the tasks, which do not need to order the objects, there may be two form design methods: smallest-unit-based and cluster-based. The smallest unit may include only one object, such as in filter queries or a pair of objects, such as in join and entity resolution queries. Otherwise, there may be three form design methods overall: smallest-unit-based, cluster-based, and rate-based, wherein the smallest-unit-based or the cluster-based forms collectively refer to the comparison-based form.

Question Selection: In Chapter 4, the question selection for a microtask is addressed two times. The first time is the next vote problem in *maximum* queries, the second time is the

question selection in *entity resolution*. The next vote problem means which objects pairs should be selected based on the already obtained conclusion in order to get the most benefits. Question selection in *entity resolution* is to select the most valuable objects pairs based on pairs, which are obtained by applying a kind of entity resolution algorithm. These two question selection problems are very specific but we can still find the common way to cope with. At most cases, the goal of the question selection problem is to find the best questions in order to achieve the most benefits. The benefit of each question can be calculated according to a formula. The exact conclusion can be made in the way that calculating the benefits of all the possible questions and then selecting the corresponding questions with relatively larger benefits. Unfortunately, the complexity of this is usually #P-hard. Then the approximation algorithms may be developed case by case, which are just like in the *maximum* queries or in *entity resolution*.

4.3.1.2 Response Aggregation

In general, there are the following kinds of common strategies to aggregate the responses and for each strategy there are certain cases in which each strategy can be applied:

- **Half-voting:** Half-voting means that the final result is obtained only if more than half of the workers return a same response, then this response is the final result. Half-voting is more suitable for cases where the possible responses are very few, for the probability of no existing response that more than half workers agree with is higher, if there are more possible answers for a microtask.
- **Conservative strategy:** The conservative strategy means as long as a specific number of negative responses are obtained, the final result is negative. It is suitable for "yes" or "no" question, because for other cases the positive or negative of a response cannot directly be judged. Although the conservative strategy is better applied for the microtasks that contain "yes" or "no" questions, it still cannot be ignored, because "yes" or "no" questions are the very familiar form in crowdsourcing.
- **Probability strategy:** The probability strategy is that given a specific equation, calculating the probabilities that a response is correct for each response according to the equation, the responses with the highest probability is the final result. The most easy case for applying the probability strategy is the microtask containing only "yes"

or "no" questions as well, since the more responses are received, the more complicated the probability strategy is applied and "yes" or "no" questions only have two possible answers "yes" and "no". However, it can be applied to almost every case as long as the probability equation is provided properly.

- **Majority-voting:** Majority-voting means that the final result is a response, which is returned by more workers than other responses. Majority-voting is the most common strategy used in crowdsourcing. It can be simply applied, and an acceptable final result is received at most cases. However, if the responses are in numerical formats, it is better applying the average strategy to get the final result.
- **Average strategy:** The average strategy means the final result is equal to the average value of all responses. As mentioned in the majority-voting, the average strategy is more suitable for the microtasks whose responses are in numerical formats. For instance, the microtask is asking workers to rate something.
- **Duplicates elimination:** Duplicates elimination means the repetitive responses should be eliminated; then the remaining responses are the final result.
- **Identical entities elimination:** Identical entities elimination is similar to the former duplicates elimination. The difference is that identical entities elimination has a pre-processing step: entity resolution. After the entity resolution is made, the identical entities are eliminated although some of them have different original values.

Both duplicates elimination and identical entities elimination are better applied to the cases, which allow multiple answers, for instance, the microtasks with the purpose of opinions or ideas collection.

4.3.2 Monetary Cost, Latency, and Quality

In contrast to the traditional computer-based way to fulfill the specific tasks, additional factors need to be considered in crowdsourcing. The factors are mainly the monetary cost, the latency and the result quality.

Latency: Latency is very difficult to handle with, since the workers are free and out of control, they can accept a task but do not solve it immediately. Due to this reason, different microtasks are assigned on the crowdsourcing market in parallel to the largest extent. If the

number of needed microtasks for a task can be decided beforehand, then all microtasks are assigned at the same time. While for other tasks, the number of the target entities is predetermined but the number of needed microtasks is uncertain, in order to avoid the situation that more microtasks are assigned to workers than needed, assigning the microtasks should be in an iterative way, i.e. in the first phase, only a specific part of the microtasks is assigned. After the responses are returned, a specific number of the microtasks based on the responses from the former phase continue to be assigned until enough satisfied objects are found. In such way, the smallest latency can be achieved under the premise of no unneeded assignments. The corresponding algorithms of finding has already introduced in Section 4.1.1.

Monetary Cost and Quality: The inter-relationships between the monetary cost and the quality are two-fold. Sometimes the expected result quality of a task is not very high but the task has to be fulfilled in a very limited budget, while sometimes the result quality is very important and no budget limit exists. In the first case, strategies are designed to get the best quality under the budget. In the second case, strategies are designed to minimize the monetary cost to achieve the quality goal. The strategy here means the combination of the microtask assignment strategy and the response aggregation strategy. The naïve method for the first case is to calculate the quality scores of all strategies, whose monetary cost is under the budget limit, then choose the strategy with the highest quality score as the best strategy. For the second case the naïve method is similar, by calculating the needed monetary cost of all strategies, whose quality is better than the standard, and choosing the strategy with the minimum monetary cost as the best strategy. There are three algorithms in Section 4.1.1 to find the best strategy: naïve2, naïve3 and ladder shapes. Since the complexity of the naïve algorithms is often not acceptable, the ladder shapes algorithm is a novel but useful technique. However, this algorithm can only be applied to the microtasks, which contain only “yes” or “no” question. Efficient algorithms for other kinds of microtasks remain as future work. No current research is found which has already resolved this problem.

5. Optimization Techniques of Crowd-based Data Processing

The new business pattern "crowdsourcing" brings much benefit, and it is applied more and more widely. However, there are also some criticisms on it. The criticisms reflect mainly on the output quality, the required monetary cost and the relatively large latency. Therefore, a mass of research aiming at solving these problems was made to optimize the crowdsourcing services. In this chapter, innovative optimization techniques of crowd-based data processing are introduced.

5.1 CDAS[/70/]

CDAS (Crowdsourcing Data Analytics System) is a crowdsourcing data analytic system. It provides optimization techniques on all problems: the output quality, the monetary cost and the latency. In this section, its architecture will be first showed. After that, its two key models will be introduced.

5.1.1 CDAS Architecture

The core part of CDAS is a quality-sensitive answering model, which is designed to significantly improve the quality of query results and effectively reduce the processing cost at the same time. Figure 5-1 shows its architecture. As can be seen from Figure 5-1, there are three major components in CDAS: Job Manager, Program executor, and Crowdsourcing engine. The job manager accepts an analytic job and generates a processing plan, which shows how the other two components work interactively. The program executor collects the results of the crowdsourcing engine and may request to change the job schedule of the

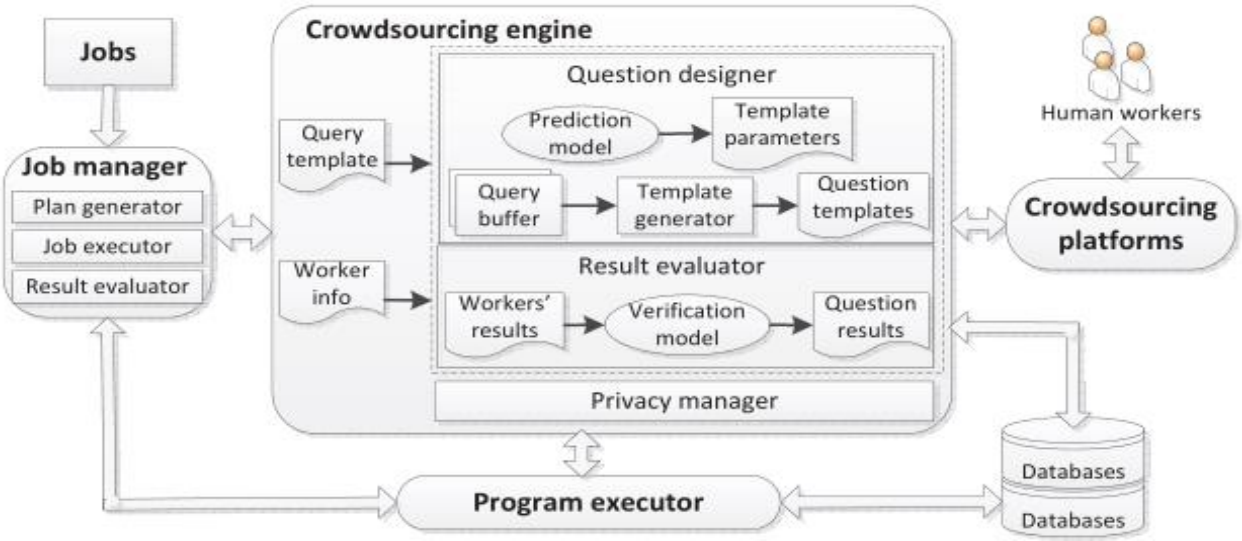

Figure 5-1: CDAS Architecture[/70/]

crowdsourcing engine. There are two subcomponents in the crowdsourcing engine: Question Designer and Result Evaluator. The question designer accepts the crowdsourcing tasks and divides the tasks into microtasks. The question templates are generated according to the microtasks, which follow the format defined by the crowdsourcing platform, such as AMT (Amazon Mechanical Turk). The question templates should be understood easily by the crowd. Next, the prediction model estimates a proper number of workers for each microtask by collecting the distribution of all workers' historical performance and then the microtasks are assigned correspondingly. After some responses are obtained, the result evaluator collects these responses and by the probability-based verification model generates the final result of each microtask and integrates the results into a complete result for the analytic job. Since a microtask is assigned to multiple workers, from the experience, the responses will be returned asynchronously, and there will be some workers that return the responses very quickly. Hence, the result evaluator collects these quick responses, and the probability-based verification model generates the current result of the microtask and integrates the results into a complete result for the analytic jobs. This technique is called online processing and it is extended with the techniques of data fusion in /71/ /72/ for the sake of calculating the results' confidence. If the confidence is too low, when more responses are submitted, the result will be refined. As long as the confidence is higher than the expected quality, the unfinished assignments will be ignored and the whole task is finished. In this way, the latency is reduced. Sometimes, the

result of an analytic task is sensitive to the public, thus the privacy manager is designed to cope with this situation. The privacy manager may adaptively change the format of the microtasks for the workers and may refuse some workers for the sensitive tasks.

A quality-sensitive answering model is the core part of CDAS. It can be divided into two sub-models: Prediction Model and Probability-based verification model. As mentioned in 4.3.1: Quality Control, three major segments of the output quality are the microtask design, the microtask assignment, and the response aggregation. The prediction model is a countermeasure to the difficulties of microtask assignment. The microtasks assignment problem refers mainly to the number of workers that a microtask is assigned. The quality of completed tasks will be low if too few workers are asked for one microtask. On the contrary, the needed monetary cost will be unacceptable. The prediction model estimates a proper number of workers for each microtask by collecting the distribution of all workers' historical performances. The probability-based verification model is a countermeasure to the difficulties of the response aggregation. In Section 4.3.1.2, several kinds of strategies have been concluded. The strategy used in the probability-based verification model is a kind of realization of the probability strategy in Section 4.3.1.2. In the next two subsections, these two models will be presented in detail.

5.1.2 Prediction Model

The prediction model in CDAS is designed to ensure high-quality result and reduce the monetary cost. It is highly related to how much the requesters must pay to a microtask. As an example, the economic model of AMT is used to explain the prediction model. The economic model of AMT has been introduced in Chapter 3. Here it is briefly introduced again. The requesters have to pay for the workers and the AMT at the same time for each HIT (Human Intelligent Task). Assume that each worker is paid a fixed amount of money m_c for each HIT and the requester pays AMT a fixed amount of money m_s per worker for each HIT. According to the prediction model in CDAS, n workers should work on one HIT, then the cost for each HIT amounts to $(m_c+m_x)n$. In the prediction model, the number of workers is correlated to the required accuracy C of the final result. The correlation between the number of workers n and the required accuracy C is defined using a function $n = g(C)$.

- **Vote-based Prediction Model:** The prediction model is vote-based. The voting strategy means that only the answers returned by more than $\lceil \frac{n}{2} \rceil$ workers are accepted (n is odd). This strategy ensures that no other answers receive more votes of being the correct answer. The goal of the prediction model is to determine the function *g()*.
  - **A Conservative Estimation:** In order to achieve this goal, the probability $P_{\frac{n}{2}}$ that at least $\lceil \frac{n}{2} \rceil$ workers provide the correct answer should be first calculated. It is calculated using the following equation:

$$P_{\frac{n}{2}} = \sum_{\mathbb{U} \subseteq U, |\mathbb{U}| \geq \frac{n}{2}} \left(\prod_{u_i \in \mathbb{U}} a_i \prod_{u_j \notin \mathbb{U}} (1 - a_j) \right)$$

U = { u_1, u_2, …, u_n }is the set of workers on the HIT, A={a_1, a_2, …, a_n} is the set of the workers accuracy. $\mathbb{U}$ is the set of workers which provides the correct answer and its size must not be smaller than $\lceil \frac{n}{2} \rceil$. This equation enumerates all the possibilities that the correct answer can be obtained by voting. Let μ be the mean value of the workers' accuracy. Then the expectation of the probability that at least $\lceil \frac{n}{2} \rceil$ workers provide the correct answer can be calculated using the following equation:

$$E\left[P_{\frac{n}{2}}\right] = \sum_{k=\lceil \frac{n}{2} \rceil}^{n} \binom{n}{k} \mu^k (1-\mu)^{n-k}$$

For a given task, $E\left[P_{\frac{n}{2}}\right]$ should be not smaller than the expected accuracy C, i.e. $E\left[P_{\frac{n}{2}}\right] \geq C$. According to Chernoff Bound and the theory that for any odd n, $\lfloor \frac{n}{2} \rfloor + 1 = \lceil \frac{n}{2} \rceil$, the lower bound of $E\left[P_{\frac{n}{2}}\right]$ is provided:

$$E\left[P_{\frac{n}{2}}\right] \geq 1 - e^{-2n\left(\mu - \frac{1}{2}\right)^2}$$

therefore,

$$1 - e^{-2n\left(\mu - \frac{1}{2}\right)^2} \geq C$$

Then,

$$n \geq \frac{-\ln(1-C)}{2(\mu - \frac{1}{2})^2}$$

Since n is an odd integer, the minimum number of n is

$$2\left\lfloor \frac{-\ln(1-C)}{4\left(\mu-\frac{1}{2}\right)^2} \right\rfloor + 1$$

The full proof for the equations can be found in /70/. This estimation ensures the accuracy can be obtained by assigning one HIT to *n* workers. It is called the conservative estimation, because the Chernoff Bound provides a tight estimation for a large enough number *n*. Thus, the above introduced estimation may generate too large *n*. To address this problem, the above minimum number of n is as the upper bound of n and optimization with binary search is suggested.

- **Optimization with Binary Search:** The binary search method can be used to find the proper number of workers n for one HIT. Since the Conservative Estimation provides a upper bound of n, the binary search begins to calculate the accuracy of one worker for one HIT, then repeats the calculation of the accuracy with two more worker each time. The binary search stops, as long as the accuracy is higher than C, or n reaches the upper bound $2\left\lfloor \frac{-\ln(1-C)}{4\left(\mu-\frac{1}{2}\right)^2} \right\rfloor + 1$. The calculation of the accuracy in each phase can be found in /70/.
- **Sampling-based Accuracy Estimation:** Both the conservative estimation and the optimization with binary search need the worker accuracy information. This accuracy may not be explicitly obtained because of the privacy policy or the policy of crowdsourcing service providers. Hence, a sampling-based estimation is designed to get the workers' accuracy. That is, for some given tasks, whose ground truth is known beforehand, random number of workers are assigned to the corresponding HITs. Then all responses from these workers are collected, and the corresponding accuracy is obtained and stored in CDAS.

5.1.3 Probability-based Verification Model

In the following, the probability-based verification model in CDAS will be introduced. It is a kind of realization of the probability strategy and presents a calculation method of the probability that a response is correct.

- **Probability-based Verification Model in CDAS:** The above introduced estimation of the number of the workers in the prediction model is based on the half-voting strategy. In fact, half-voting is not suitable for the cases with several possible responses, because the half agreement on a response does not always exist, and the voting strategy in the prediction model assumes that all workers have the same accuracy, which is not true since the accuracy actually varies and the worker with a higher accuracy can get more trust. Therefore, the verification model in CDAS is probability-based not vote-based. Although it is so, the vote-based prediction model is only for the sake of the calculation of the required worker number.

  The probability-based verification attempts to evaluate the answer confidence based on the worker confidence. Therefore, the worker confidence should be known. The worker confidence is calculated according to the workers' accuracy. As mentioned above, the workers' accuracy can be known by using the gold standard sampling method. Additionally, there are other algorithms that are suggested to evaluate the workers' performances, such as the algorithms in /73//74/. Let R denote the set of all possible responses, |R| = m and a_j denote the worker u_j's accuracy. Then the worker confidence is:

$$c_j = \ln \frac{a_j}{\frac{1-a_j}{m-1}} = \ln \frac{(m-1)a_j}{1-a_j} = \ln(m-1) + \ln \frac{a_j}{1-a_j}$$

  $\frac{1-a_j}{m-1}$ is the probability that the worker u_j provides each incorrect response. Next, the calculation of each response's accuracy is derived step by step. Based on Bayesian analysis, the probability of a specific response $\overline{r} \in R$ being the correct response given the distribution Ω of the responses in R is calculated as follows:

$$\mathrm{P}(\overline{r}|\Omega) = \frac{\mathrm{P}(\Omega|\overline{r})\mathrm{P}(\overline{r})}{P(\Omega)} = \frac{\mathrm{P}(\Omega|\overline{r})\mathrm{P}(\overline{r})}{\sum_{r_i \in R} P(\Omega|r_i)\, p(r_i)}$$

  Each response r_i appears with an equal probability $\frac{1}{m}$, therefore,

$$\mathrm{P}(\overline{r}|\Omega) = \frac{\mathrm{P}(\Omega|\overline{r})}{\sum_{r_i \in R} P(\Omega|r_i)}$$

  Wherein,

Movie Title	Green Latern				
Tweet	*Oh. My. GOD. "Green Lantern" movie is terrible. Like, "Lost In Space" movie terrible.*				
Worker ID	w_1	w_2	w_3	w_4	w_5
Accuracy	0.54	0.31	0.49	0.73	0.46
Answer	pos	pos	neu	neg	pos

Figure 5-2: An example of workers' responses and their accuracy[70]

$$\mathrm{P}(\Omega|\,\overline{r}) = \prod_{f(u_j)=\overline{r}} a_j \prod_{f(u_j)\neq\overline{r}} \frac{1-a_j}{m-1}$$

$$\mathrm{P}(\,\overline{r}|\Omega) = \frac{\prod_{f(u_j)=\overline{r}} a_j \prod_{f(u_j)\neq\overline{r}} \frac{1-a_j}{m-1}}{\sum_{r_i\in R}(\prod_{f(u_j)=r_i} a_j \prod_{f(u_j)\neq r_i} \frac{1-a_j}{m-1})} = \frac{\prod_{f(u_j)=\overline{r}} \frac{(m-1)a_j}{1-a_j}}{\sum_{r_i\in R}(\prod_{f(u_j)=r_i} \frac{(m-1)a_j}{1-a_j})}$$

The answer confidence $\mathrm{P}(\overline{r})$ is equal to the probability of $\overline{r}$ being the correct answer. Therefore, the answer confidence $\mathrm{P}(\overline{r})$ can be expressed using the worker confidence cj:

$$\mathrm{P}(\overline{r}) = \mathrm{P}(\,\overline{r}|\Omega) = \frac{e^{\sum_{f(u_j)=\overline{r}} c_j}}{\sum_{r_i\in R}(e^{\sum_{f(u_j)=r_i} c_j})}$$

For instance, a microtask asks five workers to determine a twitter is positive, negative, or neutral. The received responses from the five workers and their accuracy are showed in Figure 5-2. As can be seen from it, three workers consider the twitter as positive, only one worker consider the twitter as negative and one worker consider the twitter as neutral. However, the worker, which considers the twitter as negative, has a high accuracy. Then, according to the calculation approach in the probability-based verification model, the probabilities that one answer is correct can be calculated as follows:

	pos	neu	neg	Answer
Half-Voting	3	1	1	pos
Majority-Voting	3	1	1	pos
Verification	0.329	0.176	0.495	neg

Figure 5-3: Results of different strategies[/70/]

$$P(\text{pos}|(\text{pos},\text{neg},\text{neu})) = \frac{\frac{2*0.54}{0.46}*\frac{2*0.31}{0.69}*\frac{2*0.46}{0.54}}{\frac{2*0.54}{0.46}*\frac{2*0.31}{0.69}*\frac{2*0.46}{0.54}+\frac{2*0.49}{0.51}+\frac{2*0.73}{0.27}} = 0.329$$

$$P(\text{neu}|(\text{pos},\text{neg},\text{neu})) = \frac{\frac{2*0.49}{0.51}}{\frac{2*0.54}{0.46}*\frac{2*0.31}{0.69}*\frac{2*0.46}{0.54}+\frac{2*0.49}{0.51}+\frac{2*0.73}{0.27}} = 0.176$$

$$P(\text{neg}|(\text{pos},\text{neg},\text{neu})) == \frac{\frac{2*0.73}{0.27}}{\frac{2*0.54}{0.46}*\frac{2*0.31}{0.69}*\frac{2*0.46}{0.54}+\frac{2*0.49}{0.51}+\frac{2*0.73}{0.27}} = 0.495$$

Therefore, the negative is as the correct result. Figure 5-3 shows the corresponding results of three different strategies: Half-voting, Majority-voting, Probability-based verification. Both half-voting and majority-voting consider positive as the correct answer. However, the probability-based verification model chose negative as the correct answer, because the worker answering Negative has a much higher accuracy. Since this twitter is negative in fact, as a conclusion, the probability-based verification model is more accurate than the simple majority-voting and half-voting.

5.2 HumanGS[/75/]

HumanGS (Human-assisted Graph Search) can be used to solve the question selection problem. The graph in it is a DAG (Directed Acyclic Graph) with some (unknown) target nodes. Then the question selection problem is transformed into finding the target node that is the best choice to ask a question on it. This approach can be adopted by the following five applications:

- **Image Categorization:** Given an image, to which category does it belong? The categories are the nodes in the DAG.
- **Manual Curation:** This application is isomorphic to the image categorization. It means that the crowds insert new concepts, topics or items to the best place in the whole architecture. For instance, editing in Wikipedia. The places are the nodes in the DAG.

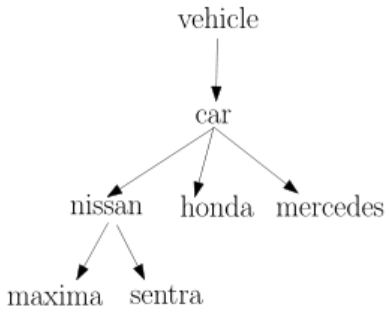

Figure 5-4: DAG in Image Categorization[75]

- **Filter Synthesis:** This application is to ask the crowds to identify the proper filters to some ad-hoc analysis. The candidate filters are the nodes in the DAG.
- **Interactive Search:** In interactive search, the search engine asks the user questions that help isolate the concepts that best meet the user's information need. The concepts are the nodes in DAG.
- **Debugging of Workflows:** When there is an erroneous result in the output of a workflow, people have to find the earliest step leading to the error. The steps are the nodes in the DAG.

Note the last two applications belong to the human computation, not crowdsourcing. In order to indicate how these applications are considered as the HumanGS problem, an example of the image categorization is taken. Given a car image, the categories may be vehicle, car, Nissan, Honda, Mercedes, Maxima, Sentra. The corresponding DAG is showed in Figure 5-4. The target node is the most suitable category of the image. Therefore, the best question to be asked to crowds is transformed into finding the target node in the DAG. The HumanGS is nontrivial because of its following properties: First, the answers of such search questions may be correlated. For example, if the answer at a node is "yes", then the answers of its ancestor nodes will be "yes" as well. If the answer at a node is "no", then the answers of its child nodes will be "no" as well. Second, the location of a node determines the amount of information that can be obtained from the corresponding search questions at this node. Asking search questions at nodes close to leaves is more likely to get a "no" answer, while at nodes close to the root it is more likely to get a "yes". Asking search questions at "middle" nodes has a larger possibility to get more information. From this point, the HumanGS

dimensions / applications	Single/Multi	Bounded/Unlimited	DAG/Downward/Upward
Image Categorization	Single	Bounded	Downward
Manual Curation	Single	Unlimited	Downward
Filter Synthesis	Multi	Bounded	DAG
Interactive Search	Multi	Bounded	Upward
Debugging of Workflows	Multi	Unlimited	DAG

Table 5-1: Classification based on the three dimensions /75/

is similar to the 20 questions game[1], too specific or too general questions make little sense.

The above introduced five applications are classified according to three dimensions: single or multi, bounded or unlimited, DAG or upward-Forest or downward-Forest.

Single/multi means that it needs a single target node or multiple target nodes. Bounded/unlimited means that whether there is a budget to constrain the numbers of the questions. DAG/downward-forest/upward-forest means that the form of the graph, it can be a general DAG or a downward-forest (the root at the top) or a upward-forest (the root at the bottom).

In /75/, the algorithms for all dimensions are considered. However, Image Categorization is the most common case in crowdsourcing; the single/bounded/downward-forest algorithm is briefly introduced here. The single/bounded/downward-forest problem can be reduced to the single/bounded/downward-tree problem. Then the single/bounded/downward-tree problem can be considered equal to the partition problem. The proof of the two transformations can be found in /75/. At last, the single bounded/downward-forest algorithm is a dynamic programming algorithm introduced in /76/ with complexity O(nlogn).

[1] 20 questions is a game between two players. One player thinks an object, person or place inwardly and the other player has to guess it by asking the other player up to 20 YES/NO questions.

5.3 Worker Activities Supervision/77/

Workers' accuracy is non-trivial in crowdsourcing, since it closely relates to the microtask assignment problem and the response aggregation problem. In order to evaluate a worker's accuracy, the most common way is to record his/her historical responses, compare the responses to the standard result, and then conclude the worker's accuracy. The standard result can be obtained by the way that assigning a same microtask to multiple workers and applying a proper responses aggregation strategy or by the gold standard approach. A novel alternative and complementary technique for evaluating a worker's accuracy is presented in /77/. It examines the way a worker works, rather than the responses a worker submits, since the work process can reflect the efforts or behaviors of a worker. For instance, two different workers are working on a same microtask with tagging an image. The lazy worker may directly scroll to the text fields, type the first tag after accepting the microtask and then tab to the second field and type the second tag. The conscientious worker may observe the first image for a while after accepting the microtask, then scroll to the first field, type the first tag and then scroll back to observe the second image and so on. This novel technique with implicit behavioral measure is called *task fingerprinting*. It captures the process that the worker use to complete a microtask and then logs what the workers do when the worker is working on a microtask. A quantitative description of their process is developed to enable the comparison among workers, the evaluation of the responses and the further prediction of the accuracy of workers working on the specific microtasks. The raw form of a quantitative description is sequential logs containing different behaviors from workers and their corresponding timestamps. Various valuable information is encoded in the sequences, such as the order of operations, the time interval between the actions and so on. The raw form can be refined to collect some statistical information, such as the number of different actions, which can be used to compare workers. At last, the data is trained with the machine learning approach and conclusions are obtained. Since assigning a same microtask to multiple workers need more monetary cost, gold standard sampling needs the standard result on the sampling and sometimes is not available, *task fingerprinting* is a valuable alternative and complementary technique. More details and the performance evaluation can be found in /77/.

5.4 Monetary Cost, Latency, and Quality

The basic inter-relationships between monetary cost and quality has been concluded in 4.3.2. In this section, the vision for the cost and quality trade-off in crowdsourcing environments that is suggested in /78/ is introduced. /78/ takes the sorting algorithms as examples to indicate that the performance evaluation of the algorithms in crowdsourcing environments is not the same as in the traditional way. Traditionally, QuickSort has a better performance than BubbleSort, for QuickSort needs a lower number of comparisons. While in crowdsourcing environments, since workers may make mistakes, QuickSort does not always perform better than BubbleSort. According to the experiment results, QuickSort generates good quality results in low noise environments while BubbleSort can be more adequate in environments where workers often make mistakes. The reason is BubbleSort has a natural robustness to the mistakes. At each run, BubbleSort provides an ingrained chance to the right sorting and the information is leveraged over multiple runs. However, BubbleSort needs a larger budget than QuickSort. Therefore, if the result quality is more important than the monetary cost, BubbleSort is advised in crowdsourcing environments. Conversely, QuickSort is suggested. /78/ expects to get an abstraction from sorting algorithms to build an adaptive model with respect to quality, budget and worker error rate in crowdsourcing environment for the sake of better estimation. A simple two-step estimation process is suggested: Initial Estimate and Adjust Estimate. The Initial Estimate calculates the complexity of the algorithm in crowdsourcing environments according to its properties. Then the Adjust Estimate adjusts the estimate to meet the required budget parameter and the worker accuracy parameter. The process is only sketched roughly; more specific implementation and design details remain as future work.

5.5 Indexing in Crowdsourced Databases[/79/]

A database index is an auxiliary data structure that allow for quicker retrieval of data[/80/]. Tree indexes are the most commonly used data structures for database indexes. Although the retrieval performance in databases is improved by indexes, additional index updates are needed when tables are updated and additional space is needed to store the indexes as well. Hence, indexes should be carefully designed to pay a minimum price and obtain the most

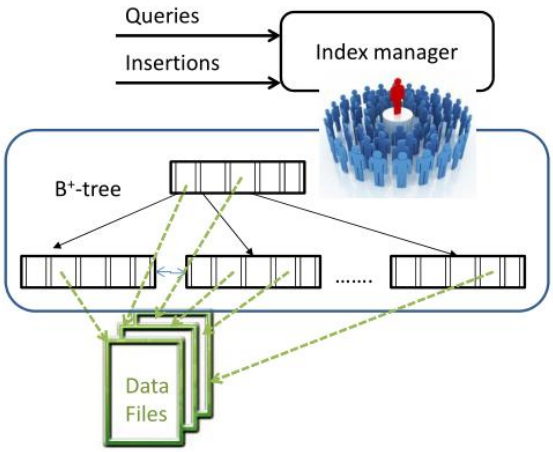

Figure 5-5: Palm-tree index model[79]

benefits. In the following, the investigation of proper indexes and algorithms for the index operations in crowdsourced databases is presented. Let *S* be a set of items ($|S| = n$) and q be the query item.

Index Model: The tree-based index in crowdsourced databases is called the palm-tree index. The reason for this name is that the real palm tree needs also to be pollened manually in order to produce fruits. It consists of two components: the index manager and a B^+-tree. Figure 5-5 depicts its structure.

- **B+-tree Component:** The palm-tree index is built on the top of a B^+-tree. Traditional B+-tree splitting or merging algorithms are used to insert into the B+-tree or delete from it. The items in S are first ordered, then the palm-tree index is built based on the ordered items. There are two types of keys: quantitative key and qualitative key. A quantitative key is a key with both the subjective property and an assessed value. For instance, if the set contains images of damage cars, the quantitative key for a car may be its image and the cost to repair the car. A qualitative key is a key only with the subjective property. For instance, the set contains images of butterflies, then the qualitative key of a butterfly is the beauty level that people consider for it. For both types of keys, crowd-search is used to search the B^+-tree for a query, i.e. using crowdsourcing to process a query. The set that contains items with quantitative keys can be easily ordered based on the assessed value and then the index is bulkloaded. The set that contains items with qualitative keys has to be ordered using human-based

Operation	Quantitative keys	Qualitative keys
Query	Crowd-search	Crowd-search
Insert	B^+-tree insert	Crowd-search then B^+-tree insert
Build	Bulk loading or Successive inserts	Successive inserts
Delete	Query then B^+-tree delete	Query then B^+-tree delete
Update	Delete then insert	Delete then insert

Figure 5-6: Palm-tree index operations/79/

comparison and successive insertions. Each node in the B^+-tree has *n* ordered keys and stands for a microtask. *n* should be chosen carefully, too large *n* will confuse the workers and let them easier to make mistakes; too small *n* will lead to the waste of monetary cost.

- **Index Manager:** The index manager is responsible for the palm-tree index construction and query processing within the palm-tree. It generates the microtasks according to the task from the requester, assigns the microtasks to workers and aggregates the responses. Majority-voting is used to aggregate the responses.

Palm-tree index operations: Figure 5-6 shows how the operations are performed in the palm-tree index. As described above, the index is built by easily bulk loading or successive inserts for the quantitative keys, while for qualitative keys only successive inserts with the human comparison beforehand is used to build the index. For both keys, the query is performed by crowd-search. Figure 5-7 is a sample microtask, which asks workers to estimate the proximal repair cost. The B^+-tree insert is still adopted in the palm-tree index, but for the qualitative keys, the crowd-search is first used to determine the subject property. Delete is performed by first performing the query operation then performing the original B^+-tree delete. Update is performed by first performing the delete operation then performing the insert operation.

Algorithms for Crowd-search: In /79/, three algorithms are developed for the crowd-search. They will be explained in the following.

- **Leaf-Only Aggregation Strategy:** This strategy asks each worker to perform the search from the root to the leaf to find the best match to the query item. Then the leaves, which submitted by different workers, are collected. Then the final result is obtained according to majority-voting.

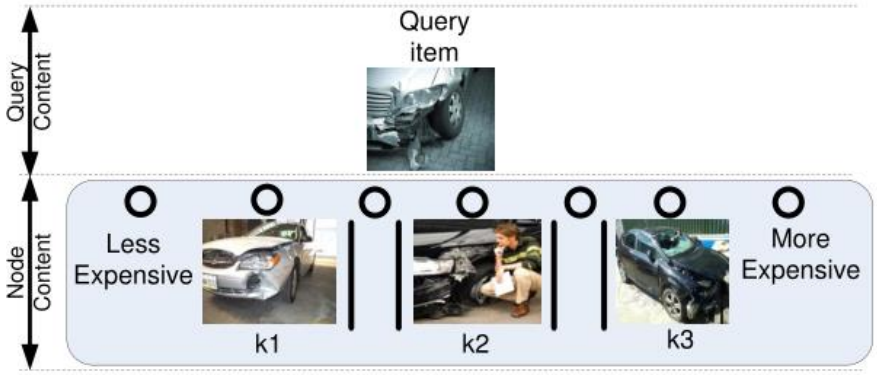

Figure 5-7: Sample microtask to estimate the proximal car repair cost/79/

All-levels Aggregation Strategy: In contrast to the leaf-only aggregation strategy, each worker is only responsible for performing the search from the given nodes to its child node. The majority voting is used to aggregate the nodes in each level, which are submitted by different workers.

- **All-levels Aggregation Strategy with Backtracking:** This strategy is changed by extending the all-levels aggregation strategy with backtracking. Backtracking means that if the results returned from the workers show the query item does not belong to this node, i.e. the choice made in the parent level is wrong, the search should be back to the parent nodes level and choose another most possible node to continue. For the sake of selecting the most possible node after backtracking, a priority queue is maintained. The queue contains all the unexplored nodes, the nodes are ordered according to their height and the obtained votes.

5.6 Query-driven Schema Expansion/81/

In most cases, each time a crowdsourcing microtask is performed, requesters have to pay money to workers, even sometimes still need to pay money to the marketplace. For the sake of saving money, a microtask cannot be assigned to too many workers to improve the result quality. In addition, requesters suffer from the large latency of crowdsourcing. Hence, /81/ suggests to exploit resources from the social web to perform specific kinds of tasks instead of crowdsourcing or as the assistant to crowdsourcing to obtain a better result quality. The tasks may be rating movies, restaurants, products or something else, since there are numerous user rating websites in the Internet. The information in those websites is mainly visible to the

public and may be provided by tens of thousands of people. In this way, crowdsourcing issues mentioned above can be improved. In order to exploit the user ratings *perceptual spaces* are built, which contain highly-compressed representations of opinions, impressions, and perceptions from a large amount of people. The expert crowdsourcing trains few samples, then the missing information can be extracted from the perceptual spaces automatically with high quality and at little cost. It is a pity that this innovative idea can only be applied to rating tasks. However, the user ratings play a very important role in nowadays society and this innovative idea leads to better crowdsourcing and brings much benefit.

5.7 Conclusion

This Chapter provided an overview of several innovative optimization techniques. CDAS is a crowdsourcing data analytic system. Its core part, a quality-sensitive answering model, which can be divided into two sub-models: Prediction Model and Probability-based verification model. The Prediction Model optimizes the microtask assignment problems, and the Probability-based verification model provides an optimization of the response aggregation problems. Besides, its online processing technique reduces the latency for a query. By applying these optimization techniques, CDAS reaches the goals of minimizing monetary cost to reach the expected output accuracy with an acceptable latency. HumanGS optimizes the question selection problem from a very different perspective. It transforms the crowdsourcing tasks, such as image categorization, into a graph search problem. Each nodes represents a possible question in a microtask. The target node, that is the question that should be asked in a microtask, can be found by applying existing approaches to solve the graph search problem. In order to evaluate a worker's accuracy, the most common way is to record workers' historical responses, compare the responses to the standard result, and then conclude the worker's accuracy. Section 5.3 proposes a novel alternative and complementary technique for evaluating a worker's accuracy by examining the way a worker works, rather than the responses a worker submits. This optimization technique is called task fingerprinting. Since assigning a same microtask to multiple workers need more monetary cost, gold standard sampling needs the standard result on the sampling and sometimes is not available, task fingerprinting is a valuable alternative and complementary technique to evaluate workers' accuracy. Section 5.4 introduces a new vision for the cost and quality trade-off in

crowdsourcing environments. It proves that in crowdsourcing environments the performance of existing algorithms may be skewed. Therefore, choosing a suitable algorithm for a problem in crowdsourcing environments should be carefully considered, rather than simply according to the experience in traditional environments. All above mentioned techniques aim at solving the difficulties in crowdsourcing in Section 4.3. While indexing in crowdsourced databases and query-driven schema expansion aim at solving other issues. Indexing in crowdsourced databases optimizes the performance of data retrieval in crowdsourced databases. Query-driven schema expansion proposes to fulfill some crowdsourcing tasks by exploiting the resources in the social web. In this way, certain tasks do not need to be performed each time per crowdsourcing. The monetary cost and the latency is improved.

6. Conclusion

Since crowdsourcing was first coined in 2006, it has developed rapidly in recent years and attracts a variety of research. This thesis focuses on the crowd data sourcing, which utilizes the power of crowdsourcing to process data. The current research is overviewed and classified in this thesis. The current research on crowd data sourcing includes two major aspects: crowdsourced databases and crowd-based data processing. Crowdsourced databases extend conventional databases to support more kinds of queries by means of the power of people. These crowdsourced databases associated with certain crowdsourcing marketplaces, such as AMT (Amazon Mechanical Turk), CrowdFlower and so on, to attract crowds to work for them. In order to enable conventional databases to support queries processed by crowdsourcing, a series of adjustments on the architecture, data model, query language, query processing are made. In Chapter 3, three most popular crowdsourced databases CrowdDB, Deco, and Qurk are demonstrated and compared with each other. Next, in Chapter 4 the crowd-based data processing is presented. Crowdsourced database queries are focused on, and other crowd-based data processing is only introduced briefly. The current research studies different areas of the crowd-based data processing. Some of them investigate the overall implementation flow, others aims at solving the numerous difficulties in crowd-based data processing. The difficulties are mainly reflected in how to improve the task result quality and at the same time reduce the monetary cost and latency. Although crowdsourcing brings much benefit and is applied more and more widely. However, there are still some criticisms on it. These criticisms in turn attract a mass of research to optimize crowdsourcing. These optimization techniques of crowd-based data processing are demonstrated in Chapter 5.

Although crowdsourcing faces such challenges and so far, the challenges are not solved completely and perfectly, crowdsourcing has already brought huge benefit for our society and

will continue to contribute itself more in the future beyond all doubt and the difficulties in it will be further solved.

Bibliography

/1/ Jeff Barr and Luis Felipe Cabrera. AI Gets a Brain, ACM Queue, pages 24-29,May 2006.

/2/ Deutch D, Milo T. Mob data sourcing. Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, pages 581-584, 2012.

/3/ Safire W. On Language. New York Times Magazine, February 5, 2009.

/4/ Howe J. The rise of crowdsourcing. Wired Magazine, 2006.

/5/ Estellés-Arolas EG-L-d-G, Fernando. Towards an Integrated Crowdsourcing Definition. Journal of Information Science 38, pages 189–200,2012.

/6/ "Wikipedia", http:/www.wikipedia.org/, Retrieved 2013-11-06.

/7/ "Wikipedia", http:/meta.wikimedia.org, Retrieved 2013-11-06.

/8/ "Five companies aspiring to change the world", http://www.36kr.com/p/96077.html, Retrieved 2013-12-15.

/9/ "MTurk CENSUS", https:/forums.aws.amazon.com/thread.jspa?threadID=58891#,Retrieved 2013-11-08.

/10/ "IMDb Database Statistics", http:/www.imdb.com/stats, Retrieved 2013-11-08.

/11/ Luis von Ahn, "Games With A Purpose". IEEE Computer Magazine: 96–98, June 2006.

/12/ "Players with a purpose", http:/www.crowdsourcing.org/document/players-with-a-purpose/6750, Retrieved 2013-11-08.

/13/ "Tagasauris", http://en.wikipedia.org/wiki/Tagasauris, Retrieved 2013-12-15.

/14/ "99designs", https:/www.facebook.com/99designs/info, Retrieved 2013-11-08.

/15/ Jeff Bullas, "35 Mind Numbing YouTube Facts, Figures and Statistics", http://www.jeffbullas.com/2012/05/23/35-mind-numbing-youtube-facts-figures-and-statistics-infographic/#zjzA6gcOkQZB3IvX.99, Retrieved 2013-11-08.

/16/ Alice R. Burks, The First Electronic Computer. *University of Michigan Press*, Ann Arbor, MI, 1989.

/17/ Reimer, Jeremy (2 December 2012). "Personal Computer Market Share: 1975–2004". http:/jeremyreimer.com/m-item.lsp?i=137, Retrieved 2013-10-25.

/18/ "object recognition", http:/www.mathworks.com/discovery/object-recognition.html, Retrieved 2013-10-29.

/19/ Hanna Köpcke, AT, Erhard Rahm. Evaluation of entity resolution approaches on real-world match problems. *Proceedings of the VLDB Endowment*, pages 484-493, 2010.

/20/ Jiannan Wang, Tim Kraska, Michael J. Franklin, Jianhua Feng, CrowdER: crowdsourcing entity resolution, *Proceedings of the VLDB Endowment*, pages 1483-1494, July 2012.

/21/ Maurizio Lenzerini, Data integration: a theoretical perspective, *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, June 03-05, 2002

/22/ Fernando Osorno-Gutierrez NWP, Alvaro A, A.Fernandes. Crowdsourcing Feedback for Pay-As-You-Go Data Integration. *Proceedings of the First VLDB Workshop on Databases and Crowdsourcing*, 2013.

/23/ Gianluca Demartini, Beth Trushkowsky, Tim Kraska, and Michael J. Franklin. CrowdQ: Crowdsourced query understanding. *CIDR*, 2013.

/24/ James Surowiecki, The Wisdom of Crowds, *published by Doubleday and Anchor*, 2005.

/25/ Steven Komarov KR, Krzysztof Z. Gajos. Crowdsourcing Performance Evaluations of User Interfaces. CHI, pages 207-216,2013.

/26/ Anhai Doan RR, Alon Y. Halevy. Crowdsourcing systems on the worldwide web. *Communications of the ACM*, pages 86-96, 2011.

/27/ Geiger, David; Seedorf, Stefan; Schulze, Thimo; Nickerson, Robert C.; and Schader, Martin, "Managing the Crowd: Towards a Taxonomy of Crowdsourcing Processes", *Proceedings of the Seventeenth Americas Conference on Information Systems*, 2011.

/28/ Corney, J. R., Torres-Sanchez, C., Jagadeesan, A. P., and Regli, W. C. Outsourcing labour to the cloud, *International Journal of Innovation and Sustainable Development*, pages 294-313, 2009.

/29/ Malone, T. W., Laubacher, R., and Dellarocas, C. N. The collective intelligence genome, *MIT Sloan Management Review*, pages 21-31, 2010.

/30/ Piller, F. T., Ihl, C., and Vossen, A. A Typology of Customer Co-Creation in the Innovation Process, *SSRN eLibrary*, 2010.

/31/ Rouse, A. C. A preliminary taxonomy of crowdsourcing, *Proceeding ACIS 2010*, 2010.

/32/ Schenk, E., and Guittard, C. Towards a characterization of crowdsourcing practices, *Journal of Innovation Economics*, 2011.

/33/ Zwass, V. Co-Creation: Toward a Taxonomy and an Integrated Research Perspective, *International Journal of Electronic Commerce*, pages 11-48, 2010.

/34/ Allan Leake, "database", http:/searchsqlserver.techtarget.com/definition/database, Retrieved 2013-11-01.

/35/ Date, C. J, An Introduction to Database Systems, Fifth Edition. Addison Wesley, 2003.

/36/ Siba Mohammad, A Survey and Classification of Data Management Research Approaches in the Cloud, *Master Thesis*, 2011.

/37/ Avi Silberschatz, Henry F. Korth, S. Sudarshan; Database System Concepts, *published by McGraw-Hill*, 2010.

/38/ Nguyen Anh, "Query Processing and Optimization", http:/cnx.org/content/m28213/latest/, Retrieved 2013-11-01.

/39/ Aniket Kittur , Boris Smus, Susheel Khamkar and Robert E. Kraut, CrowdForge: Crowdsourcing Complex Work, UIST'11, Pages 43-52, 2011.

/40/ Patrick Minder and Abraham Bernstein, CrowdLang: A Programming Language for the Systematic Exploration of Human Computation Systems, *SocInfo*, pages 124-137, 2012.

/41/ A. Morishima, N. Shinagawa, T. Mitsuishi, H. Aoki and S. Fukusumi. CyLog/Crowd4U: A Declarative Platform for Complex Data-centric Crowdsourcing. *PVLDB*, pages 1918-1921, 2012.

/42/ Kenji Gonnokami, Atsuyuki Morishima, and Hiroyuki Kitagawa, Condition-Task-Store: A Declarative Abstraction for Microtask-based Complex Crowdsourcing. *Proceedings of the First VLDB Workshop on Databases and Crowdsourcing*, 2013.

/43/ Michael J. Franklin , Donald Kossmann , Tim Kraska , Sukriti Ramesh , Reynold Xin, CrowdDB: answering queries with crowdsourcing, *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 61-72, 2011.

/44 / A. Feng, M. J. Franklin, D. Kossmann, T. Kraska, S. Madden, S. Ramesh, A. Wang, and R. Xin. CrowdDB: Query Processing with the VLDB Crowd. *PVLDB*, 4(11): pages 1387--1390, 2011.

/45/ Adam Marcus , Eugene Wu , David R. Karger , Samuel Madden , Robert C. Miller, Demonstration of Qurk: a query processor for human operators, *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 1315-1317, 2011.

/46/ Adam Marcus, Eugene Wu, David R. Karger, Samuel Madden, Robert C. Miller, "Crowdsourced Databases: Query Processing with People." *5th Biennial Conference on Innovative Data Systems Research (CIDR '11)*, pages 211-214, 2011.

/47/ Aditya. Parameswaran, Hyunjung. Park, Hector. Garcia-Molina, Neoklis. Polyzotis, and Jennifer. Widom. Deco: Declarative crowdsourcing. *Technical report*, Stanford University.

/48/ Hyunjung Park , Hector Garcia-Molina , Richard Pang , Neoklis Polyzotis , Aditya Parameswaran , Jennifer Widom, Deco: a system for declarative crowdsourcing, *Proceedings of the VLDB Endowment*, v.5 n.12, pages 1990-1993, August 2012.

/49/ Aditya Ganesh Parameswaran , Hyunjung Park , Hector Garcia-Molina , Neoklis Polyzotis , Jennifer Widom, Deco: declarative crowdsourcing, *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 1203-1212, 2012.

/50/ Hyunjung Park , Richard Pang , Aditya Parameswaran , Hector Garcia-Molina , Neoklis Polyzotis , Jennifer Widom, An overview of the deco system: data model and query language; query processing and optimization, *ACM SIGMOD Record*, v.41 n.4, pages 22-27, 2012

/51/ Adam Marcus , David Karger , Samuel Madden , Robert Miller , Sewoong Oh, Counting with the crowd, *Proceedings of the VLDB Endowment*, v.6 n.2, pages 109-120, 2012.

/52/ Aditya G. Parameswaran , Hector Garcia-Molina , Hyunjung Park , Neoklis Polyzotis , Aditya Ramesh , Jennifer Widom, CrowdScreen: algorithms for filtering data with humans, *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 361-372, 2012

/53/ A. D. Sarma, A. Parameswaran, H. Garcia-Molina, and A. Halevy. Finding with the crowd. *In Infolab Technical Report*, 2012

/54/ Adam Marcus , Eugene Wu , David Karger , Samuel Madden , Robert Miller, Human-powered sorts and joins, *Proceedings of the VLDB Endowment*, v.5 n.1, pages 13-24, September 2011.

/55/ Stephen Guo , Aditya Parameswaran , Hector Garcia-Molina, So who won?: dynamic max discovery with the crowd, *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 385-396, 2012.

/56/ C.J. Date, An Introduction to Database Systems, Addison-Wesley Publishing Company, the fifth edition, 1990.

/57/ Paolo Atzeni, Stefano Ceri, Stefano Paaboschi and Riccardo Torlone, Database Systems: Concepts, Languages and Architectures, The McGraw-Hill Company, 1999.

/58/ Tomomi Mitsuishi, Atsuyuki Morishima, Norihide Shinagawa, Hideto Aoki, Efficient evaluation of human-powered joins with crowdsourced join pre-filters, *ICUIMC '13*, 2013.

/59/ J. Wang, T. Kraska, M. J. Franklin, and J. Feng. CrowdER: Crowdsourcing Entity Resolution. *PVLDB, Vol. 5, No. 11*, pages1483-1494, 2012.

/60/ Whang, Steven Euijong and Lofgren, Peter and Garcia-Molina, Hector (2013) Question Selection for Crowd Entity Resolution. *PVLDB* , 2013.

/61/ J. M. Valério and D. Carvalho. Exact solution of cutting stock problems using column generation and branch-and-bound. *International Transactions in Operational Research,5(1):* pages 35–44, 1998.

/62/ Whang, Steven Euijong and Lofgren, Peter and Garcia-Molina, Hector, Question Selection for Crowd Entity Resolution, *Technical Report. Stanford InfoLab*, 2012.

/63/ Jiannan Wang, Guoliang Li, Tim Kraska, Michael J. Franklin, Jianhua Feng, Leveraging Transitive Relations for Crowdsourced Joins, *SIGMOD*, 2013.

/64/ S. R. Jeery, M. J. Franklin, and A. Y. Halevy. Pay-as-you-go user feedback for dataspace systems, *SIGMOD* , 2008.

/65/ R. McCann, W. Shen, and A. Doan. Matching schemas in online communities: A web 2.0 approach. *ICDE*, 2008.

/66/ P. P. Talukdar, Z. G. Ives, and F. Pereira. Automatically incorporating new sources in keyword search-based data integration. *SIGMOD*, 2010.

/67/ K. Belhajjame, N. Paton, A. Fernandes, C. Hedeler, and S. Embury. User feedback as a first class citizen in information integration systems. *CIDR*, pages 175–183, 2011

/68/ Demartini G, Trushkowsky B, Kraska T and Franklin M, CrowdQ: Crowdsourced query understanding, *CIDR ,* 2013.

/69/ Greg Little , Lydia B. Chilton , Max Goldman , Robert C. Miller, TurKit: human computation algorithms on mechanical turk, Proceedings of the 23nd annual ACM symposium on User interface software and technology, 2010.

/70/ Xuan Liu , Meiyu Lu , Beng Chin Ooi , Yanyan Shen , Sai Wu , Meihui Zhang, CDAS: a crowdsourcing data analytics system, *Proceedings of the VLDB Endowment*, v.5 n.10, pages 1040-1051, 2012.

/71/ X. L. Dong, L. Berti-Equille, and D. Srivastava. Integrating conflicting data: The role of source dependence. *PVLDB, 2(1):* pages 550–561, 2009.

/72/ X. Liu, X. L. Dong, B. C. Ooi, and D. Srivastava. Online data fusion. *PVLDB, 4(11):* pages 932–943, 2011.

/73/ Vikas C. Raykar , Shipeng Yu , Linda H. Zhao , Gerardo Hermosillo Valadez , Charles Florin , Luca Bogoni , Linda Moy, Learning From Crowds, *The Journal of Machine Learning Research*, pages1297-1322, 2010.

/74/ Panagiotis G. Ipeirotis , Foster Provost , Jing Wang, Quality management on Amazon Mechanical Turk, *Proceedings of the ACM SIGKDD Workshop on Human Computation*, 2010.

/75/ Aditya Parameswaran , Anish Das Sarma , Hector Garcia-Molina , Neoklis Polyzotis , Jennifer Widom, Human-assisted graph search: it's okay to ask questions, *Proceedings of the VLDB Endowment*, v.4 n.5, pages 267-278, 2011.

/76/ S. Kundu et. al. A linear tree partitioning algorithm. SIAM Journal on Computing, pages 151–154, 1977.

/77/ Jeffrey M. Rzeszotarski, Aniket Kittur, Instrumenting the Crowd: Using Implicit Behavioral Measures to Predict Task Performance, UIST'11, pages 13-22, 2011.

/78/ Anja Gruenheid, Donald Kossmann, Cost and Quality Trade-Offs in Crowdsourcing, *Proceedings of the First VLDB Workshop on Databases and Crowdsourcing*, 2013.

/79/ A Mahmood, W Aref, E Dragut, S Basalamah, The Palm-tree Index: Indexing with the crowd, *Proceedings of the First VLDB Workshop on Databases and Crowdsourcing*, 2013.

/80/ "Database Indexes", http://20bits.com/article/interview-questions-database-indexes, Retrieved 2013-10-29.

/81/ Joachim Selke, Christoph Lofi, Wolf-Tilo Balke, Pushing the Boundaries of Crowd-enabled Databases with Query-driven Schema Expansion, *VLDB, Vol. 5, No. 6*, pages 538-549, 2012.

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Magdeburg, den 10. Januar. 2014