ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Master's Thesis Nr. 59

Systems Group, Department of Computer Science, ETH Zurich

Query Optimization in CrowdDB

by

Erfan Zamanian Dolati

Supervised by

Prof. Donald Kossmann

May 2012–November 2012

inf | Informatik
Computer Science

Abstract

While database management systems have successfully established themselves as reliable and highly optimized tools for managing data, they still fall short when it comes to certain types of queries, such as subjective comparisons and finding missing data. Crowdsourcing databases, such as CrowdDB, offer a solution by harnessing the knowledge and problem-solving abilities of large groups of people in order to perform the tasks at which humans excel. Nevertheless, people are prone to make mistakes in their judgements and give inaccurate answers, which can negatively impact the quality of results. To solve this issue, the commonly adopted approach is to ask the same question several times from the crowd. However, such redundancy significantly increases the cost of solutions, making it an infeasible approach for crowdsourcing databases where a large number of queries needs to be answered with a limited budget and possibly a certain level of quality.

In this thesis, we address the inherent trade-off between cost and quality in the query optimization of crowd-enabled databases. Based on the intuitive idea that an answer is more likely to be correct if it comes from diverse and independent sources, we introduce the concept of access paths, and propose an abstraction model that allows the system to encourage the crowd to arrive at the solution from various paths.

In order to infer the answer with more confidence and less cost, we design a model that, using various access paths, guides the query optimizer to efficiently spend the user's budget to achieve quality results.

Acknowledgement

First and foremost, I offer my sincerest gratitude to my supervisor, Prof. Dr. Donald Kossmann, for his constant patience, motivation and enthusiasm during the course of this thesis. Our meetings and discussions made this work possible. I would also like to thank Besmira Nushi for her valuable comments and suggestions.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

Despite the significant advances in many areas of the field of computer science, there are certain tasks that are very challenging for machines. For such tasks, computer algorithms fail to provide satisfactory results, or in some cases, they provide no solutions at all.

Crowdsourcing has emerged as a feasible solution to these problems. It incorporates the crowd participation to accomplish a variety of tasks, such as labeling images, natural language processing, finding missing data and subjective comparisons.

Crowdsourcing internet marketplaces, such as Amazon Mechanical Turk, provide their users with a platform that facilitates easy distribution of large numbers of repetitive microtasks out to a community of individuals for completion.

With the help of these platforms, crowdsourcing has found its way into the realm of large data management. CrowdDB and Deco are two examples of database management systems that leverage the crowd's capabilities to integrate crowdsourced data with existing data in order to answer queries that a traditional database system fails to do so.

Due to the fact that people, unlike machines, may err in their judgements, rigorous quality control is required in any crowdsourcing system, especially data management systems, where manual supervision needs to be at its lowest level. In addition, employing human labour costs money. Therefore, in order to use the crowd in a performant and cost-efficient way, quality control must be accompanied with query optimization techniques. In other words, there is a trade-off between the cost and the quality of final result.

1.2 Problem Statement

In a crowdsourcing system, human workers are employed to perform various types of tasks. While this enables solving problems that are otherwise very difficult or even impossible for machines, it raises its own issues. People, unlike machines, are prone to errors. Some workers might not possess the adequate ability or training to perform the task. Some workers might become fatigued or lose focus during a complex task, which affects their accuracy. There are also malicious workers who try to maximize their profits by spamming the system. Therefore, quality control is a crucial aspect of almost any crowdsourcing system.

Hence, proper validation methods should be deployed in order to increase the chance of identifying the correct result. However, in sharp contrast to traditional information retrieval systems, there is usually no ground truth in crowd-enabled databases to evaluate the workers responses. In an information retrieval system, quality of the retrieved data is conventionally judged as the measure of relevance to the known ground truth, whereas in crowdsourcing databases there is no or little prior knowledge about the gold standard. Consequently, quality acquires a new meaning in the context of crowdsourcing databases.

The query optimization in crowdsourcing systems introduces new challenges that do not usually arise in its traditional counterpart. Asking the crowd incurs monetary cost. Besides, workers usually take much more time than machines to respond to questions. Addressing these problems reveals a fundamental trade-off between quality, monetary cost and latency. In particular, we are interested to study two optimization problems. First, we want to know that with a given budget, how good our answer could be. Second, we want our crowdsourcing system to predict how much it costs to satisfy user's required quality.

In order to compensate for poor quality of some workers, the solution that is commonly employed in such systems is redundancy. That is, to validate results by asking the same question several times from different people and then aggregate workers' responses. Another approach would be to ask different questions with presumably the same answer from different workers. To illustrate this approach, assume that two different research groups who work separately on a hypothesis reach the same conclusion and approve the hypothesis. In this case, one can claim with more confidence that the initial hypothesis has been valid.

This simple example suggests that using diverse paths to arrive at the logically same data ensures quality results for crowdsourced tasks. The idea of considering different paths resembles different access paths in relational database systems using which, the query optimizer considers multiple query plans and attempts to determine the most efficient one. However, unlike traditional query optimizers, not all plans are practical to employ when dealing with a crowd. Therefore, we need a meta model that takes into account the capabilities of the crowd. The system will then be enabled to focus on more efficient paths and encourage the workers to explore them. In this case, the obtained solution costs less and results in more quality.

1.3 Contribution

We introduce the concept of access paths and discuss their implications for the design of the database optimizer. Access paths are different ways to reach the same data. We present an abstraction model that allows the developers to define the set of access paths related to the application scenario.

The optimization problem is formally defined using the metrics of cost and quality. More specifically, we provide the database users the ability to specify their requirements in terms of these two metrics. We consider two optimization problems: the budget-constrained and the quality-constrained optimization. In the former, the system attempts to efficiently invest the user's budget, while in the latter, the cheapest possible plan that satisfies the quality requirement is sought for.

Our proposed query optimizer consists of three models. The fist component of the optimizer is the decision-making model that aggregates workers' responses. Various aggregation methods along with their advantages and limitations are studied in great detail. The second component is an expectation model that allows the users to specify what is important to them. Based on their priorities, the users will be provided with two options for the quality metric. Specifically, they can either choose the completeness or F1-score. The model then evaluates plans based on the user-preferred metric. The last component of the optimizer is the prediction model that generates all possible plans, iterates over the alternatives and finds the best one that satisfies user requirements. Since resource management is of great concern in database systems, we propose a greedy algorithm which employs heuristics to obtain approximate solutions in a reasonable time.

We also present the results of various experiments and investigate the interplay of many factors including the aggregation method, the objective function and the user's requirements. Based on our findings, we provide some guidelines for the developers to adjust the query optimizer.

1.4 Overview

The rest of the thesis is organized as follows: chapter 2 details the components of our proposed query optimizer. First, we will thoroughly explain the abstraction model the system uses to describe access paths. We will continue with describing the aggregation methods. The crowdsourcing database needs to examine all possible paths and select the cheapest one. However, unlike traditional RDBMS, access paths are not equivalent in terms of their quality. That is why the system needs to predict the efficiency of each path. This will be discussed in sections 2.3 and 2.4. Results and findings are given in chapter 3. Chapter 4 provides a summary of prior work related to crowdsourcing communities and optimization in these systems. The last chapter summarizes our findings and the contributions made by this work.

Chapter 2

Crowdsourcing Optimization Model

Section 2.1 describes the proposed model for access paths. Section 2.2 details three methods using which the system can aggregate the results of various workers. In section 2.3, we introduce a model that helps the system evaluate plans based on two different quality metrics. In section 2.4, we propose algorithms that examine alternative plans and select the cheapest plan that satisfies the user's requirements.

2.1 Access Paths Abstraction Model

2.1.1 Motivation

James Surowiecki, in his famous and widely discussed book *"The Wisdom of Crowds"* [29], argues that contrary to common belief, using a group of experts to solve a decision making problem does not always guarantee the most optimal solution. Such groups might lack a fundamental element that might prevent them from making the best decision. As a matter of fact, in many problems, a small group of experts can be outperformed by large groups of ordinary people with the proper characteristics and structure. He states that the most important characteristics for the crowd to be wise include diversity, independence and decentralization. To put it differently, a solution can be considered plausibly valid if it is produced by a group of people who independently take different paths and reach a conclusion.

However, the crowd is not always motivated to take multiple paths. Sometimes, there exists a simple way of answering to a question that the crowd prefers to go that way, effortlessly or with little trouble. The situation appears to be more problematic when the workers receive monetary rewards for answering questions, because not only they now do poor work, but they also do a lot of it in order to receive more rewards.

	Traditional DBMS	Crowd-enabled databases
Cost	Running time of different paths may vary	Monetary cost of different paths may vary
Quality	All paths are equivalent in terms of output	Some paths are more trustable than others
Paths vary in	The existence/types of indices (B^+ tree, clustered hashing), types of scans (nested loop, full scan)	Cost and reliability

Table 2.1: Comparison of the concept of access paths in traditional DBMS and crowd-enabled databases

Accordingly, crowdsourcing systems should be equipped with a mechanism by which they are enabled to encourage their crowd (or workers, in our terminology) to explore various directions to a given problem. It is only then that the system can avoid falling into *fake consensus* traps that might be likely due to the fact that the crowd only considers a few paths.

We will return to this point, but for now, let us revisit the concept of access paths in database systems. In a traditional DBMS, an access path is the path chosen by the system to retrieve the requested data. To a great extent, SQL is a declarative language, meaning that queries are stated non-procedurally. The issuer does not specify any reference to access paths in his/her query. The DBMS optimizer lays out possible paths of retrieving the answer to the query. It then assigns cost to each alternative, iterates through them and attempts to choose the most efficient one based on the optimization criteria [10]. Many relational database management systems, such as IBM System R [27], use access paths to improve their performance tremendously. Table 2.1 summarizes the differences between access paths in traditional DBMS and crowdsourcing systems. .

For the rest of this thesis, we demonstrate our approach using a sample movie theater database with the following schema:

```
Movies(id, name, length, director, genre, year)
MovieTheaters(id, name, city, addr, URL, tel)
MovieShowing(theater_id, movie_id)
```

Now, assume that we are interested to know what movie is screened by cinema *Hollywood*. The question is translated to the following SQL query:

```
SELECT M.name FROM Movies M, MovieTheaters MT, MovieShowing MS
WHERE MT.name = "Hollywood"
```

```
AND MT.id = MS.theater_id
AND MS.movie_id = M.id
```

An easy way to find the answer is to directly ask the crowd: "What movie is screened by cinema *Hollywood*", and in order to ensure that we will receive the correct answer, we may obtain results from more workers. Even in this scenario, we cannot be confident enough that the results we receive truly reflect responses that are obtained from independent sources of information. The reason lies on the fact that many workers rely only on the same sources. For example, it is quite probable that a large number of these workers consult the same lists on the web. It is not an entirely unreasonable assumption that they search the name of the movie theater on a search engine like Google, and because they click on the first few links, they will all return the same answer.

The dependency of workers in making errors has been also noted in other research works. For example, Trushkowsky et al. [30] posted some tasks (including the list of UN nations, US states and ice cream flavours) on Amazon Mechanical Turk platform, and observed that there are some workers who submit the same answers in the same order. They referred to this effect as list walking and claimed that list walking most likely happens because such workers rely only on the same lists on internet when performing a task.

The movie theater example suggests that there might be a relationship between using various paths and quality of the final result. It can be argued that what accounts for quality answers is that the workers' responses from different paths are independent of each other. In other words, two workers who visit the theater's website might both consult the same wrong source (for example an outdated website), while workers who take different paths are likely independent from each other. If one makes a mistake, the likelihood of the other workers submitting the same wrong answer does not necessarily increase.

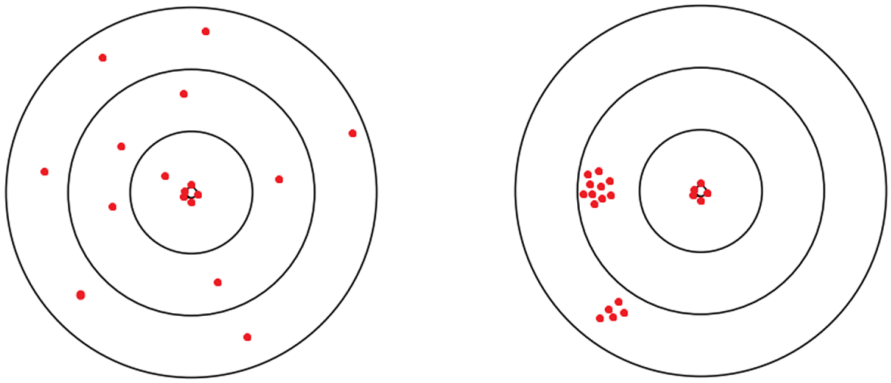
What we are looking for is ways to encourage and motivate workers to reach the answer from various paths. For example, the crowdsourcing system should ask one worker to dial the movie theater's number and ask them directly. Another worker can be asked to go to the theater's website and look for the answer there. Although not very rational, but if we have enough time and budget, we can also have some workers personally go there and ask the box office cashier.

2.1.2 Model Description

The core concept of crowdsourcing that justifies leveraging the crowd in tasks that are difficult or expensive for machines is that crowd is not always correct and may make mistakes, however on aggregate the errors cancel out, resulting in a quality answer. Figure 2.1a illustrates the preceding idea of the crowd using a simple model. Each point in the figure represents one worker and the task is to find the center of the circle Each worker submits his/her guess. There are some workers who, of course, respond with the exact correct answer. All

workers independently submit their guesses. In this case, it is very likely that a reasonable approximate can be found by aggregating all guesses.

Although this assumption is not entirely false (and that is why crowdsourcing functions, at least for many problems), sometimes it does not hold. As described in the previous section, the responses might be not completely independent of each other. Because sources of information might be reused many times by various workers, we expect to see *clusters of answers* in reality. This idea is depicted in figure 2.1b. Instead of random responses, there exists clusters of points, suggesting that if one worker makes an error, it is likely that some other workers make the same error.



(a) Random crowd **(b)** Clustered crowd

Figure 2.1: A simple model of crowd

The argument we make here that justifies using various access paths to get to the presumably same result is that we assume that workers are less correlated when they use different paths. More specifically, we make the following fundamental assumption:

Assumption 1. *Errors are correlated within an access path, but independent across access paths.*

Figure 2.2 depicts the crowd model incorporating the above assumption. Furthermore, we make a simplifying assumption for the error model. We argued before that different access paths may have different reliabilities. One way to model reliability is to associate *error rates* to access paths. Our simplifying assumption states that:

Assumption 2. *Errors are associated to access paths, and not individual workers.*

We need a framework that helps the crowdsourcing system discover various plans for performing a task, evaluate and compare these plans (similar to a query optimizer in a traditional RDBMS) and find the most optimal plan in terms of cost and quality.

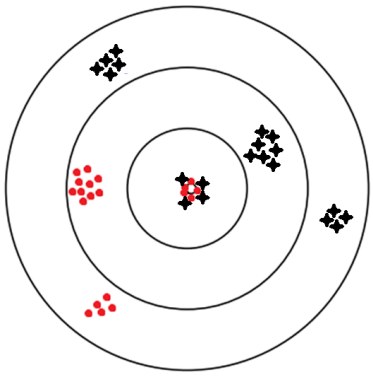


Figure 2.2: The simple crowd model with two access paths. The points represent the workers' responses by the first access path, and the stars represent the responses by the second access path.

The framework must contain metadata about the database and the crowd. The metadata describes practical (doable for the crowd) paths for questions. Practicality of a path is entirely associated to the capabilities of the crowd. Paths have different costs and qualities, but combined they may increase quality. In the example of the movie theater in section 2.1.1, having a worker visit the theater's website might cost lower than having him call and ask, but also it is more likely that the worker gives a wrong answer. Compared to these two paths, having a worker go to the theater's place costs much higher and probably takes more time, making it an undesirable path.

More specifically, the model must describe all possible paths along with information about each of them. Similar to traditional databases, it must know the conditions under which an access path can be used. Beside the cost of each access path (which can be considered as the monetary reward for workers who explore the path), it must also store information regarding error properties of the path. These properties include error rate and error domain size. For example, in figure 2.1b, the error domain size is two, as there are two clusters of error points.

Let us now formally define our abstraction model. $\mathcal{A}$ is the set of all access paths stored in the system. Each access path $i \in \mathcal{A}$ is characterized by the following entries:

- **Access index** a_i: The list of the table columns that must have known values for the access path to be used. Workers who use the access path will be provided with these known values. This can be considered as the input to the access path.
- **Properties**: This entry constitutes:
  - **Cost** c_i: Monetary cost of exploring the access path.

- **Error Rate** e_i: The associated probability of returning erroneous results.
- **Domain Size** s_i: If we consider the worker's response as a variable, the domain size of an access path simply equals the number of possible values that this variable can take on.
- **Error Overlap Size** o_i: Errors in different access paths might overlap with each other. Note that the value of this entry is shared by all paths.
- **Description** d_i: A human-readable text that provides the workers with an instruction to use the access path.

- **Path result** r_i: The list of the table columns whose values will be determined after exploring the access path. A star ('*') is used to show that the access path fills in the values for all columns of the table. This entry can be considered as the output of the access path.

Besides, we make another assumption regarding the distribution of errors in an access path.

Assumption 3. *Within an access path, all error values are equally likely to occur. Therefore, we have:* $p(error_i) = \frac{e_i}{s_i - 1}$.

Two of the abovementioned entries need further explanation. The domain size of an access path specifies the number of clusters of wrong responses associated to the path. Since only one value can be the correct answer, and the rest is wrong, the domain size of two means that all the workers who give wrong responses agree on the same answer. The second entry is the error overlap size. It is the number of common wrong values (associated to wrong responses) between all access paths. Please note that for simplicity, we assume that the error overlap size is not pair-wise between each two access paths. In other words, it represents the number of clusters of wrong responses that are common in all access paths.

Example 1. *Table 2.2 defines a possible set of access paths for the movie database.*

Note that the crowdsourcing database cannot invent such rules, as it needs a great deal of domain knowledge of the problem. These paths are defined by the user who writes the application on top of the database. However, the system must provide means to allow its users to specify such paths. Furthermore, the crowdsourcing database must find the best way to exploit an access path. In other words, if it leads workers along one access path, it must facilitate the process and support the workers along that path as much as possible.

The query optimizer will be able to explore an access path only if the path's access indices (a_i) have already defined values. In this case, workers are provided with these pieces of data, and are asked to find the missing information (as defined by *path result* r_i).

a_i	c_i	e_i	s_i	o_i	d_i	r_i
M.name	1	0.3	2	0	Search on internet	M.*
M.director	1	0.3	2	0	Search on internet	M.*
MT.name	2	0.2	3	0	Look up in tel directories	MT.add, MT.tel
MT.name	1	0.3	3	0	Search on internet	MT.*, MS.*, M.name
MT.tel	3	0.15	2	0	Call the number	MT.*, MS.*, M.name
MT.URL	1	0.35	2	0	Visit the URL	MT.*, MS.*, M.name
MT.addr	2	0.2	4	0	Use Google map	MT.name, MT.URL
MT.addr	6	0.1	2	0	Go and ask directly	MT.*
MT.city	3	0.3	3	0	Look up in yellow pages	MT.name, MT.tel

Table 2.2: The set of access paths for the movie database

After the user issues a query, the system can then design a set of *plans* to answer the query. Each plan might consist of one or more access paths, each should be executed after another. This helps the system break down a query to step-by-step parts, each of which is a human-doable task.

Example 2. *Assume that the user wants to know the information regarding the movies that are screened in the movie theater next to his/her house. Therefore, the following query is issued:*

```
SELECT M.name FROM Movies M, MovieTheaters MT, MovieShowing MS
WHERE MT.addr = "Universitatstr.  60"
      AND MT.id = MS.theater_id
      AND MS.movie_id = M.id
```

One possible plan for this query consists of three steps:

1. *MT.addr $\longrightarrow$ (MT.name, MT.URL)[By using Google map]*
2. *MT.URL $\longrightarrow$ (MT.*, MS.*, M.name) [By visiting the URL]*
3. *M.name $\longrightarrow$ (M.*) [By searching on internet]*

Workers may give various responses for a given task. In the next section, we will study methods that help the system aggregate workers' responses. In section 2.3, we will introduce a toolbox of techniques that allow the system to reason about the quality and cost of a plan.

2.2 Decision-making Models

In section 2.1, we explained how a crowdsourcing system can benefit from encouraging its workers to explore various access paths. The access path model

allows the system to design different plans. In the context of relational DBMS and particularly SQL, only one result must be given as the value of a tuple. That is why the system requires a model to analyze and aggregate workers' responses. In this section, three decision-making strategies along with their advantages and limitations will be thoroughly investigated.

In order to be able to fully focus on the decision-making function, for the rest of the thesis we will assume that quality control is only carried out for a single record. In other words, it will be assumed that there exists only one record in the database that satisfies the query, and all plans have no ambiguity regarding the record's key. We further assume that the goal is to find the correct value for the requested non-key attribute. This thesis will not deal with *entity resolution* problem and its implications for the crowdsourcing system. There are other research works that have investigated this critical issue and its importance for data integration and cleaning [33].

Furthermore, we will only examine single-step plans. To put it another way, our focus will be mainly on the following scenario: we have a list of access paths $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ which are all applicable to the query. Each a_i produces a response r_i. The goal is to aggregate all results $\mathcal{R} = \{r_1, r_2, \ldots, r_n\}$ and accept an answer r_j which satisfies the requirements of the decision-making function. If no such answer exists, it will respond with 'undecided'.

Figure 2.3 illustrates the role of the decision-making function in the presence of multiple access paths. The decision maker receives and aggregates votes from workers and attempts to make the final decision based on these votes and their associated access paths. More precisely, the input of this function is votes from one or more access paths, and its output can be either one of the two choices:

- **Winner vote**: The single vote that satisfies the requirements of the decision-making function.
- **Undecided**: If no vote meets the requirements, the decision-making function returns this response.

2.2.1 Majority Vote Strategy

Majority vote has been extensively employed for quality control in many crowdsourcing systems, such as CrowdDB [7], CrowdSearch [35], Qurk (along with other methods) [24] and Soylent [6].

The idea is simple. Given n responses from the workers, an answer will be accepted if the majority of them, i.e. at least $\lceil \frac{n}{2} \rceil$ workers, agree on the same answer. If no majority exists, it returns undecided.

Example 3. *Let us continue and elaborate more on the figure 2.3. Assume that the mentioned three access paths have the properties described in table 2.3.*

Let the latent correct answer be 'a', and the results of the access paths are as follows: $r_1 = \{\text{'}a\text{'}, \text{'}b\text{'}\}$, $r_2 = \{\text{'}a\text{'}, \text{'}c\text{'}\}$ *and* $r_3 = \{\text{'}a\text{'}\}$. *The majority vote strategy accepts 'a', as the majority agrees on it (three out of five workers), which in this example , happens to be the latent correct answer.*

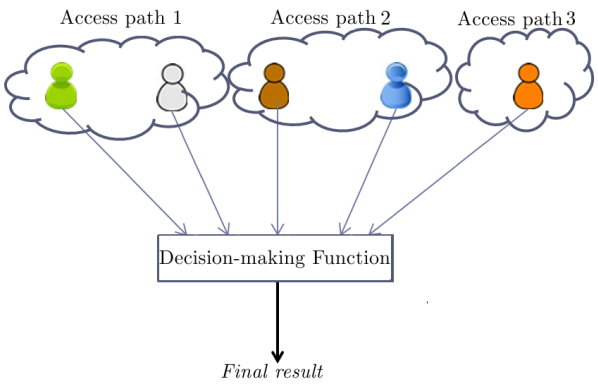


Figure 2.3: The role of decision-making function in the presence of multiple access paths

Access path #	Cost	Error rate	Domain size	Error overlap size	Description
1	1	0.35	3	1	Visit URL
2	2	0.20	3	1	Ask a friend
3	3	0.15	2	1	Call and ask

Table 2.3: The access paths and their associated information in example 3

The major advantage of the majority vote strategy is its simplicity and as a result, efficiency. It also guarantees that no other responses have more votes than the accepted one. However, its simplicity is also its greatest disadvantage. The strategy is too simple that it does not take into account the characteristics of the access paths. It considers all access paths similar, even though some are more reliable than others. The second drawback of this strategy is that in many cases, it might be unable to decide which vote is the winner vote. Therefore, it is very likely that it returns undecided responses.

2.2.2 Frequency Vote Strategy

As it was argued in the previous section, in many scenarios, due to absence of any response with or more than $\lceil \frac{n}{2} \rceil$ votes, majority vote is unable to decide and accept a response, and therefore remains undecided. *Frequency vote* is a slightly different variation of this method, where instead of accepting the majority, it accepts the most frequent one, i.e. the response with the highest number of votes. If one does not exist, it answers with undecided.

The vote on which the majority agrees, obviously, is the most frequent one too. As a result, whenever the frequency vote answers with undecided, the majority vote responds with the same. However, in some cases, the frequency vote returns a winner vote while the majority remains undecided.

Example 4. *In example 3, it is clear that the frequent vote is the same as the majority vote, i.e. 'a'. Let us consider another set of votes for the scenario in figure 2.3. This time, assume that* $r_1 = \{\text{'}a\text{'}, \text{'}b\text{'}\}$, $r_2 = \{\text{'}a\text{'}, \text{'}c\text{'}\}$ *and* $r_3 = \{\text{'}d\text{'}\}$.

Unlike majority vote, that answers with undecided, the output of the frequency vote will be 'a' (as this response has two votes, as opposed to all other responses that have only one vote).

The frequency vote provides the same advantages as the majority vote (being simple and efficient), while reduces the likelihood of ties. Nevertheless, in cases where the first two or more responses obtain the same number of votes, it cannot provide a result, even if one response comes from a more reliable source.

2.2.3 Likelihood Vote Strategy

We discussed in the last two sections that majority and frequency vote, although offer a simple model for evaluating workers' responses, occasionally fail to provide the correct answer or any answer at all. This issue stems from two aspects of both methods.

On the one hand, they both consider only *the number of votes* on responses. Accordingly, their evaluation approach is highly sensitive to the number of workers who are assigned the task and the domain size of the responses. For example, if a question with expected boolean answers, such as a polar question (a question whose answer is either yes or no), is asked from an even number of workers, then it is not that unlikely that it finally reaches a tie, and therefore both abovementioned methods answer with undecided.

On the other hand, both methods count the number of votes for each response across all paths, and then decide whether it is the most frequent response (or in the case of majority vote, whether it gets an agreement above 50%). So from their perspective, there is no difference between various workers' responses. They consider all access paths the same, which is not the case, as some paths are more trustable than others.

The latter turns out to be more important when the crowdsourcing system explores a mix of reliable and unreliable paths. The whole idea of using various access paths is that by having different routes explored (which get to the logically same data), the system will be able to provide its users with better answers for their queries.

Example 5. *For the scenario depicted in figure 2.3 (and the access paths listed in table 2.3), assume that the responses are as follows:* $r_1 = \{\text{'}b\text{'}, \text{'}c\text{'}\}$, $r_2 = \{\text{'}a\text{'}, \text{'}b\text{'}\}$ *and* $r_3 = \{\text{'}a\text{'}\}$.

Both 'a' and 'b' have two votes, resulting in an undecided answer by the majority and frequency vote. However, 'a' comes from more trustable sources than 'b'.

In this section, we propose *likelihood vote*, another decision-making method that helps the system evaluate the workers' responses by their probability of being the correct answer. We want to know that given the responses submitted

by workers, along with their associated access paths, how likely it is for a specific response to be the correct answer. Then the system can accept the response with the highest likelihood. Two recent research studies propose similar probability-based approaches in quality control in a crowdsourcing system [18, 12].

Let us start by formally defining the variables used. We previously assumed that for a list of n access paths $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$, $\mathcal{R} = \{r_1, r_2, \ldots, r_n\}$ represents the set of response sets from all access paths, where $r_i = \{r_{i1}, r_{i2}, \ldots, r_{im}\}$ in turn is the set of m_i workers' answers associated to a_i. Because some responses from different access paths might be the same, assume further that the set of unique responses is $\mathcal{U} = \{u_1, u_2, \ldots, u_i\}$. Let random variable X denote the correct answer of the query. Table 2.4 shows a summary of defined notations.

For a specific response $\bar{r}$, we want to calculate the probability of $\bar{r} = X$ given the responses from all workers, $\mathcal{R}$. In the probability theory language, it is translated to $P(\bar{r} = X|\mathcal{R})$.

Using the formula for Bayes' theorem, we have:

$$\begin{aligned} P(\bar{r} = X|\mathcal{R}) &= \frac{P(\mathcal{R}|\bar{r} = X)P(\bar{r} = X)}{P(\mathcal{R})} \\ &= \frac{P(\mathcal{R}|\bar{r} = X)P(\bar{r} = X)}{\sum_{u_i \in \mathcal{U}} P(\mathcal{R}|u_i = X)P(u_i = X)} \end{aligned} \tag{2.1}$$

Without any priori knowledge, all responses are equally likely to be the correct answer. In other words, $P(u_i = X)$ remains the same for all $u_i \in \mathcal{U}$. Therefore, equation 2.1 can be simplified to:

$$P(\bar{r} = X|\mathcal{R}) = \frac{P(\mathcal{R}|\bar{r} = X)}{\sum_{u_i \in \mathcal{U}} P(\mathcal{R}|u_i = X)} \tag{2.2}$$

Let us now compute the numerator of equation 2.2, $P(\mathcal{R}|\bar{r} = X)$. Each access path a_i produces $\bar{r}$ with probability $1 - e_i$ and produces any other responses with probability $\frac{e_i}{s_i - 1}$. Therefore, we have:

$$P(\mathcal{R}|\bar{r} = X) = \prod_{a_i \in \mathcal{A}} \left(\prod_{\substack{r_{ij} \in r_i \\ r_{ij} = \bar{r}}} (1 - e_i) \prod_{\substack{r_{ij} \in r_i \\ r_{ij} \neq \bar{r}}} (\frac{e_i}{s_i - 1}) \right) \tag{2.3}$$

If we plug what we have calculated into equation 2.2, we will have:

$$P(\bar{r} = X|\mathcal{R}) = \frac{\prod_{a_i \in \mathcal{A}} \left(\prod_{\substack{r_{ij} \in r_i \\ r_{ij} = \bar{r}}} (1 - e_i) \prod_{\substack{r_{ij} \in r_i \\ r_{ij} \neq \bar{r}}} (\frac{e_i}{s_i - 1}) \right)}{\sum_{u_i \in \mathcal{U}} P(\mathcal{R}|u_i = X)} \tag{2.4}$$

The goal of the decision-making model is to find the *best* answer from a list of workers' responses. If no such answer exists, it must answer with undecided. In the likelihood vote approach, the best answer is the response that is most

Notation	Explanation
$\mathcal{A}$	The set of all access paths
n	The number of access paths applicable to the query
a_i	The i-th access path
e_i	The error rate of the i-th access path
c_i	The cost of the i-th access path
s_i	The answer domain size of the i-th access path
$\mathcal{R}$	The set of response sets of all access paths
r_i	The response set of the i-th access path
r_{ij}	The response of the j-th worker of the i-th access path
$\mathcal{U}$	The set of unique responses
u_i	The the i-th unique response
f_{dm}	The adopted decision-making function
X	Random variable that denotes the correct answer to the query
W_{ij}	The j-th wrong answer of the i-th access path
$\mathbb{V}_l$	The set of possible values for plan l

Table 2.4: The summary of notations

likely to be correct, i.e. the response with the highest likelihood, termed as $r_{accepted}$:

$$\begin{aligned} r_{accepted} &= \arg\max_{u \in \mathcal{U}} P(u = X|\mathcal{R}) \\ &= \arg\max_{u \in \mathcal{U}} \frac{\prod_{a_i \in \mathcal{A}} \left(\prod_{\substack{r_{ij} \in r_i \\ r_{ij} = u}} (1 - e_i) \prod_{\substack{r_{ij} \in r_i \\ r_{ij} \neq u}} (\frac{e_i}{s_i - 1}) \right)}{\sum_{u_i \in \mathcal{U}} P(\mathcal{R}|u_i = X)} \end{aligned} \tag{2.5}$$

The denominator in equation 2.5 is the summation of $P(\mathcal{R}|u_i)$ for all $u_i \in \mathcal{U}$, and therefore is not dependent on the choice of u in $\arg\max$. In other words, we have:

$$P(u = X|\mathcal{R}) \propto \prod_{a_i \in \mathcal{A}} \left(\prod_{\substack{r_{ij} \in r_i \\ r_{ij} = u}} (1 - e_i) \prod_{\substack{r_{ij} \in r_i \\ r_{ij} \neq u}} (\frac{e_i}{s_i - 1}) \right) \tag{2.6}$$

Since we are interested to find the answer with the *maximum* likelihood, and the exact values of probabilities are not important for us, we can therefore remove the denominator and transform the equation 2.5 into:

$$\begin{aligned} r_{accepted} &= \arg\max_{u \in \mathcal{U}} P(u = X|\mathcal{R}) \\ &= \arg\max_{u \in \mathcal{U}} \prod_{a_i \in \mathcal{A}} \left(\prod_{\substack{r_{ij} \in r_i \\ r_{ij} = u}} (1 - e_i) \prod_{\substack{r_{ij} \in r_i \\ r_{ij} \neq u}} (\frac{e_i}{s_i - 1}) \right) \end{aligned} \tag{2.7}$$

In summary, equation 2.7 can be interpreted as follows: for each unique response, go through all access paths. For each access path, iterate over all responses. If the response is the same as the unique answer for which we are calculating the probability, consider it as a success, and otherwise, as a failure. Finally, check if there is any answer that has uniquely the highest probability and return it. If no such answer exists, answer with undecided.

Example 6. *Let us now apply the likelihood vote method on the scenario in example 5. The set of response sets is* $\mathcal{R} = \{r_1, r_2, r_3\}$*, where* $r_1 = \{\text{`}b\text{'}, \text{`}c\text{'}\}$*,* $r_2 = \{\text{`}a\text{'}, \text{`}b\text{'}\}$ *and* $r_3 = \{\text{`}a\text{'}\}$*. The list of unique answers is therefore* $\mathcal{U} = \{\text{`}a\text{'}, \text{`}b\text{'}, \text{`}c\text{'}\}$*. We now apply equation 2.6 to each unique answer:*

$$P(\text{`}a\text{'} = X|\mathcal{R}) \propto (\tfrac{e_1}{2})^2 \cdot (1-e_2) \cdot (\tfrac{e_2}{2}) \cdot (1-e_3) = 0.175^2 \cdot 0.8 \cdot 0.1 \cdot 0.85 = 0.0020825$$
$$P(\text{`}b\text{'} = X|\mathcal{R}) \propto (1-e_1) \cdot \tfrac{e_1}{2} \cdot \tfrac{e_2}{2} \cdot (1-e_2) \cdot e_3 = 0.65 \cdot 0.175 \cdot 0.1 \cdot 0.8 \cdot 0.15 = 0.001365$$
$$P(\text{`}c\text{'} = X|\mathcal{R}) \propto \tfrac{e_1}{2} \cdot (1-e_1) \cdot (\tfrac{e_2}{2})^2 \cdot e_3 = 0.175 \cdot 0.65 \cdot 0.1^2 \cdot 0.15 = 0.000170625$$

As the numbers show, response 'a' is 1.5 times and 12 times as likely as 'b' and 'c', respectively. Therefore, likelihood vote would accept 'a' as the correct answer.

In example 6, the reason that 'a' is accepted, and for example not 'b', is because 'a' comes from more trustable sources. Putting more weight on more reliable access paths is what the frequency and majority vote do not consider, and are unable to return a winner response in this example.

Accepting an answer based on the comparison of absolute values of probabilities might be very sensitive to small differences. To explain this issue, assume that for a hypothetical scenario, the probability associated to one response is p, while that of another response is $p + \epsilon$, where ϵ is a very small number and close to zero. In this case, the decision of the likelihood vote would not be robust.

The model must be provided with a *measure of confidence* that allows the decision-making function to distinguish between an answer that has strong evidence and an answer that has weak evidence. To accomplish this goal, we define **quorum** as the level of certainty in accepting an answer.

More precisely, the comparison of two likelihood values of v_1 and v_2 is considered a tie, if

$$\frac{|v_1 - v_2|}{\max(v_1, v_2)} < q \tag{2.8}$$

where q is the predefined quorum. By normalizing the quorum, we ensure that its value is in the range of 0 to 1. With this definition, a small value of quorum means that the likelihood function is sensitive to even slight differences, while a high value of quorum implies that the method only cares about significant differences. If the two values are close to each other, the method responds with undecided.

2.3 Expectation Model

The methods proposed in section 2.2 enable the crowdsourcing system to aggregate workers' responses and accept an answer. However, since those methods take as input the set of workers' responses, it is not possible to apply them before posting the tasks on the crowdsourcing platform. We need an *expectation model* that allows the system to evaluate plans based on a quality criterion. In other words, the query optimizer must be able to assign *scores* to plans and execute the plan with the highest score.

Please recall that in the beginning of the previous section, we stated that we would only examine single-step plans. With this assumption, a plan specifies the number of workers that must be asked using each access path.

Let l represent the plan to which we want to assign a score. A plan is described by a set each of whose members is an ordered pair of (a, q), where a is the access path and q is the number of questions (workers) that must be asked using path a.

Example 7. *The plan $l = \{(a_1, 1), (a_3, 2)\}$ specifies that one worker must be asked using access path a_1, and two workers using access path a_3 (indices refer to access path numbers in table 2.3).*

Let us consider plan l as a random variable, where the set of possible values that it can take on is the set of all possible workers' responses. To put it differently, a value for l consists of a specific set of votes for each access path. We will denote the set of values for plan l by $\mathbb{V}_l$.

Remember that the random variable X represents the correct answer to the query. We will use W_{ij} to refer to the j-th wrong answer of the i-th access path.

Example 8. *One possible value for the described plan in example 7 is $\{a_1 : \{W_{12}\}, a_3 : \{X, W_{31}\}\}$. The set of all possible values is as follows:*

$$\begin{aligned}\mathbb{V}_l = \{&\{a_1 : \{X\}, a_3 : \{X, X\}\},\\ &\{a_1 : \{X\}, a_3 : \{X, W_{31}\}\},\\ &\{a_1 : \{X\}, a_3 : \{W_{31}, W_{31}\}\},\\ &\{a_1 : \{W_{11}\}, a_3 : \{X, X\}\},\\ &\{a_1 : \{W_{11}\}, a_3 : \{X, W_{31}\}\},\\ &\{a_1 : \{W_{11}\}, a_3 : \{W_{31}, W_{31}\}\},\\ &\{a_1 : \{W_{12}\}, a_3 : \{X, X\}\},\\ &\{a_1 : \{W_{12}\}, a_3 : \{X, W_{31}\}\},\\ &\{a_1 : \{W_{12}\}, a_3 : \{W_{31}, W_{31}\}\}\}\end{aligned}$$

From the perspective of the expectation model (which attempts to evaluate plans), the decision-making function takes as input a specific value from $\mathbb{V}_l$,

and responds with either a result (which can be correct or wrong) or undecided. Therefore, it can be specified by:

$$f_{dm} : \mathbb{V}_l \mapsto \{correct, wrong, undecided\} \tag{2.9}$$

where f_{dm} represents the decision-making function.

For ease of future reference, we borrow two terms from the traditional information retrieval terminology. **Correctness** represents the fraction of decisions (excluding undecided) made by the decision-making function that are correct. Furthermore, we will use **completeness** to refer to the fraction of decisions that are either correct or wrong (or simply, not undecided).

Let us formally define these two terms. Assume that the decision-making function has made some decisions, denoted by $|Y|$. We have:

$$Correctness = \frac{\text{count}_{y_i \in Y}(f_{dm}(y_i) = X)}{\text{count}_{y_i \in Y}(f_{dm}(y_i) \neq undecided)}$$
$$Completeness = \frac{\text{count}_{y_i \in Y}(f_{dm}(y_i) \neq undecided)}{|Y|}$$

Different users may have different requirements for the quality of results to their queries. For example, one user may prefer to receive results with higher correctness. For this user, the completeness may not be equally important. The other users may want the system to maintain a balance between correctness and completeness.

Therefore, the expectation model must allow the users to specify their requirements. To accomplish this goal, we define two quality objectives and allow the users to choose the one that satisfies their needs.

- **Correctness objective**: it evaluates plans based on their correctness. Between two plans, the plan with the greater correctness is evaluated higher.

- **F1-score objective**: in information retrieval, F1-score is a measure that considers both correctness and completeness. It is the harmonic mean of these two metrics:

  $$F_1 = 2 \cdot \frac{Correctness \cdot Completeness}{Correctness + Completeness} \tag{2.11}$$

  In F1-score objective, the goal is to strike a balance between correctness and completeness. Between two plans, the plan with the greater F1-score is evaluated higher.

In order to evaluate a plan l, which as we discussed is a random variable whose domain of values is $\mathbb{V}_l$, **expected value** appears to be a natural choice.

Expected value of a random variable is the weighted average of all possible values it can take on. In our case, however, we are not interested in the expected value of the random variable l. What we are looking for is the expected value of correctness or F1-score that corresponds to $f_{dm}(l)$.

Let us now calculate the expected value of quality metrics:

$$\mathrm{E}[Correctness] = \frac{\sum_{\substack{v_i \in \mathbb{V}_l \\ f_{dm}(v_i)=X}} p(v_i)}{\sum_{\substack{v_i \in \mathbb{V}_l \\ f_{dm}(v_i)\neq undecided}} p(v_i)} \tag{2.12a}$$

$$\begin{aligned} \mathrm{E}[Completeness] &= \sum_{\substack{v_i \in \mathbb{V}_l \\ f_{dm}(v_i)\neq undecided}} p(v_i) \\ &= 1 - \sum_{\substack{v_i \in \mathbb{V}_l \\ f_{dm}(v_i)=undecided}} p(v_i) \end{aligned} \tag{2.12b}$$

$$\mathrm{E}[F_1] = 2 \cdot \frac{\mathrm{E}[Correctness] \cdot \mathrm{E}[Completeness]}{\mathrm{E}[Correctness] + \mathrm{E}[Completeness]} \tag{2.12c}$$

In the above equations, $p(v_i)$ is the likelihood that v_i appears as the output of the crowd. The responses for each access path can be seen as extracting n balls of k different categories. Here, categories are equivalent to various possible answers to an access, and balls are votes on each answer. Similar to multinomial distribution, votes are statistically independent. Therefore, $p(v_i)$ is the probability mass function of this multinomial distribution. Let q_i represent the number of workers asked using the i-th access path. Although not precisely representative, we use $|X_i|$ and $|W_{ij}|$ to refer to the number of votes on the correct answer and on the j-th wrong answer of the i-th access path, respectively. By the definition of probability mass function, we can calculate $p(v_i)$:

$$\begin{aligned} p(v_i) &= \binom{q_i}{|X_i|, |W_{i1}|, \ldots, |W_{i(d_s-1)}|} (1-e_i)^{|X_i|} (\frac{e_i}{d_s-1})^{|W_{i1}|} \ldots (\frac{e_i}{d_s-1})^{|W_{i(d_s-1)}|} \\ &= \binom{q_i}{|X_i|, |W_{i1}|, \ldots, |W_{i(d_s-1)}|} (1-e_i)^{|X_i|} (\frac{e_i}{d_s-1})^{q_i-|X_i|} \end{aligned} \tag{2.13}$$

Example 9. *In this example, we calculate the two quality metrics of correctness and F1-score for the plan in example 7. We use the frequency vote because it is easier to perform the calculations. For each $v \in \mathbb{V}_l$, we calculate its probability of appearing, $p(v_i)$, along with the output of the aggregation. You can find the calculations in table 2.5.*

We are now ready to calculate the quality metrics:

$$\begin{aligned} \mathrm{E}[Correctness] &= \frac{p_(v_1) + p(v_2) + p(v_4) + p(v_7)}{1 - (p(v_5) + p(v_8)} \simeq 0.9753 \\ \mathrm{E}[Completeness] &= 1 - (p(v_5) + p(v_8) = 0.91075 \\ \mathrm{E}[F_1] &\simeq 2 \cdot \frac{0.9753 \cdot 0.91075}{0.9753 + 0.91075} \simeq 0.9420 \end{aligned}$$

i	$v_i \in \mathbb{V}_l$	$p(v_i)$	$f_{dm}(v_i)$
1	$\{a_1 : \{X\}, a_3 : \{X, X\}\}$	$\binom{1}{1,0,0} \cdot (1-e_1) \cdot \binom{2}{2,0} \cdot (1-e_3)^2 = 0.469625$	correct
2	$\{a_1 : \{X\}, a_3 : \{X, W_{31}\}\}$	$\binom{1}{1,0,0} \cdot (1-e_1) \cdot \binom{2}{1,1} \cdot (1-e_3) \cdot e_3 = 0.16575$	correct
3	$\{a_1 : \{X\}, a_3 : \{W_{31}, W_{31}\}\}$	$\binom{1}{1,0,0} \cdot (1-e_1) \cdot \binom{2}{0,2} \cdot e_3^2 = 0.014625$	wrong
4	$\{a_1 : \{W_{11}\}, a_3 : \{X, X\}\}$	$\binom{1}{0,1,0} \cdot \frac{e_1}{2} \cdot \binom{2}{2,0} \cdot (1-e_3)^2 = 0.1264375$	correct
5	$\{a_1 : \{W_{11}\}, a_3 : \{X, W_{31}\}\}$	$\binom{1}{0,1,0} \cdot \frac{e_1}{2} \cdot \binom{2}{1,1} \cdot (1-e_3) \cdot e_3 = 0.044625$	undecided
6	$\{a_1 : \{W_{11}\}, a_3 : \{W_{31}, W_{31}\}\}$	$\binom{1}{0,1,0} \cdot \frac{e_1}{2} \cdot \binom{2}{0,2} \cdot e_3^2 = 0.0039375$	wrong
7	$\{a_1 : \{W_{12}\}, a_3 : \{X, X\}\}$	$\binom{1}{0,0,1} \cdot \frac{e_1}{2} \cdot \binom{2}{2,0} \cdot (1-e_3)^2 = 0.1264375$	correct
8	$\{a_1 : \{W_{12}\}, a_3 : \{X, W_{31}\}\}$	$\binom{1}{0,0,1} \cdot \frac{e_1}{2} \cdot \binom{1}{1,1} \cdot (1-e_3) \cdot e_3 = 0.044625$	undecided
9	$\{a_1 : \{W_{12}\}, a_3 : \{W_{31}, W_{31}\}\}$	$\binom{1}{0,0,1} \cdot \frac{e_1}{2} \cdot \binom{2}{0,2} \cdot e_3^2 = 0.0039375$	wrong

Table 2.5: The calculations for example 9

Almost 97 out of 100 answers that the system returns are correct. Besides, in one tenth of the times, the system returns no answer.

2.4 Prediction Models

In the first section of this chapter, we introduced the concept of access paths and propose an abstraction model for storing and retrieving them. The second chapter discussed the methods that allow the system to aggregate workers' responses. Finally, in the previous chapter, we developed an expectation model that evaluates plans and estimates their expected correctness and F1-score. The final piece that completes the optimization puzzle is the prediction model.

Some plans incur more monetary costs. In return, they might provide more certainty in their results. By spending more money and asking from more workers using various access paths, we can ensure better quality answers. However, the budget is likely to be limited, and it is the system's task to produce the best possible result that can be afforded with the budget. On the other hand, the user may require a certain level of quality, and the query optimizer must find the cheapest ways to satisfy the requirement.

In this section, we will introduce models that help the system address the trade-off between cost and quality. First, we introduce a brute-force approach, that exhaustively search for the best plan. Although it is not efficient, but it produces the best possible result. Hence, we will consider it as a baseline for our greedy algorithm that will be presented in section 2.4.2.

2.4.1 Brute-force Approach

Brute-force algorithms consist of generating all possible candidates for the solution and checking whether each one of these candidates satisfies the requirements specified by the problem statement. Among other algorithms to solve problems,

they are usually the most straightforward and general problem-solving techniques that are merely a direct translation of the problem statement to a simple solution.

We first focus on the optimization with a constraint on the budget. Let B denote the budget. A brute-force approach for this optimization problem consists of enumerating all possible candidate plans that can be afforded by the budget, iterating over these candidates and choosing the plan that ensures the highest expected result quality.

Algorithm 1 shows the brute-force approach for the constrained-budget optimization. It takes as input the user's budget and the set of the access paths that are applicable to the query. Then, it enumerates all possible plans using `GenerateCandidatePlans` function. For each candidate, it calculates the expected quality using `CalculateExpectedQuality`. This function, which is simply our *expectation model* described in section 2.3, calculates the correctness and F1-score. The expectations are then stored in variable q, and is compared with that of the best plan the algorithm has so far. If the current plan has a higher quality, then the best plan is replaced by the current plan. After checking all candidates, the algorithm returns the best plan that has the highest expected quality.

Algorithm 1 BruteforceConstrainedBudgetOptimization(*budget*, *accesspaths*)

```
1: plans ← GenerateCandidatePlans(budget, accesspaths)
2: bestplan ← ∅
3: for each plan ∈ plans do
4:    q ← CalculateExpectedQuality(plan)
5:    if q > bestplan.quality then
6:       bestplan ← plan
7:    end if
8: end for
9: return bestplan
```

Example 10. *Assume that our budget is three ($B = 3$), and we have the set of access paths in table 2.3. The output of* `GenerateCandidatePlans` *function is:*

$$\{\{(a_1,1),(a_2,0),(a_3,0)\},\{(a_1,2),(a_2,0),(a_3,0)\},\{(a_1,3),(a_2,0),(a_3,0)\},$$
$$\{(a_1,0),(a_2,1),(a_3,0)\},\{(a_1,1),(a_2,1),(a_3,0)\},\{(a_1,0),(a_2,0),(a_3,1)\}\}$$

For example plan $\{(a_1,1),(a_2,1),(a_3,0)\}$ specifies that one worker must be asked using the first and second access paths.

We now change our focus from budget-constrained to quality-constrained optimization. The simplest way to do this optimization would be to start from $B = 1$, check whether it satisfies the quality requirement, if not, increment the budget by one until it meets the requirement. Although this gives us the best plan with minimum budget for the user-specified quality, it is not efficient at all.

The reason lies in the fact that spending more budget always means obtaining results with better quality. Therefore, instead of incrementing the budget one by one, we can apply a binary search algorithm to find the minimum budget for the required quality level.

Algorithm 2 depicts the use of binary search to find the minimum budget at which the user-specified quality can be satisfied. The constant `MAXBUDGET` must be set by the system developer. It can also be asked from the user, as a cost upper bound that he can afford. At each iteration, the algorithm checks whether there is any plan that satisfies the quality requirement. If such plan exists, it sets the flag to true, and at the next iteration repeats the process on the lower half . If no such plan exists, the flag remains false, and the upper half will be examined.

Algorithm 2 BinarySearchQualityConstrainedOpt(*quality*, *accesspaths*)

```
1: minbudget ← 0
2: maxbudget ← MAXBUDGET
3: while minbudget < maxbudget do
4:    mid ← (minbudget + maxbudget)/2
5:    plans ← GenerateCandidatePlans(mid)
6:    flag ← false
7:    for each plan ∈ plans do
8:       q ← CalculateExpectedQuality(plan, paths)
9:       if q > quality then
10:         flag ← true
11:      end if
12:   end for
13:   if flag then
14:      maxbudget ← mid
15:   else
16:      minbudget ← mid + 1
17:   end if
18: end while
19: return maxbudget
```

The brute-force algorithm allows the system to control the cost and quality of the final plan. Its concept is very simple to understand and implement. Because it exhaustively checks all possible options, the final plan that is recommended by this method is the best possible plan. Nevertheless, this approach is not very practical, especially when the size of our problem grows. In the context of crowd-enabled databases, the size of the problem depends on many factors: the complexity of the query (joins, sorts, complex conditions, ...), the size of the database, the number of access paths and also the number of the users. We need algorithms and heuristics that help the system prune the search space.

In the next section, we will propose a greedy algorithm that uses some heuristics to improve the exhaustive search. Since the brute-force algorithm

returns the best plan, we will use it as the baseline for our comparisons.

2.4.2 Greedy Optimization

As we argued in the previous section, the brute-force algorithm lacks efficiency and scalability, which makes it almost impractical for complex problems.

The system may prefer finding approximately optimal solutions in a reasonable time rather than exhaustively searching the whole search space with the hope of finding the best plan.

In this section, we propose a greedy algorithm that uses simple yet effective heuristics to prune the search space drastically. The brute-force algorithm consists of two levels of enumeration. At the first level, it enumerates all possible candidate plans. At the second level, for each candidate plan, the expectation function (specified by `CalculateExpectedQuality` in algorithm 1) enumerates all possible workers' responses for the access paths.

The greedy algorithm attempts to improve the first enumeration level. Instead of finding the global optimum plan for a given budget, it starts with a much lower budget and builds the local optimal plan at this stage. Like the brute-force algorithm, it then increments the budget. However, instead of generating all possible plans for the new budget, it only searches for the solution around the previous locally optimal solution.

Let us formally specify the heuristic. Assume that with budget b_i, the algorithm's choice is l_i. At the next stage, the greedy algorithm increments the budget to $b_i + s$. Instead of searching the whole space and checking all candidate plans that are affordable with $b_i + i$, it fixes l_i and attempts to explore only those plans that are the products of spending x on the existing l_i. Therefore, at each stage, it finds the locally optimal plan. At completion, it might not provide the global optimum plan. Nonetheless, its result is a fair approximate of the optimum plan produced in a more reasonable time.

Algorithm 3 shows the greedy approach. Instead of incrementing by one, it increments the current spent budget by `stepLength`, which is initialized at the beginning of the algorithm. For example, it can be initialized to the cost of the most expensive access path. Then at each iteration, it builds the candidate plans for `stepLength`, and aggregates these plans with the `currentBestPlan`. The rest is similar to algorithm 1. The lines 14 to 16 arrange the residual budget (that is less than the current `stepLength`), if any, to be spent at the next step of the algorithm.

Two potential problems might occur in the greedy approach specified in algorithm 3 that might cause the algorithm to fail to provide a reasonable solution. Both problems originate from the general properties of greedy algorithms.

The first problem is rooted in the *short-sighted* property of greedy algorithms. The proposed method cannot see far ahead of its current stage. There might be better candidate plans in a branch of the search space tree, but they do not find the chance to get explored. The first few iterations of the algorithm, in particular, can be very determinant. If these steps are taken steadily, it may lead the algorithm in a right direction, and hence, may ensure a quality result.

Algorithm 3 GreedyOptimization(*budget*, *accessPaths*)

```
1: spent ← 0
2: currentBestPlan ← ∅
3: stepLength ← InitializeStepLength(accessPaths)
4: while spent + stepLength ≤ budget do
5:    spent ← spent + stepLength
6:    temp ← GenerateCandidatePlans(stepLength)
7:    newPlans ← AggregatePlans(currentBestPlan, temp)
8:    for each plan ∈ newPlans do
9:       q ← CalculateExpectedQuality(plan, accessPaths)
10:      if q > currentBestPlan.quality then
11:         currentBestPlan ← plan
12:      end if
13:   end for
14:   residual ← budget − spent
15:   if residual > 0 and residual < stepLength then
16:      stepLength ← residual
17:   end if
18: end while
19: return currentBestPlan
```

The second problem relates to *non-recoverable* property of these algorithms. There is no turning back when taking a branch of the decision tree. Therefore, the algorithm can easily be trapped in local optima, that might be far from the globally optimum solution.

Chapter 3

Experiments and Results

Various simulations were performed to determine the effectiveness of employing access paths in a crowdsourcing database and the results are presented in this chapter. First we will overview our methodology and the settings in which the simulations were conducted. Section 3.2 summarizes the results of a set of comparisons of the performance of employing various decision-making functions. In section 3.3, we will discuss the results of two proposed prediction algorithms, i.e. brute-force and greedy methods. Based on these simulations, a summary of the observations is presented in section 3.4.

3.1 Methodology

In a relational DBMS, the query optimizer attempts to solve the optimization problem by finding the most efficient way to execute a query. We have defined the optimization problem for crowd-enabled databases using the metrics of cost and quality.

In this thesis, our focus is mainly on two optimization problems, namely budget-constrained and quality-constrained optimizations. In the former, the task of the query optimizer is to search for plans whose cost are within the given budget constraint and choose the one which provides the highest quality. In the latter, the user requires a specific level of quality, and the optimizer must provide the cheapest plan that satisfies this requirement.

However, the variables that play role in the optimization problem are not only limited to budget and quality. Unlike traditional databases, access paths are not deterministic. The number of access paths and their properties, such as their error rates, domain size and error overlap size are also important.

On the other hand, we discussed in section 2.4 that even though the exhaustive approach produces the optimal solution, it is not very scalable as the optimization problem and its variables grow in size. We presented a greedy algorithm that using simple heuristics, prunes the search space and therefore takes less time to find the solution. However, its solution is not guaranteed to

Access path	Cost	Error rate	Domain size	Error Overlap Size
Web	1	0.35	5	2
Ask	2	0.20	5	2
Call	3	0.15	5	2

Table 3.1: The access paths and their associated information used in our simulations

be globally optimal.

In this chapter, we investigate the effect of many of these factors on the quality of results. More specifically, we wish to scrutinize and understand the interplay of these factors with each other and with various decision-making functions and prediction algorithms.

In section 3.2 our focus will be on comparison of various decision-making functions based on varying properties of the problem, such as budget, required quality, domain size and error overlap size. Since we want to make sure that the results are valid, we use the exhaustive algorithm as the prediction model for the experiments. We will examine the greedy algorithm and compare it to its brute-force counterpart in section 3.3.

We assume that we have the set of access paths specified in table 3.1 that can be applied to the query. We are not willing to go into the query detail. From our perspective, the user issues a query, and the access paths in table 3.1 can be explored in order to obtain the answer. The error overlap size of the access paths is two. We assume that error overlap size is not pair-wise between each two access paths, and is an attribute of all access paths.

We will, of course, vary the parameters of the access paths in our simulations to observe their effects on the quality metrics. However, unless stated otherwise, their value will be according to the data in table 3.1.

3.2 Decision-making Models Comparison

3.2.1 Quality based on Varying Budget

We argued in section 2.1 that by exploring various paths, compared to relying only on one source, the system may give answers with more quality. As discussed in section 2.3, there exists two different objective functions that determine the quality of the answer. In *correctness objective funciton*, the goal is to maximize the correctness of the answers returned by the system, whereas in *F1-score objective function*, the goal is to maximize F1-score of the answers. We will first present the results for the first objective function. Then, our experiments and their results for F1-score will be discussed.

We will refer to plans that only use a single access path as *pure plans*. For example, plan *ask* is only allowed to use "ask a friend" access path. Plans that use various access paths will be referred as *mixed plans*. A mixed plan specifies

the number of questions that must be asked using each access path.

Pure Plans vs. Mixed Plans – Correctness Objective Function

In the first experiment, we measured the quality of the plans with limits on the budget for the correctness objective function.

Figure 3.1 shows the correctness and completeness based on varying budget for three pure plans (*web*, *ask* and *call*), and a mixed plan. We used the majority vote method for aggregating responses.

In figure 3.1a, as the budget increases, more questions could be asked, and therefore higher correctness is achieved. This result reiterates our claim that using mixed plans results in a better quality. The *call* access path, even though is the most trustable one, performs worse than the others, because it has the highest cost and therefore can afford asking fewer questions.

Spending more budget does not guarantee more complete answers. In fact, figure 3.1b suggests that increasing the budget sometimes results in sharp declines in the completeness. This means that by spending more money, the user might obtain more undecided results from the system. More importantly, the mixed plan is not the best plan in terms of completeness. The reason is that the goal in the correctness objective function is to maximize the correctness, and not completeness. Therefore, if the completeness is also of importance to the user, he/she should not use this objective function.

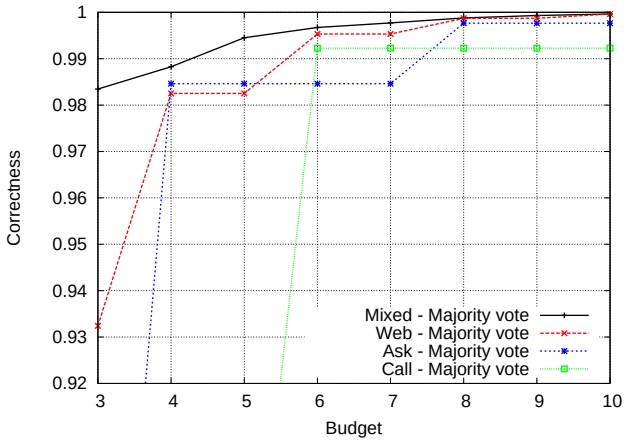
Comparison of Decision-making Methods – Correctness Objective Function

The purpose of the next experiment was to study the quality of the results obtained from different decision-making methods. The majority and frequency vote only take into account the number of votes on each answer, while the likelihood vote bases its decision also on properties of the access paths.

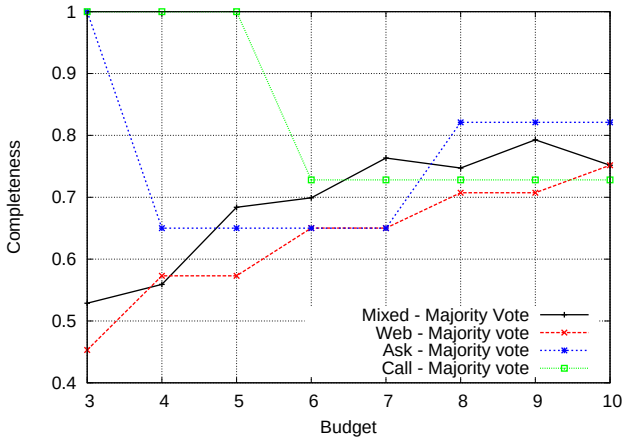
Figure 3.2 shows quality obtained from different aggregation models. We used two values for the quorum used in the likelihood vote. Except for budgets below five, the majority vote outperforms the other methods. The figure also shows that the likelihood method with higher quorum has better results than the one with the lower quorum.

One might jump into the conclusion that the reason behind this is that the majority vote, and the likelihood vote with higher quorum, base their decisions on stronger evidence. For example, the majority vote would give undecided unless there is a vote on which the majority agrees, and therefore must have high undecided rate. If it was the case, we would see lower completeness for these two decision-making methods. However, figure 3.2b shows that the majority vote, compared to the frequency and likelihood vote with $quorum = 0.1$, has better completeness for many budgets.

Figure 3.3 allows us to have a closer look at this issue. It shows the correctness and completeness for various values of quorum for the likelihood method. For example, for $budget = 10$, the curve associated to the likelihood vote with



(a) Correctness based on varying budget

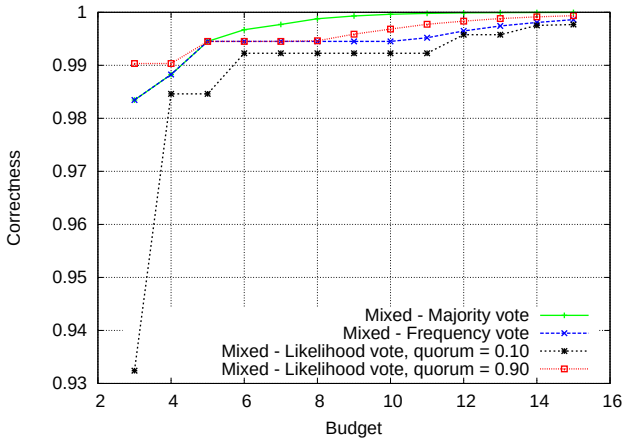


(b) Completeness based on varying budget

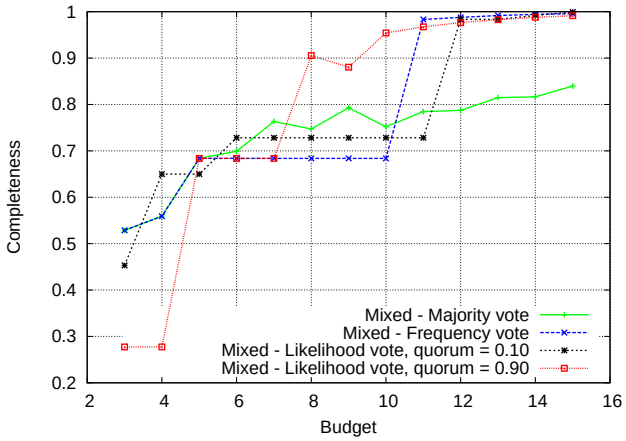
Figure 3.1: Quality metrics based on varying budget for the completeness objective function

quorum of 0.90 has higher correctness and completeness, compared to the likelihood vote with quorum of 0.10. For the ease of reference, we will refer to these decision-making functions as A and B, respectively.

This behaviour can be explained by the fact that for $budget = 10$, B and A choose different plans as the best plan. But why does B choose another plan, and not the same plan A did? The correctness objective function specifies that the best plan is the plan with the highest correctness. According to figure 3.3, the best plan chosen by A is not the best plan for B. This implies that plans are not inherently superior to each other. While one plan is more suitable



(a) Correctness based on varying budget

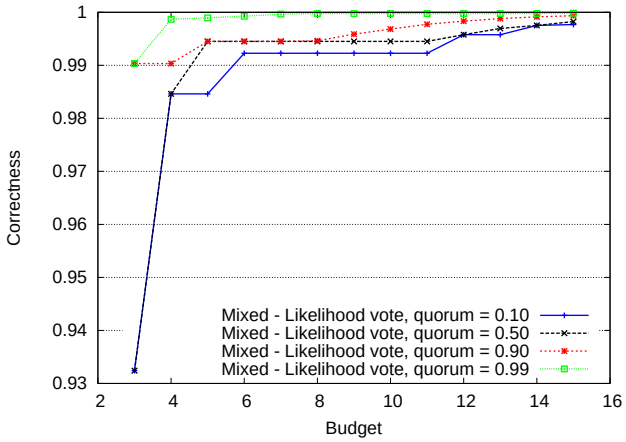


(b) Completeness based on varying budget

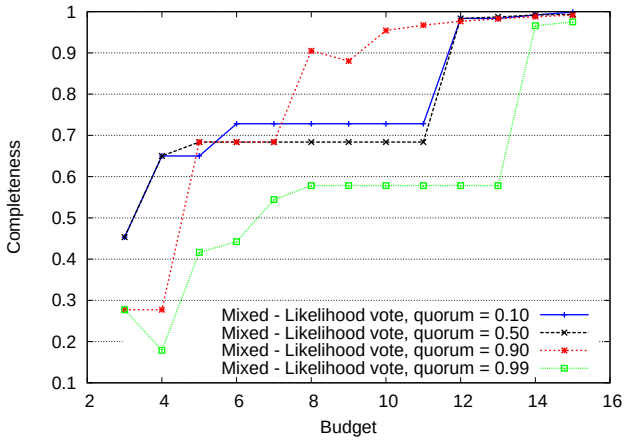
Figure 3.2: Quality metrics based on varying budget for different decision-making methods for the correctness objective function

for a decision-making function, another plan can provide better answers when another method of aggregation is used.

Figure 3.4 helps us understand the reason. While plan one is the optimal plan for the likelihood vote with $quorum = 0.10$, the second plan offers better correctness for the likelihood vote with $quorum = 0.90$.



(a) Correctness based on varying budget



(b) Completeness based on varying budget

Figure 3.3: Quality metrics based on varying budget for various quorums of likelihood vote for the correctness objective function

Pure Plans vs. Mixed Plans – F1-score Objective Function

In the previous section, we studied the budget-constrained optimization for the correctness objective function. We will now investigate the same problem, but this time for the F1-score objective function. F1-score is another measure of performance evaluation. It is simply the harmonic mean of correctness and completeness. Therefore, it attempts to strike a balance between these two metrics.

Figure 3.5 presents the comparison of pure plans with mixed plans. The

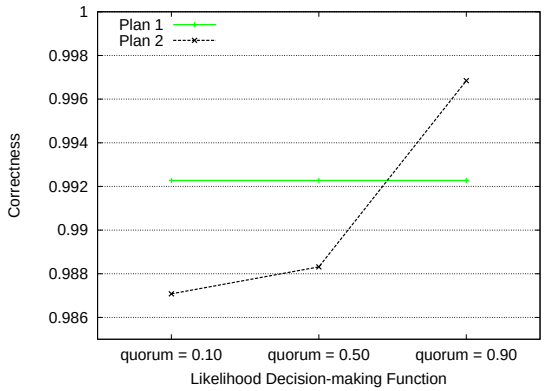


Figure 3.4: Correctness of three likelihood votes for two plans, where *plan 1* and *plan 2* are the chosen plans by the likelihood vote with quorum of 0.10 and 0.90 in figure 3.3, respectively

likelihood vote was used for aggregating responses (because as will be shown in the next section, it is the best decision-making function for the F1-score goal). As is evident, the mixed plan outperforms all pure plans. However, as the budget increases, they tend to converge. This may suggest that exploring different access paths is more important when the user's budget is more limited.

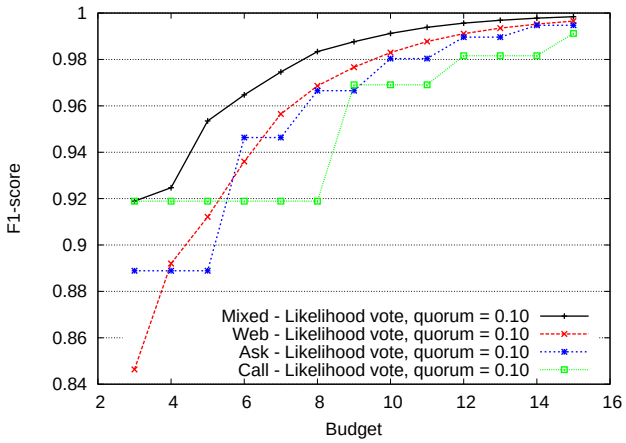


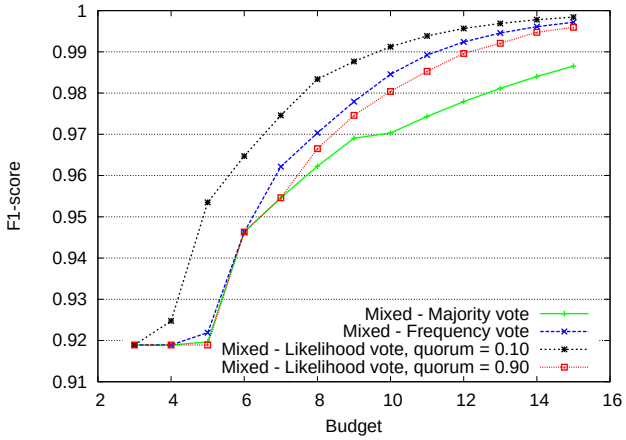
Figure 3.5: Quality based on varying budget for the F1-score objective function

Comparison of Decision-making Methods – F1-score Objective Function

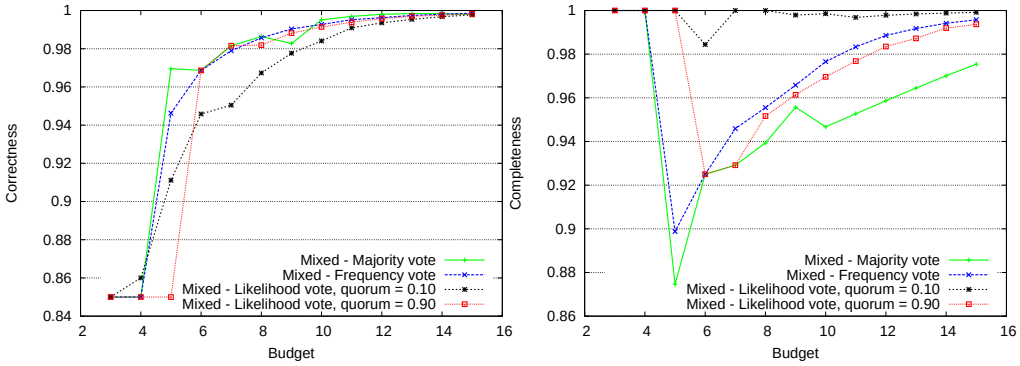
We have already observed that the majority vote seems to be the best option if the goal is to maximize the correctness. In this experiment, we want to

investigate what decision-making method performs better when maximizing F1-score is the objective function.

Figure 3.6a shows F1-score for different decision-making methods. It can be seen that the likelihood vote with the quorum of 0.10 gives the best result. Interestingly, the majority vote is outperformed by the others, especially at the higher budgets. The associated correctness and completeness are plotted in figures 3.6b and 3.6c. For example, the results show that although the correctness of the plans chosen by the majority vote is reasonable, but their poor completeness makes the majority vote undesirable. On the other hand, high completeness of the likelihood vote with the quorum of 0.1 compensates for its relatively low correctness, and in overall, making this aggregation method the best choice for the F1-score objective function.



(a) F1-score based on varying budget

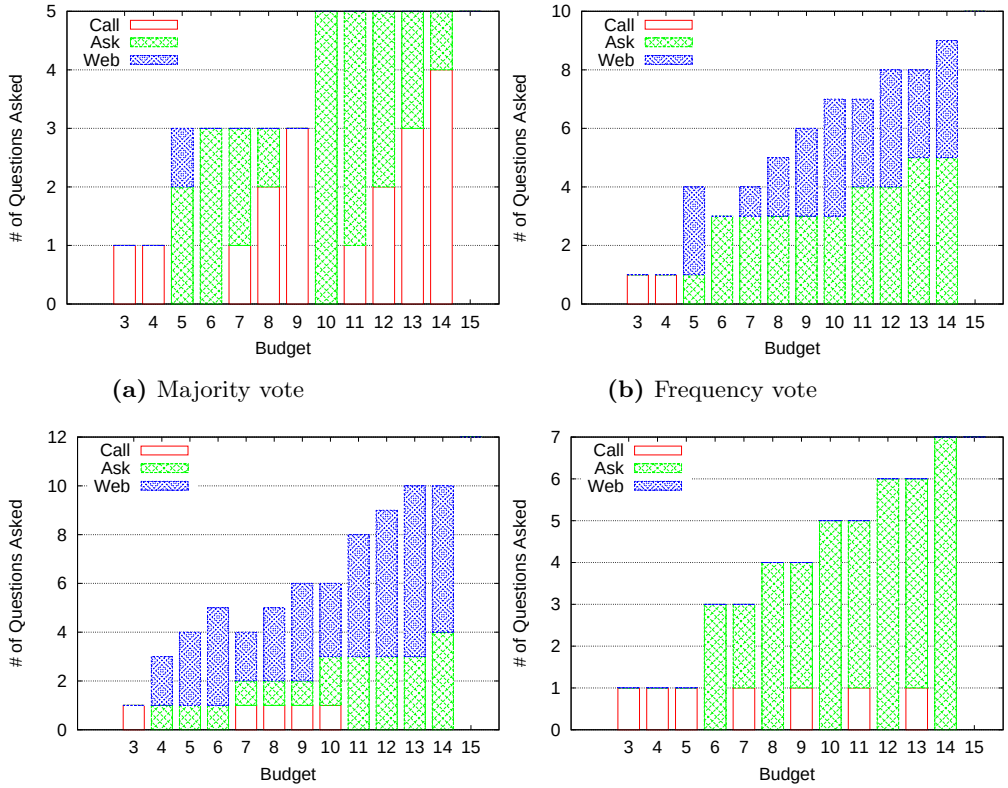


(b) Associated correctness

(c) Associated completeness

Figure 3.6: Quality based on varying budget for different decision-making methods for the F1-score objective function

The details of the chosen plans by the aggregation methods are illustrated in figure 3.7. The majority vote spends more budget on the *call* access path, and since this access path is expensive, this method can only afford asking a few number of questions. Asking fewer questions results in more undecided answers, and therefore low completeness. However, this access path is more trustable than the others. This explains why the majority vote performs well in the correctness, and poorly in the completeness. On the other hand, the likelihood vote with the quorum of 0.10 seems to spend its budget more evenly on different access paths. *Web* access path is asked the most number of questions. It is cheap, but less trustable. *Ask* is the second most popular access path, whose cost is between the cost of *web* and *call*.



(a) Majority vote

(b) Frequency vote

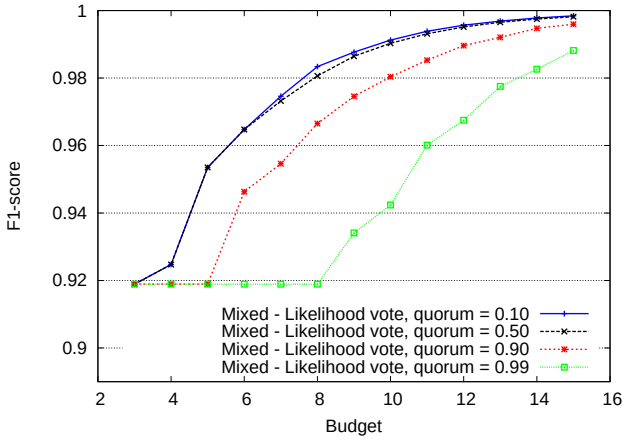
(c) Likelihood vote with quorum = 0.1

(d) Likelihood vote with quorum = 0.9

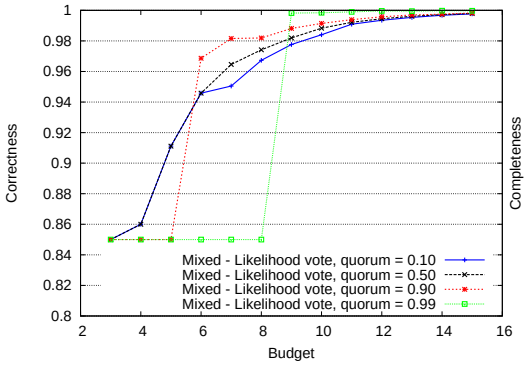
Figure 3.7: The number of questions asked by each aggregation method in figure 3.6a

With the help of figure 3.3, we found evidence that when the objective function is correctness, as the quorum increases, the likelihood method produces more correct results. Now let us examine the same for the F1-score objective

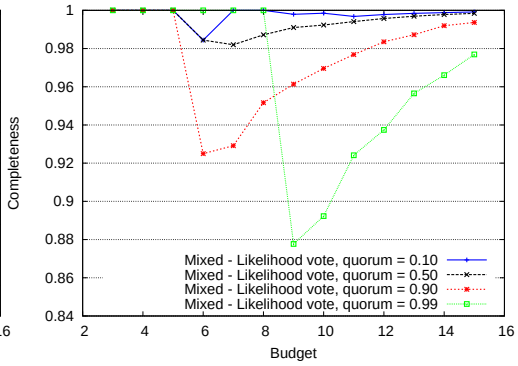
function. Figure 3.8 shows F1-score and its components (correctness and completeness) for various values of quorum based on varying budget. Here, unlike in the correctness objective function, as the quorum value increases, the metric of the quality (F1-score) declines. It can be explained by considering the other two plots. The curve for the quorum of 0.10, despite its poor performance in terms of correctness, gives better results than others in terms of completeness. Since F1-score attempts to consider both correctness and completeness, this quorum is a better choice than the others.



(a) F1-score based on varying budget



(b) Associated correctness



(c) Associated completeness

Figure 3.8: Quality metrics based on varying budget for various quorums of the likelihood vote for the F1-score objective function

3.2.2 Required Budget based on Varying Quality

In the previous section, we presented the results for the budget-constrained optimization. However, sometimes the budget is not the limiting factor and a

specific level of quality is required by the user. More precisely, the crowdsourcing system must be able to satisfy the quality-sensitive requirements of the user with as cheap plans as possible.

Similar to the previous section, we first consider the optimization problem for the correctness objective function. Afterwards, we present the experiments and their results for the F1-score objective function.

Figure 3.9 shows the minimum budget to satisfy the required quality for various aggregation methods when the objective function is correctness. As the required quality increases, all methods need more budget.

It is worth noting that until the correctness of 0.992, the likelihood performs equally or better than the majority vote. But after that, the majority vote requires less budget compared to the other methods. For example, in order to reach the correctness of 0.995, the majority vote costs six, while the frequency and the likelihood methods cost 11 and nine, respectively.

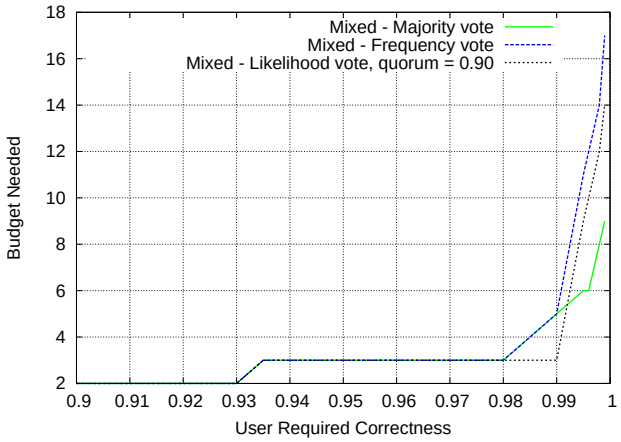
Let us now consider the quality-constrained optimization for the F1-score objective function. Figure 3.10 shows the required budget to satisfy the specified value of F1-score. Similar to our conclusions for the budget-constrained optimization for F1-score, it can be seen that the likelihood vote is the natural choice when the goal is to maximize F1-score. For example, the likelihood vote with $quorum = 0.1$ costs 10 to guarantee that the expected F1-score is more than 99%, while it costs 12 and 18 for the frequency and majority methods, respectively. The low correctness of the likelihood vote is compensated with its high completeness. This means that the likelihood vote gives less undecided results, but its answers are more likely to be wrong.

On the other hand, the frequency vote is only slightly worse than the majority vote in correctness, but because its completeness is remarkably higher, it requires significantly less budget to satisfy the F1-score quality requirements of the user.

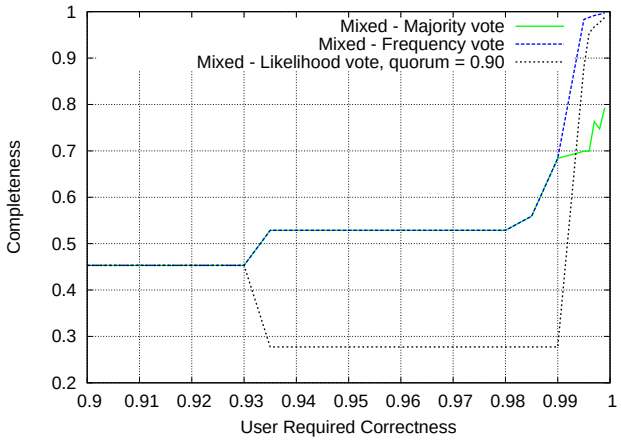
3.2.3 Quality based on Varying Access Paths Domain Size

We discussed in section 2.1.2 that answer domain size is a property of an access path. If we consider a worker's response as a variable, domain size of an access path simply equals the number of possible values that this variable can take on. Among these values, only one value is the correct answer, and the rest is wrong. Therefore, the access path's domain size is one more than the error domain size. Like the other properties, the domain size of access paths must be estimated by the application developer.

The domain size of an access path has two implications for the system. The first corresponds to the way we model human workers. Low value of domain size implies that workers who give wrong answers agree on a few values. Therefore, it becomes more difficult for the system to infer the correct answer. The second implication is on the technical level. In theory, more possible values for variables in an equation make solving it more difficult. Our optimizer is nothing more than an equation solver. Therefore, as the domain size of access paths increases, the complexity of the optimization becomes higher.



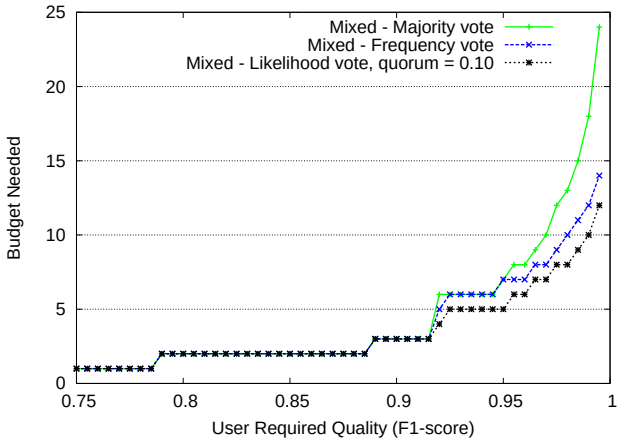
(a) Budget needed based on varying correctness



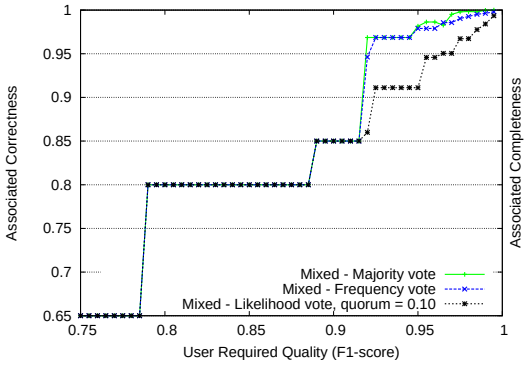
(b) Associated completeness for the chosen plans in 3.9a

Figure 3.9: Budget needed to satisfy the required quality for the correctness objective function

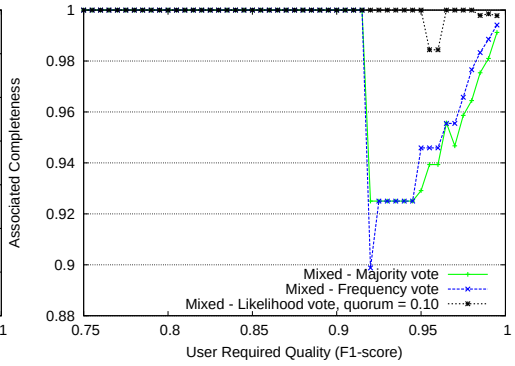
The purpose of this experiment was to study the effect of domain size on the quality of the aggregation methods. For simplicity of analysis, we varied the domain size of the three access paths together. In other words, in our experiment the value of the domain size represents the domain size of each access path. Since we wanted to focus on the effect of domain size, the error overlap size was kept intact (and equal to its default value, two) during the experiment. For a fixed budget of 10, we varied the domain size from three to 10, and measured the quality metrics. We will first present the results for the correctness objective function and then for the F1-score optimization.



(a) Budget needed based on varying F1-score



(b) Associated correctness for the chosen plans in 3.10a



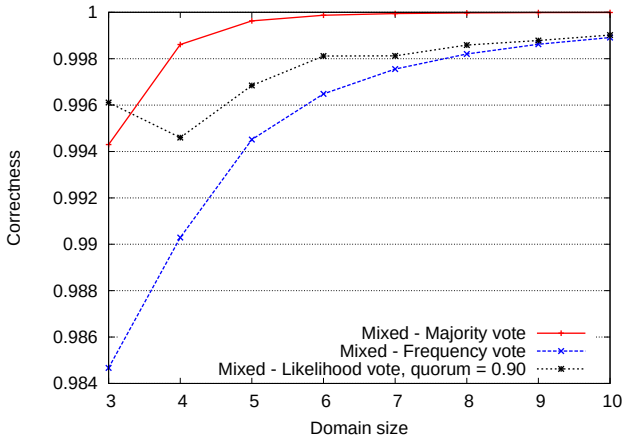
(c) Associated completeness for the chosen plans in 3.10a

Figure 3.10: Budget needed to satisfy the required quality in the F1-score objective function

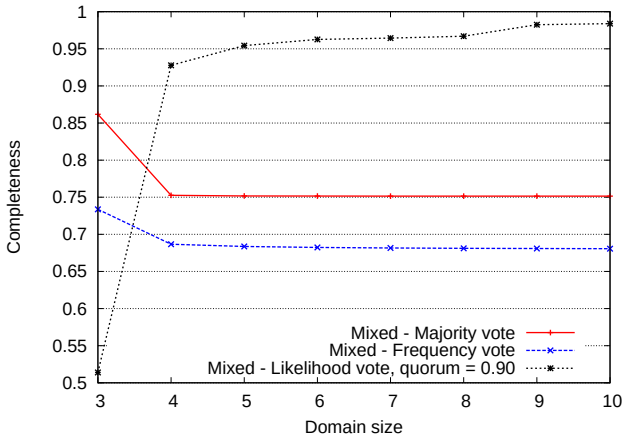
Figure 3.11 shows the correctness and its associated completeness for various values of domain size. As the domain size increases, all aggregation methods produce more correct answers. This confirms our initial hypothesis that low value of domain size results in low quality.

In addition, in all domain sizes, except three, the majority vote performs better than the other methods. Furthermore, it appears that increasing the domain size has negative effects on the completeness of the majority and frequency vote.

We now consider the same experiment for the F1-score objective function. Figure 3.12a shows F1-score for various values of domain size. The quality of the likelihood and frequency vote, as we could expect, increased. The majority vote surprisingly does not behave in the same way and slightly declines. Figures



(a) Correctness for various values of domain size



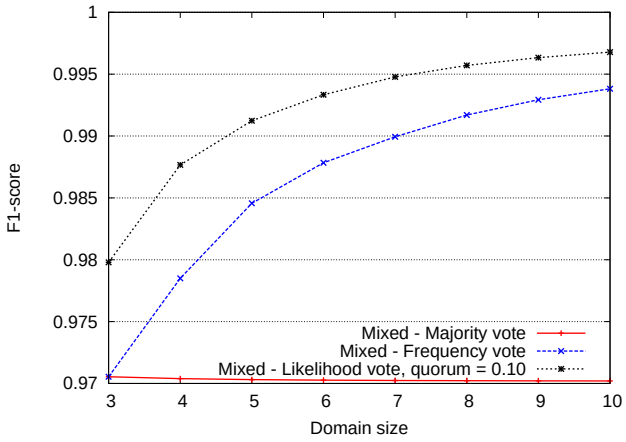
(b) Associated completeness for the chosen plans in 3.11a

Figure 3.11: Quality metrics based on varying domain size for the correctness objective function

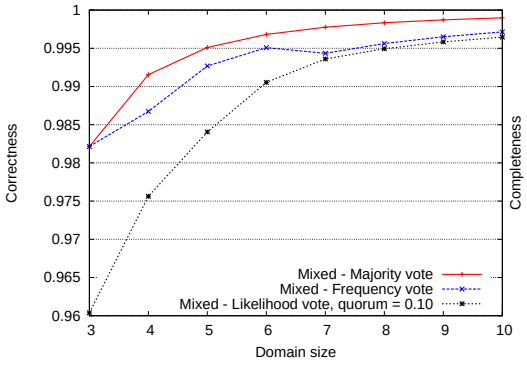
3.12b and 3.12c reveal the reason. Although increasing the domain size has a positive effect on its correctness, the decline in the completeness is so remarkable that the F1-score decreases.

3.2.4 Quality based on Varying Error Overlap Size

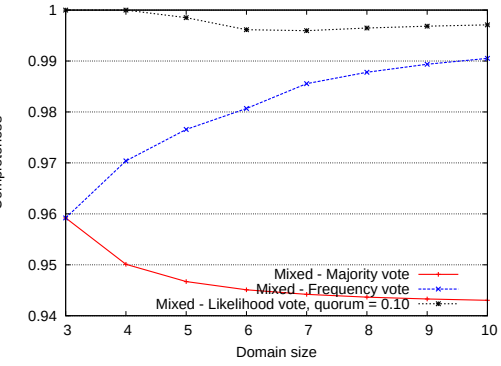
The error overlap size of a set of access paths is an indicator that represents the number of common wrong values that are shared between the access paths. In section 2.1.2, we assumed that error overlap size is not pair-wise between each



(a) F1-score for various values of domain size



(b) Associated correctness for the chosen plans in 3.12a



(c) Associated completeness for the chosen plans in 3.12a

Figure 3.12: Quality metrics based on varying domain size for the F1-score objective function

two access paths. It is an attribute of all access paths.

Error overlap size matters because it is a measure of correlation between the access paths. If it is zero, it means that errors of different paths have no correlation. In the absence of any correlation, it is significantly easier for the system to infer the correct result. This can be explained by the fact that in this case, the only answer on which these paths agree is actually the correct answer. At the other end of this spectrum, the full error overlap means that all paths produce the same wrong answers.

In this experiment, we studied the effect of error domain size on the quality of results. For a fixed budget of 10, we varied the error domain size of the access paths. In order to fully focus on its effects, we also fixed the domain size of all paths to five. Therefore, the range of possible values for the error domain size

is from zero (no overlap) to four (full overlap).

Figure 3.13 shows correctness for various values of error overlap size for the correctness objective function. It can be seen that as the overlap size increases, all three aggregation methods tend to produce less correct results. However, the decline in the majority vote is remarkably less than the decline in the other two methods. The majority vote has less tendency to distribute the budget over multiple paths (refer to 3.7a). As a result, increasing the over lap size only slightly affects the quality of the majority vote.

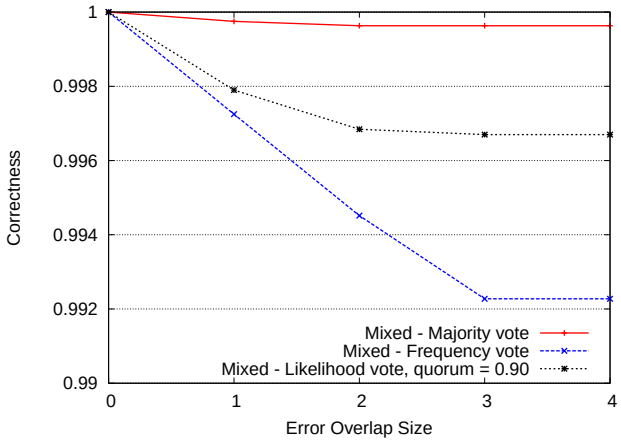


Figure 3.13: Correctness based on varying error overlap size

In the second part of this experiment, we studied the same problem for the F1-score objective function. You can see the results in figure 3.14. The results for this objective function are also consistent with our hypothesis. F1-score of the aggregation methods, except the majority vote, decreases as the overlap size increases.

When the access paths have more errors in common, less incentive will be left for the query optimizer to explore various access paths to get to the data. It is because of our fundamental assumption regarding the dependency of workers and their responses that we states in section 2.1. There, we assumed that errors are correlated within an access path, but independent across access paths. With the error overlap size as a variable, this dependency can be seen as a spectrum. Small value of error overlap size means that errors are less correlated across access paths, while high value indicates that errors are highly correlated.

Figure 3.15 illustrates the number of questions assigned to each access path for the frequency vote in figure 3.14. As the error overlap size increases, the frequency vote shows less tendency to use multiple access paths. For example, for no overlap, it makes use of all three access paths, while for full overlap, no incentive is left for the frequency vote to explore various paths, and only uses *web*.

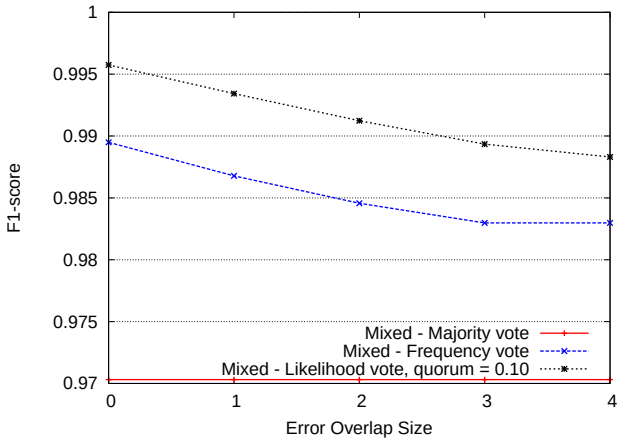


Figure 3.14: F1-score based on varying error overlap size

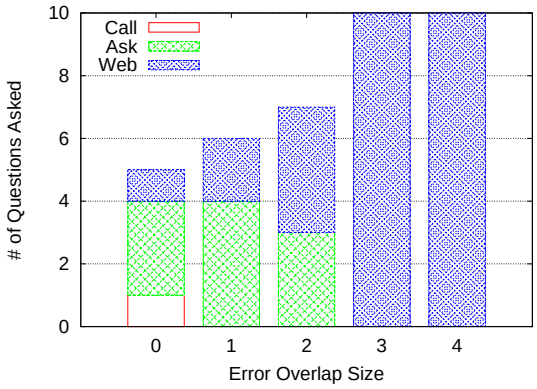


Figure 3.15: The number of questions asked by the frequency vote in figure 3.14

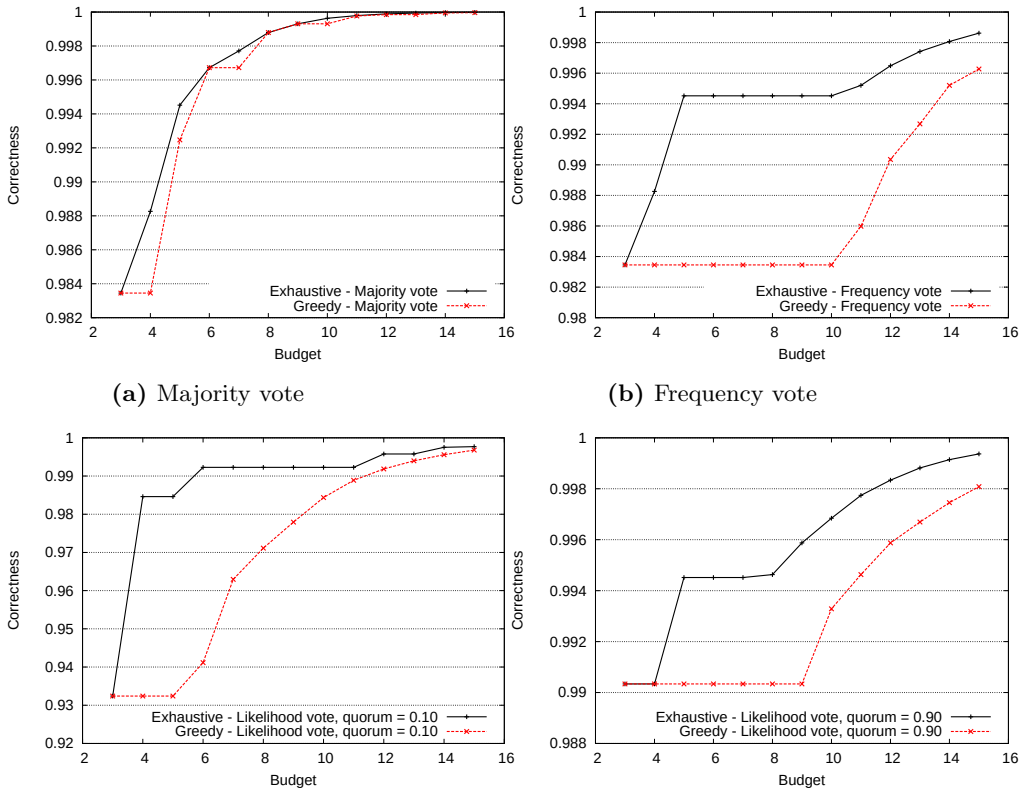
3.3 Prediction Models Comparison

In the previous section, we presented the results for various experiments using the exhaustive algorithm. Even though it guarantees that the produced result is optimal, due to its high resource requirements, it is not applicable in a real database system. Using the exhaustive method as the baseline, in this section we will examine the greedy algorithm.

We will first present the results for the budget-constrained and the quality-constrained optimizations. Then, the response time of two algorithms will be compared.

3.3.1 Budget-constrained Optimization

Each plot in figure 3.16 shows the comparison between the exhaustive search and the greedy algorithm for a specific decision-making method, where the objective function is to maximize correctness. In all plots, the exhaustive search outperforms the greedy algorithm. However, the greedy curve for the majority vote closely follows the exhaustive one. For the majority and the likelihood vote (with $quorum = 0.10$), it seems that algorithms finally converge.



(a) Majority vote

(b) Frequency vote

(c) Likelihood vote with $quorum = 0.10$

(d) Likelihood vote with $quorum = 0.90$

Figure 3.16: Comparison between the exhaustive (brute-force) search and the greedy algorithm based on varying budget for the correctness objective function

In the other two plots, until certain points (10 and nine for the frequency and the likelihood vote with $quorum = 0.90$, respectively), increasing the budget does not result in more quality. In fact, both plots illustrate the major pitfall of the greedy approach. This algorithm can easily become entrapped in a local optimum, that might be far from the globally optimum solution. This situation happens when none of the next steps of the algorithm is better that the current

stage. In other words, the algorithm has reached a local maximum in quality, and all of its adjacent points do not provide more quality. However, it seems that our proposed algorithm is able to escape local optima.

Figure 3.17 presents the summary of the quality of the aggregation methods for the greedy algorithm. Similar to our findings in 3.2.1, the majority vote seems to be the best method when the objective function is maximizing correctness.

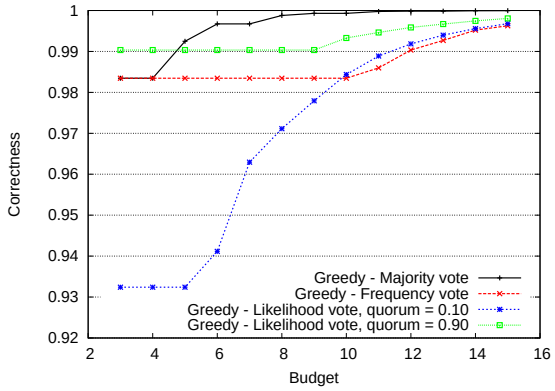


Figure 3.17: Comparison of the aggregation methods in figure 3.16 using the greedy algorithm

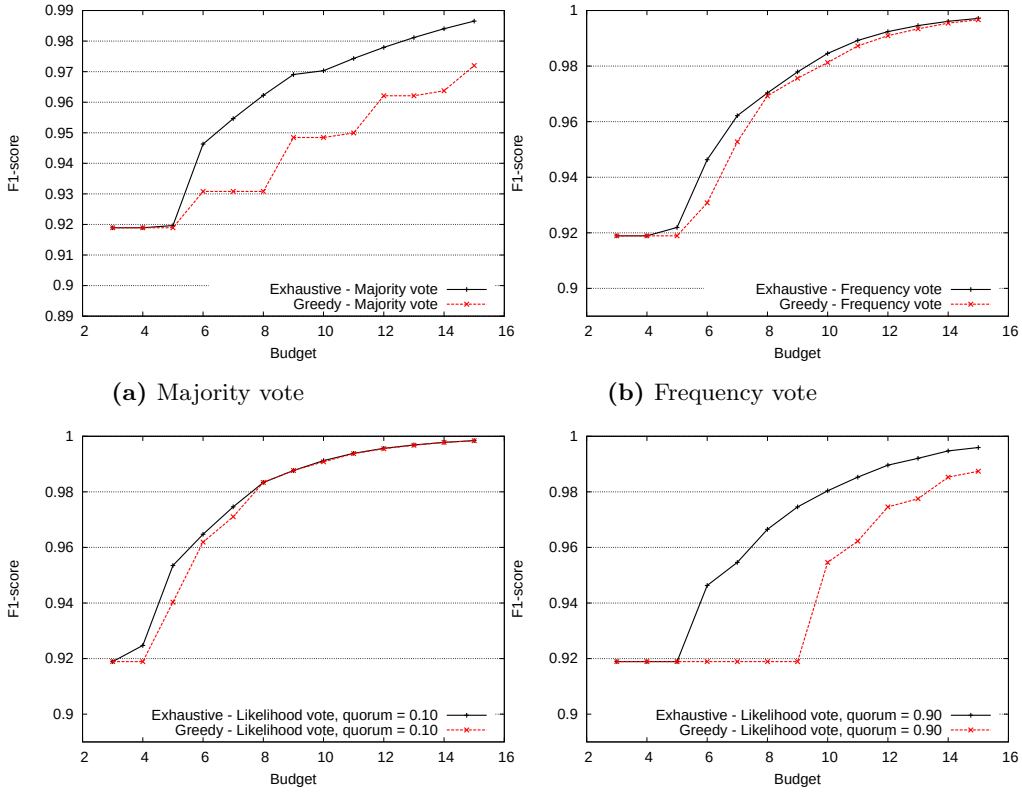
Figure 3.18 shows the results for the same experiment as in figure 3.16, but for the F1-score objective function. It can be seen that the performance of the greedy algorithm is very reasonable for the frequency and the likelihood vote with $quorum = 0.10$. For the latter, the quality values of two algorithms have only very slight differences.

Figure 3.19 shows the summary of the quality of the aggregation methods in figure 3.18. The likelihood vote with $quorum = 0.10$ is the best choice when the objective function is F1-score.

3.3.2 Quality-constrained Optimization

We only present the results for the best aggregation method for each objective function. Figure 3.20 shows the budget needed to satisfy the user required correctness, while figure 3.21 illustrates the minimum budget to satisfy various levels of F1-score. In both figures, the two algorithms finally converge.

However, before converging, the line which corresponds to the greedy algorithm is constantly higher than that of the exhaustive search. The reason for this initial phase can be explained by reviewing the pseudo code of the greedy approach which is detailed in algorithm 3. The starting point of the greedy algorithm (*stepLength*) is determined by `InitializeStepLength` function. We used the cost of the most expensive access path as the output of this function. Since *call* access path is the most expensive one with the cost of three, the



(a) Majority vote

(b) Frequency vote

(c) Likelihood vote with $quorum = 0.10$

(d) Likelihood vote with $quorum = 0.90$

Figure 3.18: Comparison between the exhaustive (brute-force) search and the greedy algorithm based on varying budget for the F1-score objective function

greedy algorithm starts from $stepLength = 3$, which can be seen in these two figures.

3.3.3 Algorithm Response Time

A real crowdsourcing database system may have to receive, process and answer to a large number of concurrent queries. Therefore, a complex database algorithm may need to keep a balance between the results optimality and the amount of time (or in a more general sense, system resources) it needs to respond to queries.

Both the exhaustive and greedy algorithm contain two nested iterative loops, one for permuting possible plans for a given budget and the other for iterating over all possible sets of votes for a given plan (refer to algorithms 1 and 3). The greedy algorithm, however, attempts to find a reasonable solution by using

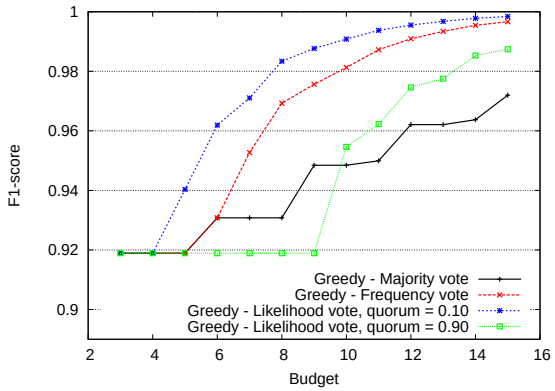


Figure 3.19: Comparison of the aggregation methods in figure 3.18 using the greedy algorithm

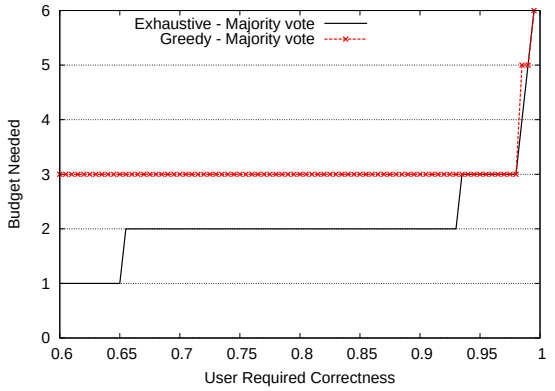


Figure 3.20: Comparison of the exhaustive search and the greedy algorithm in the quality-constrained optimization for the correctness objective function

some shortcuts and heuristics.

We measured the time for our experiments, and realized that greedy algorithm requires significantly less time to respond. For our measurements, we used a machine with the following hardware properties:

- **Processor**: Intel Core i5 CPU 2.67GHz, 4 Cores
- **Ram**: 8GB

All components of the optimizer and also the benchmark tool were implemented as single-threaded Java processes.

Figure 3.22a and 3.22b show the response time for the brute-force search and the greedy algorithm. The first figure show the time that the algorithms spent on the budget-constrained problem in figure 3.18c. The exhaustive algorithm takes

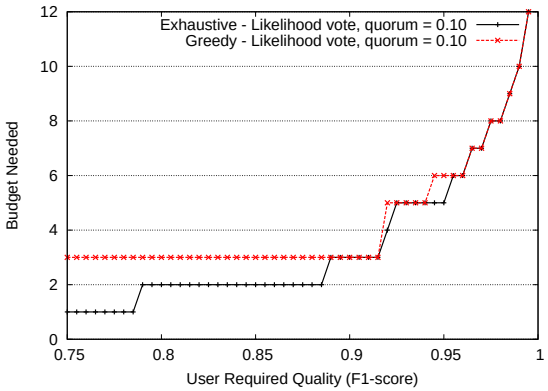
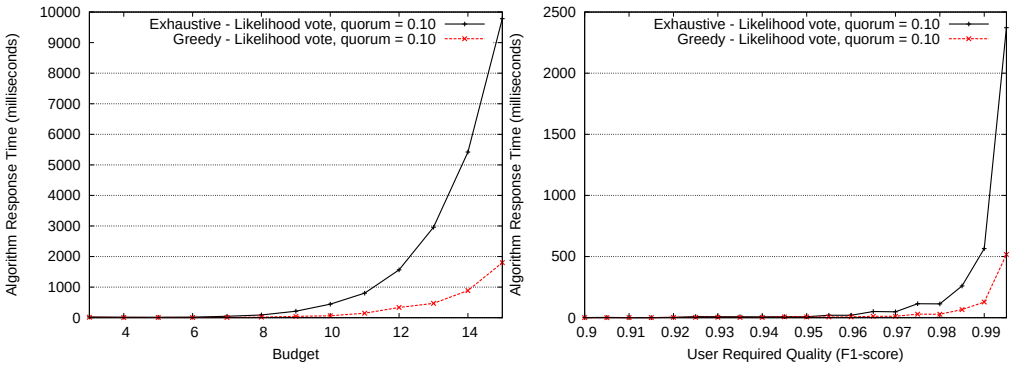


Figure 3.21: Comparison of the exhaustive search and the greedy algorithm in the quality-constrained optimization for the F1-score objective function

nearly fives time as long as the other method. However, spending more time does not help the exhaustive approach find remarkably better results. Figure 3.18c shows that the line associated to the greedy algorithm closely follows its counterpart.

On the other hand, 3.22b depicts the same comparison for the quality-constrained problem in figure 3.21. Again, the greedy algorithm is almost five times faster than the brute-force approach.

These results, although might change for different problem settings and may vary if the experiments are run on a different machine, give us an overall impression of the performance of the greedy algorithm.



(a) Response time for the budget-constrained optimization

(b) Response time for the quality-constrained optimization

Figure 3.22: Comparison of the response time for the exhaustive (brute-force) search and the greedy algorithm for the F1-score objective function

3.4 Summary

The experiments indicated that the system can benefit from exploring multiple access paths. By spending more budget, the system is able to reach higher quality in its results. We found that plans are not inherently superior to each other. Some plans produce better results when a particular decision-making method is adopted by the system. However, this does not guarantee that such plans give more quality if another aggregation method is used.

The experiments showed that there is no absolute best decision-making method. It is highly dependent on the choice of the objective function. Therefore, it is the users' requirement that determines which aggregation method should be used. For example, while the majority vote gives better results when the goal is maximizing correctness, for the F1-score objective function, the the likelihood vote with low quorum outperforms the other methods.

When the objective function is maximizing correctness, spending more budget does not guarantee more complete answers. If completeness is of importance to the user, F1-score should be used. The experiments indicated that F1-score succeeds to strike a balance between correctness and completeness.

The interplay of access paths' properties have significant effects on quality of results. We conducted some experiments on the domain size and the error overlap size, and found that as the domain size of access paths increases, it becomes easier for the correct answer to stand out and consequently be inferred by the decision-making method.

The experiments reiterated our hypothesis that the overlap of errors has a negative effect on the quality. The error overlap size is a measure of correlation between access paths. Zero overlap equals no correlation and complete independence, while full overlap indicates complete correlation. When access paths have fewer errors in common, the budget seems to be more evenly distributed among various access paths. On the other hand, if access paths have full overlap, there is no incentive left for the query optimizer to explore various access paths. In this case, it invests the budget only on one path.

We also observed that the greedy algorithm can give solutions that closely approximate the optimal solutions in a reasonable time. Our proposed algorithm was able to escape local optima and therefore skip a potential drawback of greedy algorithms.

Chapter 4

Related Work

4.1 Access Paths

For a given query, there are many plans that a traditional DBMS can follow and retrieve the answer. Access paths help the system choose the most efficient plan to retrieve the requested data [10]. Using these access paths, the query optimizer lays out all paths that can be applied to the query, examines all of them and selects the cheapest one. As mentioned in section 2.1, similar to our concept of access paths, they vary in their cost (in a RDBMS, running time), but unlike our model, in a traditional database they are equivalent in terms of output, as they all arrive at the same data, without any mistake. Almost all conventional databases leverage access paths to improve their query processing performances [27].

4.2 Crowdsourcing and Crowdsourcing Databases

Crowdsourcing continues to rise in popularity as an efficient means of mass problem-solving and collaboration. Wikipedia, Yahoo! Answers and StackExchange network are among the numerous Web 2.0 technologies that benefit from collaboration of thousands of users.

Crowdsourcing is not limited only to internet platforms and Q&A websites. In 2010, hundreds of fans recreated the popular science-fiction movie "Star Wars", where the whole movie was split into 15-second scenes, and each fan could choose one scene and recreate it [2]. It quickly became a hit on many video-sharing websites, and as of writing this report, it has been received almost three millions views on YouTube alone [1].

Crowdsourcing internet marketplaces, such as Amazon Mechanical Turk (AMT), facilitate employing large number of workers to perform small tasks, called Human Intelligence Tasks (HITs). CrowdFlower is a general purpose platform, developerd by Lukas Biewald and Chris Van Pelt, that pushes the submitted tasks to its partners, such as AMT.

The crowd power has been successfully used for a great number of applications that are very difficult or impossible for machines, including image labeling [31], audio transcription [36], text translation [37] and assessing search relevance in search engines [5].

Database management systems, despite their advances over the past few decades, are still unable to answer certain types of queries that is beyond what machines can do. For example, machines are unable to perform subjective comparisons, filling in missing data from the database, fuzzy matching and complex entity resolutions. One the other hand, enterprises that want to utilize human workers' capabilities in their applications might also need to leverage conventional DBMS functionality, stability and efficient management of their enormous amount of data.

There are a few systems that have been developed for this purpose. In hQuery [22], the query is formalized declaratively using a Datalog-like query model. Qurk [3, 19] is a query system for managing complex workflows that supports human-powered operations. In order to integrate SQL with AMT expressions, it allows the user to write UDFs (user defined functions) in his SQL query. CrowdDB is a DBMS that exploits the extensibility of conventional databases' features to add crowd functionality. For example, it extends SQL with some human-oriented operators. It also extends query processing engine and runtime system to allow for integration with crowdsourcing platforms. Deco [25] is another crowd-enabled DBMS that differs from CrowdDB in its data model. Data storage model is simpler in CrowdDB, as it only stores the cleansed data into the database and guarantees that the database always contains valid data. On the other hand, Deco stores the raw data, allowing future entity resolution as new data arrives. In addition, CrowdDB dictates stricter rules for the data model, similar to the conventional relational database management systems. In contrast, Deco is more flexible on the data model. For example, it allows for tables without primary key [24].

There have been several research works that investigate fundamental database algorithms in the context of crowdsourcing databases. CrowdScreen [21] proposes an algorithm to optimize the expected cost and quality in the problem of filtering a set of data items based on a set of properties. Sarma et al. [26] studied the same problem but with a limit on the number of items needed. Their model optimizes the problem with respect to the metrics of cost and latency, where latency is defined as the number of crowdsourcing phases. However, both of these two studies assume that all workers are equally likely to make errors. On a similar topic, Trushkowsky et al. [30] considered the problem of "Completeness" that arises in open-ended queries (such as SELECT COUNT(*)), where closed world assumption does not hold any more. In order to estimate the cardinality of the query, they applied species estimation algorithm that has been already studied in the field of Biology. Other human-assisted database algorithms include finding max [9], sorts and joins [4], graph search [23] and deduplication [8].

4.3 Crowdsourcing Optimization

Wais et al. [32] asserts that in crowdsourcing tasks, many workers do not contribute quality responses, and hence adequate quality control seems essential. They showed that for a labeling task, machine algorithms such as naïve Bayes classifier can outperform the crowd that exercises no such control.

In general, quality control approaches can be categorized broadly in two groups. First, the methods that base their judgements of correctness on a set of gold truth data. Second, the methods that attempt to infer workers' accuracies based on their historical performances.

There have been some research works on the first approach. Le et al. [16] proposed a quality assurance model which consists of manually annotation of gold standard sets and distributing these sets across the training data labels. They found that for achieving the highest utility, the optimal distribution of gold standard tasks is uniform distribution. Oleson et al. [20] pointed out that annotating labels with the described method requires extensive human effort to generate large collections of gold truth. For filling this gap and making gold-based quality assurance scalable, they proposed a model for generating the gold data automatically.

An extensive body of literature investigates the second approach of quality control (by inferring workers' accuracies). For example, some researchers have tried to have a closer look at workers' errors and mistakes. Ipeirotis et al. [11] presented a more reliable quality assessment by distinguishing between two types of errors: true intrinsic errors and biased errors. They argued that the latter form is recoverable and can be eliminated from the final result. Their proposed algorithm iteratively estimates workers' accuracies and correct answer simultaneously. However, their model provides no estimation for number of questions needed to be asked. In a similar direction, Snow et al. [28] presented a Bayesian algorithm to recover worker bias. Other researchers focused on uncertainty of estimated workers' accuracies by adding confidence intervals (as probabilistic guarantees) to their error rates and also to the final result [12].

Whitehill et al. [34] adopted a probabilistic decision-making method that is an extension to majority vote and attempts to learn characteristics of workers while asking questions and consequently filters responses of low-quality workers. Lin et al. [17] studied the same issue in the context of *open questions* (tasks with free-response formulation with infinitive size of possible responses, such as audio transcription) and proposed an algorithm based on Expected-Maximization that performs better than majority vote in those types of tasks.

There have been numerous studies on using the crowd in relevance assessments of search engines. For example, Kazai et al. [15] pinpointed the relevant factors that could be used to devise quality measures for search relevance assessments, such as familiarity of the worker with the content and length of the comment he leaves. Elsewhere, she highlights how monetary rewards, tasks difficulty and required qualifications for workers might affect the quality of the final result [14].

To achieve user's required quality with minimum cost, Karger et al. [13] de-

vised a set of techniques to identify which task should be assigned to which workers. CDAS [18], which is the work most closely to our likelihood decision-making model, is a quality-sensitive platform that provides an estimated accuracy for results based on workers' historical performances. Their model minimizes the cost by predicting how many questions must be asked in order to achieve a given quality. Although their probabilistic result evaluation model is similar to our likelihood vote, their prediction approach differs from that of ours. Due to the fact that Mechanical Turk does not allow for assigning tasks to workers with particular error rates, their estimation is based on the average of all workers' error rate.

In almost all the abovementioned studies, the cost of a query is proportional to the number of questions to be asked. Although this assumption is in line with the current AMT cost model, but it does not allow for rewarding more money to more expert workers that submit quality responses.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Since people are prone to errors, and unlike traditional information retrieval systems, there is no ground truth in crowdsourcing databases, the quality of the retrieved results must be rigorously controlled.

In real world problems, there usually exists various ways to arrive at the solutions. Some ways are more trustable, but may cost more. The crowdsourcing system must be enabled to encourage workers to explore different paths to reach the same information. We introduced the concept of access paths, and proposed an abstraction model for both representing and storing them. Our fundamental assumption states that people who answer a query using different access paths are independent of each other.

If crowdsourcing is seen as yet another method of information retrieval, two definitions for quality can be considered. In one definition, the quality is specified by the probability that the retrieved result is the latent correct answer. With this meaning, the system attempts to return those results which rely on strong evidence. Therefore, no penalty is imposed for returning no answer. The system prefers to stay undecided rather than to return a wrong answer. In the second definition, the system's goal is to strike a balance between correctness and completeness of answers.

We mainly focused on two optimization problems. In the budget-constraint case, the goal is to find the best plan that ensures the highest quality, whereas in the quality-constrained optimization, the user requires a specific quality for the obtained answer and the system must search for the cheapest plan that satisfies the requirement. Our experiments revealed a fundamental trade-off between monetary cost of plans and the quality they provide.

Our proposed optimizer consists of three sub-models. First, a decision-making model is employed to aggregate workers' responses. We investigated

three aggregation methods: the majority vote, frequency vote and likelihood vote. The majority and frequency vote consider only the number of votes on answers, whereas the likelihood vote uses a probabilistic approach to infer the correct result. The second component of the optimizer is the expectation model which evaluates plans based on a quality metric of the user's choice. The third component is the prediction model which attempts to find the plan whose cost and expected quality meet the user's requirements. For the prediction model, we proposed a greedy algorithm that, compared to an exhaustive search, is able to find approximate solutions in a reasonable time.

5.2 Future Work

In this research work, our focus was on finding the answer for a single attribute of a single tuple. Therefore, one avenue for the future work is to extend the model to incorporate specific database considerations. For example, the query model needs to deal with real complex queries such as join and sort.

While the theoretical aspect of access paths is the main topic of our work, encouraging people to explore diverse paths will be a challenge on its own. Some workers might take short-cuts and find the answer without using the specified path. Therefore, the future work in this area includes studying the mechanics of access paths and searching for ways to ensure that workers are encouraged enough to explore the access path to which they are assigned.

Another aspect of future work is to test and verify our findings in a real crowdsourcing system. Unless proven by real experiments, our hypotheses cannot be generalized and consequently, our model cannot be integrated into a database system.

Bibliography

[1] Star wars uncut. In *http://www.youtube.com/watch?v=7ezeYJUz-84*. Accessed: 5/10/2012.

[2] Million views for fans' star wars. In *http://www.bbc.co.uk/news/technology-16700913*, Jan. 2012. Accessed: 26/10/2012.

[3] A. M. 0002, E. W. 0002, S. Madden, and R. C. Miller. Crowdsourced databases: Query processing with people. In *CIDR*, pages 211–214. www.cidrdb.org, 2011.

[4] A. M. 0002, E. Wu, D. R. Karger, S. Madden, and R. C. Miller. Human-powered sorts and joins. *PVLDB*, 5(1):13–24, 2011.

[5] O. Alonso, D. E. Rose, and B. Stewart. Crowdsourcing for relevance evaluation. *SIGIR Forum*, 42(2):9–15, 2008.

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

[7] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. Crowddb: answering queries with crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, SIGMOD '11, pages 61–72, New York, NY, USA, 2011. ACM.

[8] P. Gulhane, R. Rastogi, S. H. Sengamedu, and A. Tengli. Exploiting content redundancy for web information extraction. *PVLDB*, 3(1):578–587, 2010.

[9] S. Guo, A. Parameswaran, and H. Garcia-Molina. So who won?: dynamic max discovery with the crowd. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 385–396, New York, NY, USA, 2012. ACM.

[10] Ioannidis. Query optimization. *CSURV: Computing Surveys*, 28, 1996.

[11] P. Ipeirotis, F. Provost, and J. Wang. Quality management on amazon mechanical turk. In *Proceedings of the ACM SIGKDD workshop on human computation*, pages 64–67. ACM, 2010.

[12] M. Joglekar, H. Garcia-Molina, and A. Parameswaran. Evaluating the crowd with confidence. Technical report, Stanford University, Aug. 2012.

[13] D. R. Karger, S. Oh, and D. Shah. Budget-optimal task allocation for reliable crowdsourcing systems. *CoRR*, abs/1110.3564, 2011.

[14] G. Kazai. In search of quality in crowdsourcing for search engine evaluation. In P. Clough, C. Foley, C. Gurrin, G. J. F. Jones, W. Kraaij, H. Lee, and V. Murdock, editors, *ECIR*, volume 6611 of *Lecture Notes in Computer Science*, pages 165–176. Springer, 2011.

[15] G. Kazai and N. Milic-frayling. On the evaluation of the quality of relevance assessments collected through crowdsourcing. In *In Proceedings of the SIGIR 2009 Workshop on the Future of IR Evaluation*, pages 21–22, 2009.

[16] J. Le, A. Edmonds, V. Hester, and L. Biewald. Ensuring quality in crowdsourced search relevance evaluation: The effects of training question distribution. In *SIGIR 2010 workshop on crowdsourcing for search evaluation*, pages 21–26, 2010.

[17] C. Lin, M. Mausam, and D. Weld. Crowdsourcing control: Moving beyond multiple choice. In *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[18] X. Liu, M. Lu, B. C. Ooi, Y. Shen, S. Wu, and M. Zhang. Cdas: a crowdsourcing data analytics system. *Proc. VLDB Endow.*, 5(10):1040–1051, June 2012.

[19] A. Marcus, E. Wu, D. R. Karger, S. R. Madden, and R. C. Miller. Demonstration of qurk: A query processor for human operators. Association for Computing Machinery (ACM), June 2011.

[20] D. Oleson, A. Sorokin, G. P. Laughlin, V. Hester, J. Le, and L. Biewald. Programmatic gold: Targeted and scalable quality assurance in crowdsourcing. In *Human Computation*, volume WS-11-11 of *AAAI Workshops*. AAAI, 2011.

[21] A. G. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom. Crowdscreen: algorithms for filtering data with humans. In K. S. Candan, Y. Chen, R. T. Snodgrass, L. Gravano, and A. Fuxman, editors, *SIGMOD Conference*, pages 361–372. ACM, 2012.

[22] A. G. Parameswaran and N. Polyzotis. Answering queries using humans, algorithms and databases. In *CIDR*, pages 160–166. www.cidrdb.org, 2011.

[23] A. G. Parameswaran, A. D. Sarma, H. Garcia-Molina, N. Polyzotis, and J. Widom. Human-assisted graph search: It's okay to ask questions. *CoRR*, abs/1103.3102, 2011.

[24] H. Park, H. Garcia-Molina, R. Pang, N. Polyzotis, A. Parameswaran, and J. Widom. Deco: a system for declarative crowdsourcing. *Proc. VLDB Endow.*, 5(12):1990–1993, Aug. 2012.

[25] H. Park, A. Parameswaran, and J. Widom. Query processing over crowdsourced data. 2012.

[26] A. Sarma, A. Parameswaran, H. Garcia-Molina, and A. Halevy. Finding with the crowd. 2010.

[27] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, SIGMOD '79, pages 23–34, New York, NY, USA, 1979. ACM.

[28] R. Snow, B. O'Connor, D. Jurafsky, and A. Y. Ng. Cheap and fast - but is it good? evaluating non-expert annotations for natural language tasks. In *EMNLP*, pages 254–263. ACL, 2008.

[29] J. Surowiecki. *The Wisdom of Crowds: Why the Many are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies, and Nations*. Doubleday, New York, 2004.

[30] B. Trushkowsky, T. Kraska, M. J. Franklin, and P. Sarkar. Getting it all from the crowd. *CoRR*, abs/1202.2335, 2012.

[31] L. von Ahn and L. Dabbish. Labeling images with a computer game. In *Proc. SIGCHI Conf. on Human factors in Computing Systems*, pages 319–326, 2004.

[32] P. Wais, S. Lingamneni, D. Cook, J. Fennell, B. Goldenberg, D. Lubarov, D. Marin, and H. Simons. Towards building a high-quality workforce with mechanical turk. *Proceedings of Computational Social Science and the Wisdom of Crowds (NIPS)*, pages 1–5, 2010.

[33] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: crowdsourcing entity resolution. *Proc. VLDB Endow.*, 5(11):1483–1494, July 2012.

[34] J. Whitehill, P. Ruvolo, T. Wu, J. Bergsma, and J. R. Movellan. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *NIPS*, pages 2035–2043. Curran Associates, Inc, 2009.

[35] T. Yan, V. Kumar, and D. Ganesan. Crowdsearch: exploiting crowds for accurate real-time image search on mobile phones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, pages 77–90, New York, NY, USA, 2010. ACM.

[36] C. ying Lee and J. R. Glass. A transcription task for crowdsourcing with automatic quality control. In *INTERSPEECH*, pages 3041–3044. ISCA, 2011.

[37] O. Zaidan and C. Callison-Burch. Crowdsourcing translation: Professional quality from non-professionals. In D. Lin, Y. Matsumoto, and R. Mihalcea, editors, *ACL*, pages 1220–1229. The Association for Computer Linguistics, 2011.