Managing the Quality of Crowdsourced Databases

by

YUDIAN ZHENG

A thesis submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy
at The University of Hong Kong

August 2017

Abstract of thesis entitled

"Managing the Quality of Crowdsourced Databases"

Submitted by

YUDIAN ZHENG

for the degree of Doctor of Philosophy
at The University of Hong Kong
in August 2017

Many important data management and analytics tasks cannot be completely addressed by automated processes. For example, entity resolution, sentiment analysis, and image recognition can be enhanced through the use of human input. Crowdsourcing platforms are an effective way to harness the capabilities of the crowd to apply human computation for such tasks. In recent years, crowdsourced data management has become an area of increasing interest in research and industry.

Typical crowd workers are often associated with a large variety of expertise, background, and quality. As such, the crowdsourced database, which collects information from these workers, may be highly noisy and inaccurate. Thus it is of utter importance to manage the quality of crowdsourced databases. In this thesis, we identify and address two fundamental problems in crowdsourced quality management: (1) *Task Assignment*, which selects suitable tasks and assigns to appropriate crowd workers; (2) *Truth Inference*, which aggregates answers obtained from crowd workers to infer the final result.

For the task assignment problem, we consider two common settings adopted in existing crowdsourcing solutions: task-based and worker-based. In the task-based setting, given a pool of *n* tasks, we are interested in which of the *k* tasks should be assigned to a worker. A poor assignment may not only waste time and money, but may also hurt the quality of a crowdsourcing application that depends on the workers' answers. We propose to consider evaluation metrics (e.g., Accuracy and F-score) that are relevant to an application and we explore how to optimally assign tasks in an online manner. In the worker-based setting, given a monetary budget and a set of workers, we study how workers should be selected, such that the tasks in hand can be accomplished successfully and economically. We observe that this is related to the aggregation of workers' qualities, and propose a solution that optimally aggregates the qualities from different workers, which is fundamental to selecting workers.

For the truth inference problem, although there exist extensive solutions, we find that they are not compared extensively under the same framework, and it is hard for practitioners to select appropriate ones. We conduct a detailed survey on 17 existing solutions, and provide an in-depth analysis from various perspectives.

Finally, we integrate the task assignment and truth inference in a unified framework, and apply them to two crowdsourcing applications, namely image tagging and question answering. For image tagging, where a worker is asked to answer the task, we select the correct label(s) among multiple given choices. We identify workers' unique characteristics in answering multi-label tasks, and study how it can help to solve the two problems. For question answering, where workers may have diverse qualities across different domains. For example, a worker who is a basketball fan should have better quality for the task of labeling a photo related to '*Stephen Curry*' than the one related to '*Leonardo DiCaprio*'. We leverage domain knowledge to accurately model a worker's quality, and apply them to addressing the two problems.

An abstract of exactly 492 words

Declaration

I declare that this thesis represents my own work, except where due acknowledgement is made, and that it has not been previously included in a thesis, dissertation or report submitted to this University or to any other institution for a degree, diploma or other qualifications.

. . . .
YUDIAN ZHENG
August 2017

List of Publications

1. [Research] * **Yudian Zheng**, Jiannan Wang, Guoliang Li, Reynold Cheng, Jianhua Feng.
QASCA: A Quality-Aware Task Assignment System for Crowdsourcing Applications.
In SIGMOD'15: Proceedings of the 2015 ACM International Conference on Management of Data, Pages 1031-1046, 2015.

2. [Research] * **Yudian Zheng**, Reynold Cheng, Silviu Maniu, Luyi Mo.
On Optimality of Jury Selection in Crowdsourcing.
In EDBT'15: Proceedings of the 18th International Conference on Extending Database Technology, Pages 193-204, 2015.

3. [Research] * **Yudian Zheng**, Guoliang Li, Reynold Cheng.
DOCS: Domain-Aware Crowdsourcing System.
In PVLDB'16: Proceedings of the VLDB Endowment VLDB Endowment, Volume 10, Issue 4, November 2016, Pages 361-372, 2016.

4. [Research] Zhipeng Huang, **Yudian Zheng**, Reynold Cheng, Yizhou Sun, Nikos Mamoulis, Xiang Li.
Meta Structure: Computing Relevance in Large Heterogeneous Information Networks.
In KDD'16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Pages 1595-1604, 2016.

5. [Research] Huiqi Hu, **Yudian Zheng**, Zhifeng Bao, Guoliang Li, Jianhua Feng, Reynold Cheng.
Crowdsourced POI Labelling: Location-Aware Result Inference and Task Assignment.

In ICDE'16: Proceedings of the 32nd International Conference on Data Engineering, Pages 61-72, 2016.

6. [Survey] Guoliang Li, Jiannan Wang, **Yudian Zheng**, Michael J. Franklin.
Crowdsourced Data Management: A Survey.
In TKDE'16: IEEE Transactions on Knowledge and Data Engineering, Volume 28, Issue 9, February 2016, Pages 2296-2319, 2016.

7. [Research] Xiang Li, Ben Kao, **Yudian Zheng**, Zhipeng Huang.
On Transductive Classification in Heterogeneous Information Networks.
In CIKM'16: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, Pages 811-820, 2016.

8. [Research] Zhipeng Huang, Bogdan Cautis, Reynold Cheng, **Yudian Zheng**.
KB-Enabled Query Recommendation for Long-Tail Queries.
In CIKM'16: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, Pages 2107-2112, 2016.

9. [Experimental Analysis] * **Yudian Zheng**, Guoliang Li, Yuanbing Li, Caihua Shan, Reynold Cheng.
Truth Inference in Crowdsourcing: Is the Problem Solved?
In PVLDB'17: Proceedings of the VLDB Endowment, Volume 10 Issue 5, January 2017, Pages 541-552, 2017.

10. [Tutorial] Guoliang Li, **Yudian Zheng**, Ju Fan, Jiannan Wang, Reynold Cheng.
Crowdsourced Data Management: Overview and Challenges.
In SIGMOD'17: Proceedings of the 2017 ACM International Conference on Management of Data, Pages 1711-1716, 2017.

11. [Research] Xiang Li, Yao Wu, Martin Ester, Ben Kao, Xin Wang, **Yudian Zheng**.
Semi-supervised Clustering in Attributed Heterogeneous Information Networks.
In WWW'17: Proceedings of the 26th International Conference on World Wide Web, Pages 1621-1629, 2017.

12. [Research] Guoliang Li, Chengliang Chai, Xueping Weng, Ju Fan, Jian Li, **Yudian Zheng**, Yuanbing Li, Xiang Yu, Xiaohang Zhang, Haitao Yuan.
CDB: Optimizing Queries with Crowd-Based Selections and Joins.
In SIGMOD'17: Proceedings of the 2017 ACM International Conference on Management of Data, Pages 1463-1478, 2017.

13. [Survey] Guoliang Li, Jiannan Wang, **Yudian Zheng**, Michael J. Franklin.
Crowdsourced Data Management: A Survey (Extended Abstract).
In ICDE'17: Proceedings of the 33rd International Conference on Data Engineering, Pages 39-40, 2017.

14. [Tutorial] Reynold Cheng, Zhipeng Huang, **Yudian Zheng**, Jing Yan, Ka Yu Wong, Eddie Ng.
Meta Paths and Meta Structures: Analysing Large Heterogeneous Information Networks.
In APWeb-WAIM'17: Proceedings of the Asia Pacific Web and Web-Age Information Management Joint Conference on Web and Big Data, 2017.

*Note: Publications marked by * are used in this thesis.*

Acknowledgements

I would like to thank my supervisor, Dr. Reynold Cheng, for all kinds of support during these years. Through and far beyond meetings and seminars, he has enriched my understanding towards research; and I truly believe that everything I have learnt from him will keep benefiting me in my future career. To him, I feel extraordinarily grateful.

I am truly thankful to Dr. Guoliang Li, Dr. Jiannan Wang, Dr. Yeye He, Dr. Cong Yu, Prof. Laks V.S. Lakshmanan, Prof. Ben Kao, Prof. Nikos Mamoulis, Prof. David W.-L. Cheung, Prof. Michael Franklin, Dr. Yang Yu, Dr. Silviu Maniu, Dr. Ju Fan, Dr. Yizhou Sun, Dr. Jian Li, Prof. Jianhua Feng, Prof. Zhihua Zhou, Dr. Zhifeng Bao, Dr. Xiaokui Xiao, Prof. Bogdan Cautis and Prof. Martin Ester for all the collaborations and instructions. They offered valuable comments and much help, making me a better researcher. I must also thank all the respectable anonymous reviewers who appreciated and criticized my work, for they strengthened the work in many aspects.

I appreciate all those friends that I met in University of Hong Kong, Tsinghua University, Microsoft Research and Google Research. Shoulder to shoulder, we back up each other; side by side, we move forward to the future. I feel so fortunate to be friends with these great people. I enjoyed very much those days we had together, and deep inside my heart the unforgettable memory shall be cherished forever.

My special thanks to my family, without whom I can never make it this far. My parents, i.e., Meirong Wang and Zhongping Zheng are always ready to support me, in every possible way they can. Wenshuo Chen, my dear, has all the way been understanding, encouraging, patient, and unconditionally supportive. To them, my deepest gratitude can be expressed by no word but only loving them the same way back with my heart and soul.

Contents

List of Figures

List of Tables

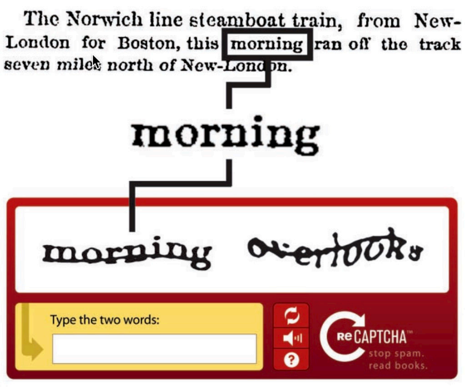
List of Algorithms

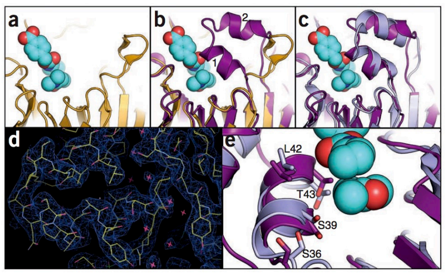
Chapter 1

Introduction

Existing algorithms often cannot effectively address computer-hard tasks such as entity resolution [35, 79, 139, 176, 189, 192, 195, 199], sentiment analysis [124, 125, 142, 222], and image recognition [172, 198, 205], which can benefit from the use of human cognitive ability. Crowdsourcing is an effective way to address such tasks by utilizing hundreds of thousands of ordinary workers (i.e., the crowd). Furthermore, access to crowd resources has been made easier due to public crowdsourcing platforms, such as Amazon Mechanical Turk (AMT) [1], CrowdFlower [5] and Upwork [10]. As reported by the AMT in August 2012, over 500K workers from 190 countries worked on tasks. The large number of workers and HITs have motivated researchers to develop solutions to streamline the crowdsourcing process (see survey [116], book [135], and tutorials [13,36,53,72,73,94,117]). Thus, due to the increasing interests, data management in crowdsourcing has become an active area in both industry and research.

There are many successful applications that utilize crowdsourcing to solve computer-hard tasks. For example, in Figure 1.1(a), Von Ahn et al. digitized printed material by getting Internet users to transcribe words from scanned texts [190]. Their method achieved accuracy exceeding 99% and has transcribed


The Norwich line steamboat train, from New-London for Boston, this morning ran off the track seven miles north of New-London.
morning
morning overlooks
Type the two words:
reCAPTCHA
stop spam. read books.
(a) reCAPTCHA [190]



(b) Game-Driven Crowdsourcing [61]

Figure 1.1: Crowdsourcing Applications.

over 440 million words. As another example, in Figure 1.1(b), Eiben et al. utilized a game-driven crowdsourcing method to enhance a computationally designed enzyme [61].

Crowdsourcing can also benefit data management applications, such as data cleaning [66, 152, 193], data integration [92, 122, 221], knowledge construction [14, 17]. Consider entity resolution as an example. Suppose a user (called the "requester") has a set of objects and wants to find the objects that refer to the same entity, perhaps using different names. Although this problem has been studied for decades, traditional algorithms are still far from perfect [139, 176]. Alternatively, s/he can harness the crowd's ability to identify the same entity. To this end, the requester first designs the tasks, and then publishes their tasks on a crowdsourcing platform such as AMT. Crowd workers who are willing to perform such tasks (typically for pay or some other reward) accept the tasks, answer them and submit the answers back to the platform. The platform collects the answers and reports them to the requester. As the crowd has contextual knowledge and cognitive ability, crowdsourced entity resolution can improve the quality [79, 186, 189, 192, 195].

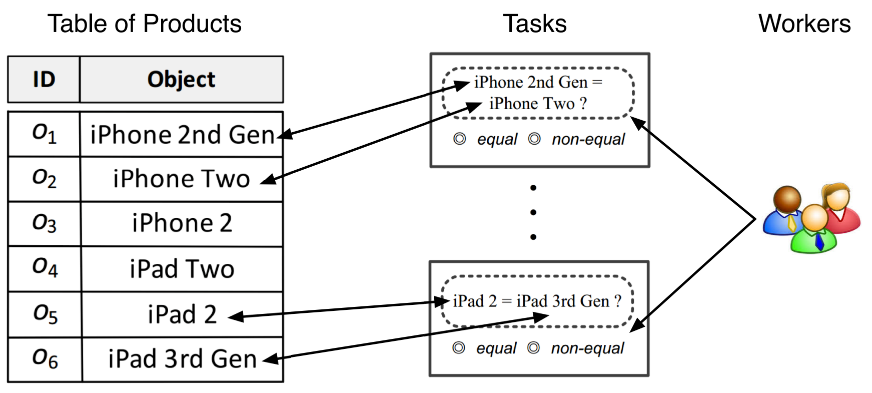


Figure 1.2: An Example Entity Resolution Application.

1.1 Crowdsourcing Workflow

In a crowdsourcing platform (e.g., AMT [1]), there are two types of users, called "*workers*" and "*requesters*", who will deal with tasks. Requesters publish tasks to the platform; Workers perform tasks and return the results. In the following, we describe the life-cycle of a task from the individual perspective of requesters and workers.

Figure 1.2 illustrates how the entity resolution application can be applied to the workflow of crowdsourcing. Given a table of products, suppose a requester aims to find the pairs of products that refer to the same real-world entity. The requester needs to first design the user interface of a task, e.g., each task contains a pair of products, and it asks the crowd to choose whether they are "*equal*" or "*non-equal*". For example, the first task contains two products, i.e., o_1: *iPhone 2nd Gen* and o_2: *iPhone Two*, and it asks workers whether the two products are "*equal*" or "*non-equal*". The requester also has to set up some properties of the tasks, e.g., the price of a task, the number of workers to answer a task, the time duration to answer a task[1]. After that, the requester publishes the tasks to the

[1]Note that the problem of how to set the properties of a task has been studied in existing works [116], and it is not the focus of the thesis.

platform, and collects the answers from the crowd.

From workers' perspective, they can browse and select the available tasks published by requesters. Often the unit that a worker interacts with the crowdsourcing platform is called a "Human Intelligent Task", or HIT, which contains a set of tasks. When accepting a HIT, they have to finish all the tasks in the HIT within the specified time duration. If a worker has accomplished a HIT, then the crowdsourcing platform will collect the worker's answers for the tasks in the HIT, and the worker will be paid by the specified monetary budget.

1.2 Crowdsourcing Framework

Based on the above discussions, we show the crowdsourcing framework in Figure 1.3. The requester will deploy tasks and budget to the crowdsourcing platform (e.g., AMT). The workers will come to interact with the crowdsourcing platform in two components:

(1) Task Assignment, which assigns tasks to workers. Since workers may have different backgrounds and diverse qualities on tasks, an intelligence task assignment algorithm will judiciously select suitable tasks to appropriate workers. Existing works study the task assignment problem by focusing on two settings, based on the perspectives of tasks and workers:

- Task-based Setting. In this setting, when a worker comes, it studies the problem that which subset of tasks should be assigned to the coming worker. It is often called "*Online Task Assignment Problem*" [85, 86, 138, 164, 222]. Existing platforms such as AMT supports assigning tasks in the task-based setting. To be specific, we could use the "external-HIT" way provided by AMT, which embeds the generated HTML pages by our server into its web frame and workers directly interact with our server through the web frame. Thus when a worker comes, we can identify the worker from the individual worker-id (a string of 14 characters) provided by AMT. Finally, we can dynamically batch the selected tasks in a HIT, and assign to the coming worker.

- Worker-based Setting. In this setting, given a task and a set of candidate workers, it studies the problem that which subset of workers should be selected to answer the task. It is often called "*Jury Selection Problem*" [33, 219]. With the increasing amount of data, large-scale Internet companies such as Google or Facebook requires human experts to label the data, e.g., they want to be sure whether there is a Starbucks coffee in a specific point of interest (POI) location. Workers with different qualities and budget requirements come to answer the task, and the problem aims at wisely selecting a subset of workers, such that the overall quality is maximized within a given budget constraint.

(2) Truth Inference, which collects workers' answers and infers the truth of each task based on considering all collected answers from workers. Note that workers may yield low quality or even noisy answers, e.g., a malicious worker will intentionally give wrong answers; workers may also have different levels of expertise, and an untrained worker may be incapable of accomplishing certain tasks. Thus, in order to achieve high quality, we need to tolerate crowd errors and infer high quality results from noisy answers.

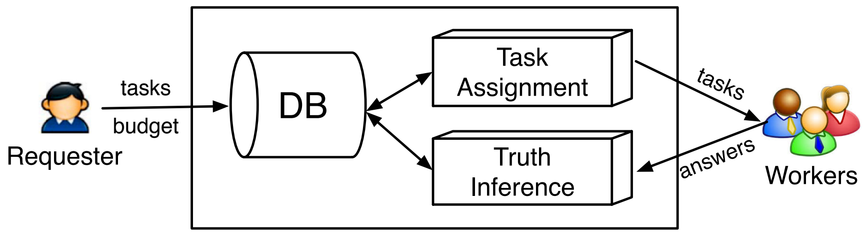


Figure 1.3: Crowdsourcing Framework.

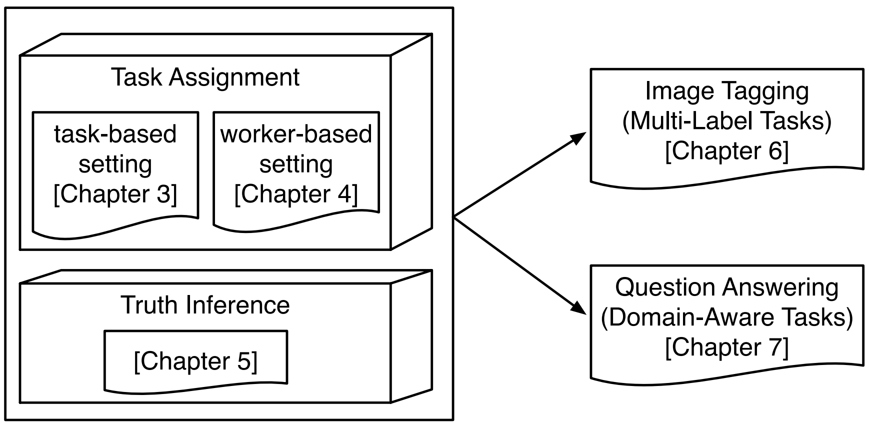

Task Assignment
task-based setting [Chapter 3]
worker-based setting [Chapter 4]
Truth Inference
[Chapter 5]
Image Tagging (Multi-Label Tasks) [Chapter 6]
Question Answering (Domain-Aware Tasks) [Chapter 7]

Figure 1.4: An Overview of the Thesis.

1.3 Problems and Contributions in the Thesis

In this thesis, we have addressed problems related to the two key components in crowdsourcing framework (Figure 1.3), i.e., task assignment and truth inference. Figure 1.4 shows an overview of the thesis. In general, we first discuss the details of task assignment (Chapters 3 and 4) and truth inference (Chapter 5), respectively. Then we study how to combine these two components together and apply them in complex crowdsourcing applications (Chapters 6 and 7). Next, we detail each chapter, respectively.

1.3.1 Task Assignment Problem (Chapters 3 and 4)

Chapters 3 and 4 address problems in the task assignment component, which studies assigning suitable tasks to appropriate workers. As discussed above, there are two settings: task-based setting and worker-based setting.

• Chapter 3 focuses on the task-based setting, and we address an *Online Task Assignment Problem* in [222]:

Problem 1.1 (Online Task Assignment Problem (Chapter 3))**.** *Given a pool of n tasks, when a worker comes to answer tasks, which set of the k tasks should be batched* *in a HIT and assigned to the coming worker?*

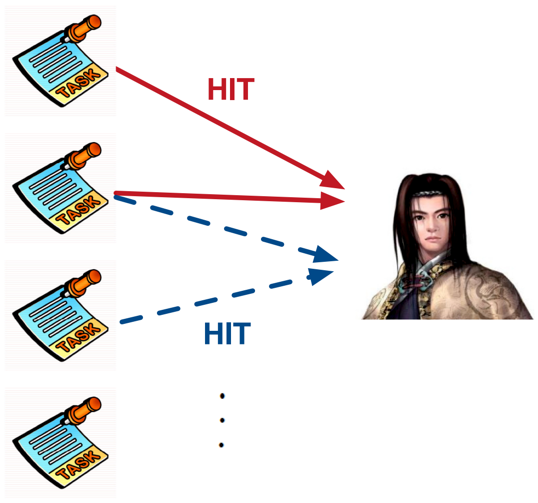


Figure 1.5: Online Task Assignment Problem (Chapter 3).

Figure 1.5 gives an example of the problem. Suppose we have $n = 4$ tasks, and each HIT contains $k = 2$ tasks. When a worker comes to answer tasks, we are interested in which 2-task-combination should be batched in a HIT and assigned to the coming worker.

In AMT, this issue is usually addressed in an *offline* manner: the tasks assigned to all HITs were decided before they are shown to the workers. As pointed out in [27, 127], the main drawback of this approach is that the difficulty level of a task is not considered: for an "easy" task, its final result can be determined even if the current number of answers received from workers is limited, whereas a more difficult or controversial task may require answers from more workers. Notice that a requester may only have a limited amount of budget to pay the workers. It is thus important to decide the task(s) to be included in a HIT, in order to obtain the best answers under the limited budget. Moreover, the problem is inherently complex, since finding the best solution for the task assignment problem can be extremely expensive. Given a pool of n tasks, there are $\binom{n}{k}$ sets of candidate tasks for a HIT. It requires fast assignment response to the worker as the worker may feel bored if waiting for a long time.

Contributions in Solving Online Task Assignment Problem (Problem 1.1)

To solve the problem, we propose a novel online task assignment framework, which takes application-driven evaluation metrics into account. We investigate two popular evaluation metrics, i.e., *Accuracy* and *F-score*, that are widely used by various crowdsourcing applications [85, 92, 127, 161, 192, 195]. Conceptually, our algorithm enumerates all sets of k tasks. For every set of k tasks, our solution estimates the improvement in the quality of the answers, *if* these tasks are really sent to the coming worker. The set of k tasks that maximizes the quality improvement will constitute the new HIT. To realize this, we have to address two key issues:

1. Lack of ground truth. Evaluation metrics, such as *Accuracy* and *F-score*, assume that each task's *ground truth* (or true answer) is known. However, during the task assignment process, it may not be possible to know the ground truth of a task. Thus, existing evaluation metrics are not readily used to solve the task assignment problem. We represent the possible true answer of a task by the *distribution matrix*, which captures the probability distributions of true answers. We study how to populate this matrix with two models, namely Worker Probability (WP) [80, 100, 127, 211] and Confusion Matrix (CM) [19, 92, 200] that are commonly used to describe the performance of workers. We further incorporate the distribution matrix into *Accuracy* and *F-score*, resulting correspondingly in the proposed two functions: Accuracy* and F-score*.
2. Expensive evaluation. Finding the best solution for the task assignment problem can be extremely expensive. Given a pool of n tasks, there are $\binom{n}{k}$ sets of candidate tasks for a HIT. Moreover, due to the incorporation of the distribution matrix, measuring the quality of the distribution matrix

under Accuracy* or F-score* can be expensive. We explore efficient algorithms for quality measurement. We also propose two respective linear-time algorithms to find the best set of tasks for assignment.

• Chapter 4 focuses on the worker-based setting, and we study to address the *Jury Selection Problem* in [219]:

Problem 1.2 (Jury Selection Problem (Chapter 4))**.** *Given a budget and a set of workers, where each worker is associated with a quality score and cost requirement, then which subset of workers should be chosen, such that the overall quality is maximized within the budget limit?*

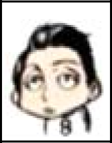
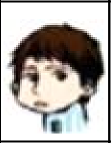
All candidate Workers Set (quality, cost)						
A	B	C	D	E	F	G
						
(0.77, \$9)	(0.7, \$5)	(0.8, \$6)	(0.65, \$7)	(0.6, \$5)	(0.6, \$2)	(0.75, \$3)

Figure 1.6: Jury Selection Problem (Chapter 4).

Figure 1.6 shows a set of seven workers labeled from A to G, where each worker is associated with a *quality* and a *cost*. The *quality* ranges from 0 to 1, indicating the probability that the worker correctly answers a task. This probability can be estimated by using her background information (e.g., her performance in other tasks) [33,127,183]. The *cost* is the amount of monetary reward the worker can get upon finishing a task. In this example, A has a quality of 0.77 and a cost of 9 units. For a jury, the *jury cost* is defined as the sum of workers' costs in the jury and the *jury quality* (or JQ) is defined as the probability that the result returned by aggregating the jury answers is correct. Given a budget of B units, a feasible jury is a jury whose *jury cost* does not exceed B. For example, if $B = \$20$, then $\{B, E, F\}$ is a feasible jury, since its *jury cost*, or $\$5 + \$5 + \$2 = \$12$, is not larger than \$20.

To solve the Jury Selection Problem, a naive solution is to compute the JQ for every feasible jury, and return the one with the highest JQ. However, how to aggregate workers' qualities to compute the JQ, and does there exist a solution that can optimally aggregate workers' qualities? Moreover, the number of feasible juries is exponentially large. So is it possible to efficiently select the jury with the highest aggregated quality?

Contributions in Solving Jury Selection Problem (Problem 1.2)

Since the solution to Jury Selection Problem (JSP) is related to the aggregation of workers' answers (called "*voting strategies*"), we investigate an interesting problem: is it possible to find the optimal voting strategy for JSP among all voting strategies? One simple answer to this problem is to consider all voting strategies. However, as listed in Table 1.1, the number of existing strategies is very large. Moreover, multiple new strategies may emerge in the future. We address this problem by first studying the criteria of a strategy that produce an optimal solution for JSP (i.e., given a jury, the JQ, or *Jury Quality* of the strategy is defined as the highest value among all the possible voting strategies). This is done by observing that voting strategies can be classified into two major categories: *deterministic* and *randomized*. A deterministic strategy aggregates workers' answers without any degree of randomness; MV is a typical example of this class. For a randomized strategy, each answer is returned with some probability. Using this classification, we present the criteria required for a voting strategy that leads to the optimal solution for JSP. We discover that BV satisfies the requirements of an optimal strategy. In other words, BV is the optimal voting strategy with respect to JQ, and will consistently produce better quality juries than the other strategies.

How to solve JSP with BV then? A straightforward solution is to enumerate all feasible juries, and find the one with the highest value of JQ. However, this approach suffers from two major questions:

Table 1.1: Classification of Voting Strategies.

Deterministic Voting Strategies	Randomized Voting Strategies
Majority Voting (MV) [33]	Randomized Majority Voting (RMV) [110]
Half Voting [141]	Random Ballot Voting [9]
Bayesian Voting (BV) [127]	Triadic Consensus [20]
Weighted MV [123]	Randomized Weighted MV [123]
...	...

1. Computing the JQ of a jury for BV requires enumerating an exponentially large number of workers' answers. In fact, we show that this problem is NP-hard;
2. The number of feasible juries is exponentially large.

To solve the first question, we develop a polynomial-time approximation algorithm, which enables a large number of candidate answers to be pruned, without a significant loss of accuracy. We further develop a theoretical error bound of this algorithm. Particularly, our approximate JQ computation algorithm is proved to yield an error of not more than 1%. To tackle the second question, we leverage a successful heuristic, the simulated annealing heuristic, by designing local neighborhood search functions. To evaluate our solutions, we have performed extensive evaluation on real and synthetic crowdsourced data. Our experimental results show that our algorithms effectively and efficiently solve JSP. The quality of our solution is also consistently better than existing works.

1.3.2 Truth Inference Problem (Chapter 5)

• Chapter 5 addresses problem in the truth inference component. To be specific, [221] focuses on the *Truth Inference Problem*:

Problem 1.3 (Truth Inference Problem (Chapter 5))**.** *Given workers' answers collected for all tasks, what is the truth of each task?*

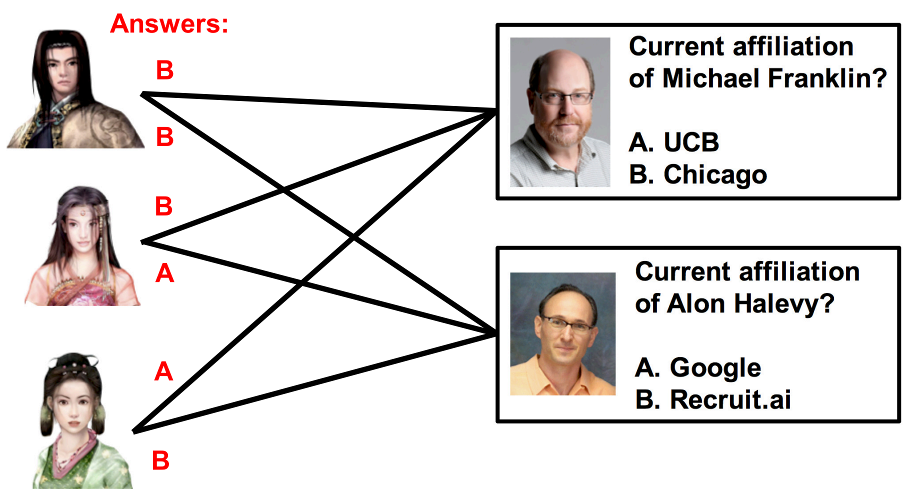

Answers:
B
B
B
A
A
B
Current affiliation of Michael Franklin?
A. UCB
B. Chicago
Current affiliation of Alon Halevy?
A. Google
B. Recruit.ai

Figure 1.7: Truth Inference Problem (Chapter 5).

Figure 1.7 gives an example of the truth inference problem. Suppose we have 2 tasks, which ask about the current affiliations of *Micheal Franklin* and *Alon Halevy*. Three workers come to answer these two tasks, respectively. For example, the first worker thinks that "*B. Chicago*" corresponds to the current affiliation of *Michael Franklin*.

To infer the truth of each task, a straightforward approach is Majority Voting (MV), which takes the answer given by majority workers as the truth. However, the biggest limitation of MV is that it regards all workers as equal. In reality, workers may have different levels of qualities: a high-quality worker carefully answers tasks; a low-quality (or spammer) may randomly answer tasks in order to deceive money; a malicious worker may even intentionally give wrong answers. Thus it is important to capture each worker's quality, which can better infer the truth of each task by trusting more on the answers given by workers with higher qualities. The database community [63, 89, 118, 119, 127, 131, 219] and data mining community [21,47,48,100,104,126,161,182,197,200,226] independently study this problem and propose various algorithms. However, these algorithms are not compared under the same experimental framework and it is hard for practitioners to select appropriate algorithms. So how are these methods similar and dissimilar? Does there exist a best method, i.e., which can beat others consistently on various datasets?

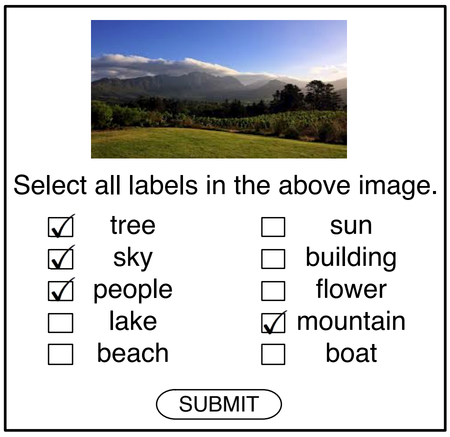
Contributions in Solving Truth Inference Problem (Problem 1.3)

Given that there are lots of existing works that solve the truth inference problem, we provide a comprehensive survey and analyze thoroughly on existing truth inference algorithms. We find that all of them can be summarized into a unified framework, which captures the inherent relationships between each worker's quality and each task's truth. Intuitively, if we know each worker's quality, then we will pay more (less) trust to the answer given by high (low) quality workers; on the other hand, if we know each task's truth, then the worker will be assigned to a high (low) quality if the answer given by the worker is similar (dissimilar) to the truth. We also categorize existing methods in terms of *task types*, *task modeling*, *worker modeling*, and *inference techniques*. We conduct a comprehensive comparison of 17 existing representative methods [21,47,48,100,104,118,119,126,161,182,197,200,226], experimentally compare them on 5 real datasets with varying sizes and task types in real crowdsourcing platforms, make a deep analysis on the experimental results, and provide extensive experimental findings.

1.3.3 Using Task Assignment and Truth Inference in Complex Crowdsourcing Applications (Chapters 6 and 7)

Having discussed the problems related to the two components (Figure 1.3), in Chapters 6 and 7, we study how task assignment and truth inference can be combined in a unified framework, such that specific crowdsourcing applications can be benefited.

• Chapter 6 studies image tagging application, which asks workers to give la-



Select all labels in the above image.

☑ tree	☐ sun
☑ sky	☐ building
☑ people	☐ flower
☐ lake	☑ mountain
☐ beach	☐ boat

Figure 1.8: Multi-Label Task (Chapter 6).

bels to an image. Note that existing crowdsourcing studies [92,127,199,211,222] focus mainly on single-label tasks, which require workers to select a single label (or choice), e.g., select one label from {*positive*, *neutral*, *negative*} in a sentiment analysis task. However, an object can have multiple labels. For example, in image tagging application, an image in Figure 1.8 has *tree*, *sky*, and *mountain* as labels. Moreover, there are other tagging applications, e.g., if we want to label movies, a movie *Matrix* can be labeled with *action* and *sci-fi*; similarly, a person *Barack Obama* can be labeled with *president*, *lawyer*, and *politician*. For these kinds of applications, we use multi-label tasks in crowdsourcing, i.e., workers can select more than one label from a set of given labels (or choices).

Although we can transform a multi-label task to several single-label tasks, this simple approach can generate many tasks, incurring a high cost and latency. For example, the multi-label task in Figure 1.8 is transformed to 10 single-label tasks, where each task inquiries about whether or not the image contains a certain label (e.g., *tree*). As reported in [51], compared with single-label tasks, multi-label tasks enable six times of improvement in terms of human computation time, without sacrificing much quality. Although some recent works [29,56,145,147,149,198] focus on solving multi-label tasks in crowdsourcing, this problem is not well addressed. Workers may have different characteristics in multi-label tasks: a conservative worker would only select labels that the worker is certain of, while a venturous worker may select more labels. Thus it is a challenging problem to design specific task assignment and truth inference algorithms that can characterize workers' behaviors in answering multi-label tasks.

Contributions in Solving Crowdsourced Image Tagging Application

To address it, we design specialized algorithms for the task assignment and truth inference to deal with multi-label tasks. Next we briefly introduce them, respectively.

1. **Online Task Assignment**. When a worker requests tasks, the component targets at instantly assigning k tasks to the worker. A poor assignment may not only waste the budget and time, but also spoil the overall quality. We first measure the uncertainty of each task based on the collected answers, and then estimate how much uncertainty can be reduced *if* the task is really answered by the worker. Finally the k tasks with the highest reduction in uncertainty will be assigned. As the worker's answer to the task is unknown, to compute its reduction in uncertainty, all possible answers given by the worker should be considered, which is exponential (e.g., 2^ℓ answers for ℓ labels). Moreover, to select k tasks (say, out of n tasks), we have to consider all $\binom{n}{k}$ combinations. To reduce the computational complexity, we prove a theorem, which computes the optimal assignment in linear time.
2. **Truth Inference**. When a worker submit answers, the component infers the truth (or correct labels) of each task based on all workers' answers. We use a novel worker model, which can capture workers' diverse characteristics in answering multi-label tasks. As the truth of each task is unknown, each worker's model can only be estimated based on the collected answers. We conduct truth inference in an iterative approach, which can

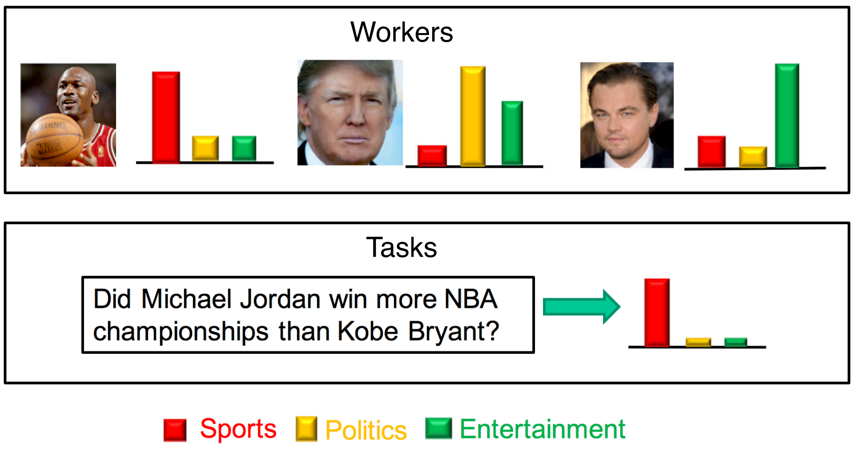

Workers
Tasks
Did Michael Jordan win more NBA championships than Kobe Bryant?
🟥 Sports
🟨 Politics
🟩 Entertainment

Figure 1.9: Domain-Aware Workers and Tasks (Chapter 7).

jointly infer all tasks' truth and workers' models with the following principle: a worker that selects correct labels often will be considered to have a higher quality; meanwhile, a label that is selected by high quality workers for a task is likely to be a correct label for the task. We study how to speed-up the computation by designing an incremental approach. We also leverage the known label correlations to improve the truth inference, by integrating them into the inference method.

• Chapter 7 focuses on another application, i.e., question answering. In that application, often different tasks are associated with different domains, which we call "*domain-aware tasks*". For example, Figure 1.9 illustrates three workers and two tasks. Suppose we have three domains: "*Sports*", "*Politics*", and "*Entertainment*", then worker 1 (*Michael Jordan*) has high quality on *Sports*, and worker 3 (*Leonardo DiCaprio*) has high quality on *Entertainment*. For worker 2 (*Donald Trump*), he may have high qualities on both *Politics* and *Entertainment*. For the task in Figure 1.9, i.e., "*Did Michael Jordan win more NBA championships than Kobe Bryant?*", it is more related to *Sports* compared to other domains. Intuitively, worker 1 should do better than other workers in the task since they have matching domains, thus the task should be more beneficial to be assigned to worker 1.

Note that a common drawback of existing solutions is that they often overlook the worker's ability in different domains. As a matter of fact, workers have a variety of expertise, skills, and cultural backgrounds. However, there are a few challenges in exploiting the domain-aware workers and tasks. For example, how to define the domains? How to exploit the domains in workers and tasks? How to incorporate the domain-ware information in designing the intelligent truth inference and task assignment algorithms?

Contributions in Solving Crowdsourced Question Answering Application

To address it, we exploit domain-aware tasks and characterize the domain-aware task models and worker models, respectively. To be specific, we first compute the domain-aware model for each task (called "*Domain Vector Estimation*") and incorporate the domain-aware task model and worker model to the design of the task assignment and truth inference component. Next, we briefly introduce them, respectively.

1. **Domain Vector Estimation**. This component is responsible for estimating the related domains of each task, based on the domain information in a KB. Specifically, an "entity-linking" algorithm [168] can be used, which extracts entities from the text description of each task. A *domain vector* is then computed for these entities, in order to capture how likely a task belongs to each domain mentioned in a KB.
2. **Online Task Assignment**. When a worker comes and requests new tasks, this component assigns tasks to her. A poor assignment may not only waste budget and time, but also hurt the quality of inference results which depend on workers' answers. To judiciously assign tasks, the component makes decisions based on three factors: (1) the worker's quality, (2) the domain vectors of tasks, and (3) how confident each task's truth can be

inferred from previously received answers. Intuitively, we assign a task to the worker if the task's domains are the worker's expertise and its truth cannot be confidently inferred. The assignment is done *online*, i.e., tasks will be assigned to the worker instantly.

3. **Truth Inference**. When a worker accomplishes tasks and submits answers, the component first stores the worker's answers into database and then infers each task's truth and each worker's model based on two principles: (1) a worker's answer is trusted, if she is a domain expert on her submitted tasks; and (2) a worker is a domain expert if she often correctly answers tasks related to that domain.

1.4 Software, Datasets, Videos

• **Software**. We have open-sourced the implementations of

(1) Task assignment: `http://i.cs.hku.hk/~ydzheng2/crowd_task_assignment/`;

(2) Truth inference: `http://i.cs.hku.hk/~ydzheng2/crowd_truth_inference/`.

• **Datasets**. We also maintained a list of public crowdsourcing datasets:

`http://i.cs.hku.hk/~ydzheng2/crowd_survey/datasets.html`.

• **Videos**. We have given a tutorial related to crowdsourcing data management in SIGMOD'17 [117]. Thanks to SIGMOD Live, our video has been made public on youtube:

(1) Part 1: `https://www.youtube.com/watch?v=-45JkIVYhvo`.

(2) Part 2: `https://www.youtube.com/watch?v=ADAp7XMGtjw`.

1.5 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 gives an overview of related workers in crowdsourced data management. As shown in Figure 1.4, task assignment and truth inference are two parallel important components in crowdsourcing. In task assignment, Chapters 3 and 4 focus on the task-based setting and the worker-based setting, respectively. In truth inference, Chapter 5 focuses on addressing the truth inference problem. Chapters 6 and 7 combine task assignment and truth inference in a unified framework, and apply to them to complex crowdsourcing applications. To be specific, Chapter 6 focuses on image tagging application, where multi-label tasks are leveraged to interact with workers; Chapter 7 focuses on question answering application, where domain-aware tasks are leveraged to interact with workers. Finally, Chapter 8 concludes and lists several future works.

Chapter 2

Related Works

2.1 Overview of Crowdsourced Data Management

Figure 2.1 shows an overview of crowdsourced data management. A requester submits tasks and collects the answers to these tasks by the workers. In the below sections, we first review related works that follow the bottom-up order, i.e., Crowdsourcing Platforms (Section 2.2), Task Design (Section 2.3), Crowdsourcing Fundamental Techniques (Section 2.4), Crowdsourced Operators (Section 2.5), and Crowdsourced Database Systems and Optimization (Section 2.6). Finally, we discuss the focus of the thesis (Section 2.7).

2.2 Crowdsourcing Platforms

There are some existing crowdsourcing platforms, e.g., Amazon Mechanical Turk (AMT) [1], CrowdFlower [5]. Each platform has different goals and provides different functionalities for requesters and workers. Next, we discuss each platform, respectively.

• **Amazon Mechanical Turk (AMT)** [1] is a widely used crowdsourcing platform. AMT focuses on micro-tasks, e.g., labeling an image. A requester can

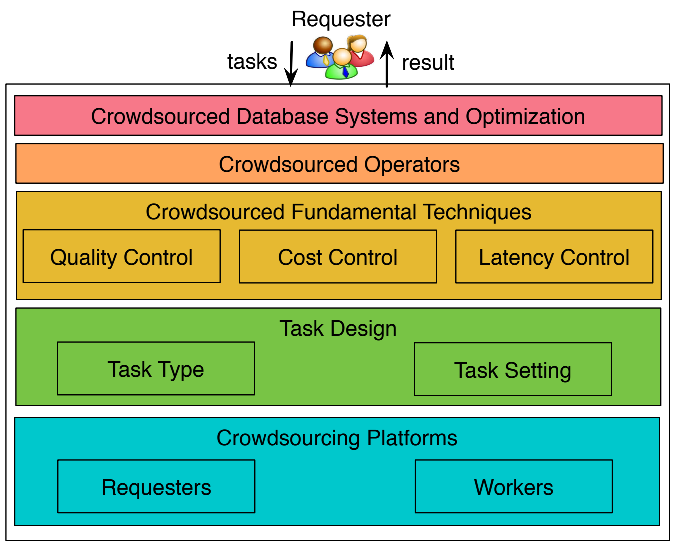


Figure 2.1: Overview of Crowdsourced Data Management.

group multiple micro-tasks as a Human Intelligence Task (HIT). The requester can also set some requirements, e.g., the price of a HIT, the time constraint for answering a HIT, the expiration time for a job to be available on AMT, and the qualification test. (1) A requester can build HITs from several different ways: the requester user interface, AMT Command Line Tools (CLT), and the AMT APIs. Moreover, requesters can also build their own server to manage the tasks and embed their tasks into AMT using innerHTML, which is called the "external-HIT" [7]. When a worker requires a task, AMT transforms the worker information to the requester's server and then the requester can decide what tasks to assign to the server. When a worker submits an answer to AMT, AMT also delivers the answer to the requester. (2) A worker can browse HITs on AMT. Each HIT has some information, e.g., the description of the task, the price, the keywords, the qualification test if required, and the requester's information. After a worker submits the answers of HITs to the platform, s/he can find the total earnings and the status of the submitted HITs on the platform.

• **CrowdFlower** [5] has similar functionalities with AMT, but they still have some differences. First, CrowdFlower has a quality-control component, and it can also embed the tasks with known ground truth in all tasks, and block a worker if the worker answers poorly on the tasks. Second, besides publishing the tasks on its own platform, CrowdFlower also publishes the tasks on other platforms.

• **Other Crowdsourcing Platforms.** There are other crowdsourcing platforms. ChinaCrowd [3] is a multilingual crowdsourcing platform, which supports Chinese and English. Upwork [10] can support macro-tasks, e.g., developing a mobile application. gMission [37] is a spatial crowdsourcing platform that supports spatial tasks.

2.3 Task Design

There are two factors to consider in designing a task: the type of a task (task type) and the setting of a task (task setting).

• **Task Type**. There are several important task types that are widely used in real-world crowdsourcing platforms.

(1) Decision-Making Task [33,126,219]. Workers select Yes/No as the answer. For example, in fact checking, a task asks the truth w.r.t. the statement of a fact.

(2) Single-Choice Task [47,92,222]. Workers select a single answer from multiple options. For example, in sentiment analysis, given a review, it asks workers to select the sentiment of the review (options: Positive, Neutral, Negative).

(3) Multi-Label Task [29,39,109]. Workers select multiple answers from multiple options. For example, given a picture, workers select (multiple) labels from a set of given options that appear in the picture.

(4) Clustering Task [77,136,192]. Workers group a set of objects into several clusters. For example, in entity resolution, given several objects, the task is to create groups of objects that refer to the same entity.

(5) Fill-in-blank Task [69,70]. Workers need to fill-in-blank for an object. For ex-

ample, given a professor, workers are asked to fill the university of the specified professor.

(6) Collection Task [179]. Workers need to collect information, e.g., collecting 100 US universities. Single/multiple choice tasks are closed-world tasks and the workers only need to select from given options, while fill/collection tasks are open-world tasks and the workers can provide any results.

• **Task Setting**. The requester also needs to determine some task settings based on his/her requirements. There are three main factors to be considered.

(1) Pricing / Incentive Design [75, 171]. The requester needs to price each task, usually varying from a few cents to several dollars. Note that pricing is a complex game-theoretic problem. Usually, high prices can attract more workers, thereby reducing the latency; but paying more does not always improve answer quality [67].

(2) Timing [82]. The requester can set time constraints for a task. For each task, the requester can set the time bound (e.g., 10 minutes) to answer it, and the worker must answer it within this time bound.

(3) Quality Control [92, 183]. The requester can select the quality-control techniques provided by the crowdsourcing platform, or design their own specific methods.

2.4 Crowdsourced Fundamental Techniques

As shown in Figure 2.2, there are three fundamental techniques in crowdsourcing, i.e., quality control, cost control and latency control. We review each one respectively, and finally discuss their trade-offs.

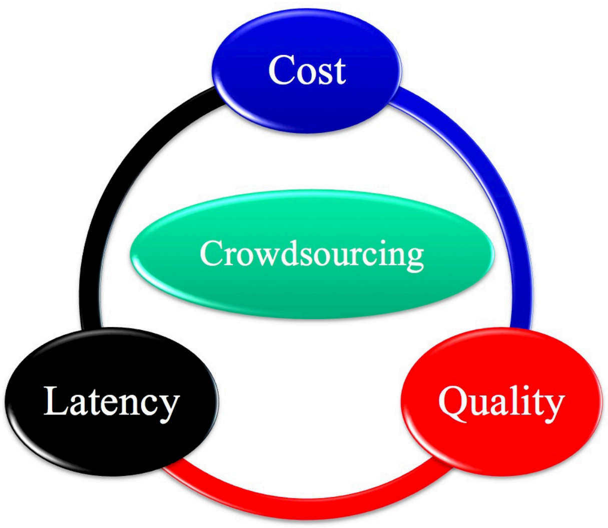


Figure 2.2: Quality v.s. Cost v.s. Latency.

2.4.1 Quality Control

Crowdsourcing may yield relatively low-quality results, e.g., a malicious worker may intentionally give wrong answers, and a worker may have different levels of expertise. To achieve high quality, we need to tolerate errors and infer high-quality results from noisy answers. As discussed in Chapter 1, there are two important components in the crowdsourcing framework: (1) *Task Assignment*, i.e., assigning suitable tasks to appropriate workers; (2) *Truth Inference*, i.e., aggregating workers' answers to infer the truth of each task. Quality control is mainly conducted in these two components.

• **Task Assignment**. There are two settings from the perspectives of tasks and workers: task-based setting and worker-based setting.

(1) In the task-based setting, when a worker comes, existing works [27,63,103,127,140,222] study how to select the most informative tasks and assign them to the coming worker, to achieve high overall quality. They are interesting in selecting k out of n tasks and assigning them to the coming worker (Problem 1.1).

When a worker comes, [27,127] compute an uncertainty score for each task based on its collected answers, select the k most uncertain tasks, and assign them to the worker. There are multiple ways to define the uncertainty. Liu et al. [127] use a quality-sensitive answering model to define each task's uncertainty, and Boim et al. [27] leverage an entropy-like method to compute the uncertainty of each task. Recently, we [222] find that different crowdsourcing applications may have different ways to define quality. In their approach, a crowdsourcing application first specifies a quality metric (e.g., Accuracy, F-score) that it would like to optimize on its data. To meet the requirement, the assignment algorithm will decide which k tasks should be assigned according to the specified metric. Specifically, for each combination of k tasks, it computes how much quality will be improved if they are assigned to a coming worker, and selects the combination that can lead to the maximum improvement in quality. There are some other works [63,177,217,218] that model workers to have diverse skills among different domains, and model tasks to be related to various domains, finally they assign to the coming worker with the tasks that have matching domains with the worker.

There are some works that study the task assignment problem in slightly different settings. Many machine learning algorithms [42,66,100,142,206,224] aim to assign a set of tasks to workers that are most beneficial to their trained models. Budget allocation, which assumes that there is a fixed cost budget, aims to optimally allocate the budget to different tasks. Intuitively, difficult tasks should be allocated with higher budgets. Li et al. [120] focuses on addressing the budget allocation problem. Gao et al. [74] propose a cost-sensitive model to decide whether a task can be better solved by humans or machines. Mo et al. [141] study how to set the *plurality* (i.e., the number of workers to answer each task) under a fixed budget.

(2) In the worker-based setting, given a task and a set of workers (with known qualities), intuitively the workers with high qualities (or having matching skills

[217,218] to the task) should be selected. In addition to these factors, *worker cost* is another key factor for the worker-based task assignment [33,219], which is the monetary cost that each worker requires to answer a task. The cost can be indicated by the worker, or learned from the worker's profiles [33] (e.g., registration date, academic degree). Considering worker budget, Cao et al. [33] propose the Jury Selection Problem: *Given a task, a set of workers (with known qualities and costs) and an overall budget, how to select a subset of workers in order to maximize the task's quality without exceeding the overall budget?* Note that as workers' answers are unknown, to solve the problem, we need to consider all possible cases of workers' answers. For example, given three workers' qualities, we aim at computing the aggregated quality, called Jury Quality (JQ) of the three workers w.r.t. the Majority Voting strategy. As in this case the Majority Voting strategy returns a task's result as the answer that receives at least 2 votes (out of all 3 votes), so in order to compute the JQ, i.e., the probability of correctly returning a result based on the three workers' answers w.r.t. Majority Voting strategy, it can be computed as the probability that at least 2 workers (out of 3) correctly answer the task, by considering all $\binom{3}{3}+\binom{3}{2}=4$ cases.

Cao et al. [33] propose an algorithm to compute JQ w.r.t. the Majority Voting strategy. The algorithm has a time complexity of $\mathcal{O}(|S| \cdot log|S|)$, where S is the given set of workers. Recently, we [219] prove that Bayesian Voting is the optimal strategy under the definition of JQ. That is, given any fixed S, the JQ of S w.r.t. the Bayesian Voting strategy is not lower than the JQ of S w.r.t. any other strategy. So given a set of workers, its collective quality (or JQ) w.r.t. Bayesian Voting strategy is the highest among all voting strategies. [219] further proves that the computation of JQ w.r.t. the Bayesian Voting strategy is NP-hard. To reduce the computational complexity, they propose an $\mathcal{O}(|S|^3)$ approximation algorithm, within 1% error bound. Based on JQ computation, both of the two works [33,219] give the solution to the Jury Selection Problem.

• **Truth Inference**. To infer the truth of each task, it is important to characterize each worker's quality [48, 92, 100, 118, 197, 220]. Some initial works [48, 100] model each worker as a single value $\in [0,1]$ (called *"worker probability"*), capturing the probability that the worker will answer tasks correctly. An extension of worker probability model is to introduce the *confidence interval* [98, 118] into the probability, which to some extent models the variance of a worker's answering behavior. There are also works [19, 126] that model each worker's quality as a *confusion matrix*, which represents the worker's ability in answering different labels, e.g., an optimistic worker tends to answer *"positive"* to a sentiment analysis task even if it holds *"neutral"* sentiment. Some recent works [131, 197, 220] model the *diverse skills* of a worker, which captures the worker's answering abilities for different domains. For example, a *sports* fan while paying no attention to *politics* might answer tasks related to *sports* more correctly compared with those related to *politics*.

In order to infer worker's quality, there are three ways. (1) The first is to adopt qualification test, and when a worker comes, he/she is required to answer qualification test (containing tasks with known ground truth) before the worker can answer real tasks. Then based on the estimated quality of workers, we can eliminate/block the low-quality workers to answer tasks [92, 134, 160]. (2) The second is to use golden tasks, which mix tasks with ground truth into the tasks assigned to workers. Different from qualification test, workers do not know which are golden tasks, and they do not perform a test at their first come. The two approaches both require the ground truth of a subset of tasks to be known in advance. (3) The third computes each worker's quality in an unsupervised manner, i.e., without requiring ground truth to be known [48, 92, 118, 131, 197]. The basic principle is two fold: the workers who have answered tasks correctly tend to have high qualities; the answers of tasks given by high quality workers tend to be the true answers. Following these two intuitive principles, existing works [92, 126, 131, 197] often regard workers' qualities and tasks' truth as two sets of parameters, and follow an iterative approach to update them until convergence. Finally, not only workers' qualities are computed, but also each task's truth is obtained. For the techniques, Majority voting directly computes the truth as the answer given by majority workers, and other methods adopt an iterative approach. Some of them design an optimization function with desired goals [118, 119, 226]; others adopt the probabilistic graphical model [47, 92, 131, 197]. We will summarize different factors of existing truth inference methods in Chapter 5. We list the task models (e.g., modeling the difficulty of tasks, the domains in tasks), the worker models as discussed above, and the techniques used.

2.4.2 Cost Control

The crowd is not free, and a large number of tasks would result in high costs. For example, in entity resolution, if there are 10K objects, there will be about 50M pairs. Even if the price per pair is $0.01, it still takes lots of money. Therefore, a big challenge for crowdsourced data management is cost control. That is, how to reduce human cost while still keeping good result quality.

There are several cost-control techniques. The first is *"pruning"*, which first uses computer algorithms to remove unnecessary tasks and then utilizes the crowd to answer only the useful ones [189,192,195,196,199]. The second is *"task selection"*, which prioritizes tasks with high benefits for crowdsourcing [62, 76, 102,150,154,186,199,209]. The third is *"answer deduction"*, which crowdsources a subset of tasks and based on the answers collected from the crowd, deduces the results of other tasks [15,79,99,189,195,196,212]. The fourth is *"sampling"*, which samples a subset of tasks to crowdsource [84, 134, 193]. There are also *"miscellaneous"* ways [134, 179], which try to leverage well-designed task interfaces and pay-as-you-go approach to reduce costs. These cost-control techniques can also be used together. For example, we can first prune many tasks and then utilize the task-selection idea to select tasks. There are some specialized cost control techniques designed to optimize cost for a particular operator. For example, Marcus et al. [134] proposed a count-based user interface to reduce the number of tasks required for crowdsourced count.

2.4.3 Latency Control

Crowd answers may incur excessive latency for several reasons. For example, workers may be distracted or unavailable, the tasks may not be appealing to enough workers, or the tasks might be difficult for most workers. If the requester has a time constraint, it is important to control latency. There are several strategies for latency control. The first is pricing [67,75]. Usually a higher price attracts more workers and can reduce the latency. The second is latency modeling [165,188]. There are mainly two latency models: the round model [165,188] and the statistical model [67,205]. The round model leverages the idea that tasks can be published in multiple rounds. If there are enough active workers on the crowdsourcing platform, the latency of answering tasks in each round can be regarded as constant time. Thus the overall latency is modeled as the number of rounds. The statistical model is also used to model latency, which leverages the collected statistics from previous crowdsourcing tasks to build statistical models that can capture the workers' arrival time, the completion time, etc. These derived models can then be used to predict and perhaps adjust for expected latency.

2.4.4 Trade-Off

There is a trade-off among cost, quality, and latency. Firstly, the cost-control techniques may sacrifice the quality. For example, the answer deduction may reduce the quality if the crowd makes an error in their answers, and pruning can decrease the quality if some important tasks are pruned as discussed above. Thus, some studies study how to balance quality and cost [192,195]. Secondly, there is also a trade-off between latency and cost. For example, in order to reduce cost, some cost-control techniques (e.g., answer detection) have to publish tasks in multiple rounds. However, increasing the number of rounds will lead to long latency. Thirdly, the similar trade-off also exists between latency and quality. For example, to increase quality, task assignment assigns hard tasks to more workers and easy tasks to fewer workers. To achieve this goal, it needs to select tasks in multiple rounds to better understand the tasks. Thus, a large number of rounds can improve the quality but reduce the latency. To balance the trade-off among quality, cost, and latency, existing studies focus on different problem settings, e.g., optimizing the quality given a fixed cost, minimizing the cost with a little sacrifice of quality, reducing the latency given a fixed cost, etc.

2.5 Crowdsourced Operators

There are many crowdsourced operators proposed to enable real-world applications, e.g., Selection [148, 149, 165, 205], Join [35, 45, 76, 187, 189, 196, 199], Topk/Sort [46, 62, 136, 214], Max/Min [102, 184], Count [134], Collection [152, 179], CrowdFill [152], etc. Various techniques are adopted to optimize the operator's trade-off among three factors: cost, quality and latency. To obtain high-quality results, different applications require to use different crowdsourced operators, which have operator-specific optimization goals over three factors: cost, quality and latency. Next, we discuss these operators, respectively.

• **CrowdSelection [148, 149, 165, 205].** Given a set of items, crowdsourced selection identifies items that satisfy a set of constraints, e.g., selecting images that have both mountains and humans. Existing works can be classified into three categories: (1) Crowd Filtering [149] (or All-Selection) returns all items that satisfy the given constraints; (2) Crowd Find [165] (or k-Selection) returns k items that satisfy the given constraints; (3) Crowd Search [205] (or 1-Selection) returns only one item that satisfies the given constraints. They focus on finding the answers within a cost or latency constraint.

• **CrowdJoin [35, 76, 189, 196, 199].** Join is a very important operator in relational database systems with different types, such as Cross-Join, Theta-Join, and Outer-Join. Existing crowdsourcing works mainly focus on Equi-Join. Given a table (or two tables), a crowdsourced Equi-Join is to find all record pairs in the table (or between two tables) that refer to the same entity. It is rather expensive to enumerate every pair to ask the crowd, and existing crowdsourcing works focus on designing user-friendly interfaces [192] or leveraging transitivity relations [195] to reduce the cost while keeping high quality.

• **CrowdSort/Topk [46, 62, 136, 214].** Given a set of items which are comparable but are hard to be compared by machines, CrowdTopk (or Sort) aims to find top-k items (or a ranking list) based on a certain criterion, e.g., ages of humans in the picture. The challenges include tolerating the comparison error and reducing the cost. Existing works propose heap-based methods [46] and hybrid solutions [136] to address the challenges.

• **CrowdMax (CrowdMin) [80, 102, 184].** Crowdsourced Max [184] is a special case of CrowdTopk where $k = 1$, which finds the max item in a dataset, e.g., finding the most beautiful picture about Great Wall. The tournament algorithm was proposed to reduce the cost.

• **CrowdCount [134].** Crowdsourced Count [134] is to count the number of items in a dataset that satisfy a given constraint, e.g., counting the number of birds in a picture. Existing works focuses on designing effective task types and devising unbiased sampling estimator.

• **CrowdCollection [179].** Different from the above-mentioned query operators which perform queries on a given set of known items, CrowdCollect [179] tries to collect the *unknown* items from the crowd, e.g., enumerating the top-100 universities in US. It focuses on improving the coverage of the collected items and the challenge is to decide whether the collected items are complete.

• **CrowdFill [152].** Similar to CrowdCollect, CrowdFill also tries to collect the *unknown* items from the crowd. However, CrowdFill [152] focuses on asking the crowd to fill the cells in a table. For example, given a table that shows the statistics of football players, it asks workers to fill in the missing cells (e.g., the position of *Lionel Messi*). It focuses on achieving high quality, without requiring too much cost and causing latency.

Note that existing works also study other specific crowdsourced queries, e.g., Categorize [150], Aggregation [84,134], Skyline [78,128,129], Planning [99, 130,174,175,212], Schema Matching [64,90,144,211], Mining [15–18], and Spatial Crowdsourcing [12,101,156,175,212].

2.6 Crowdsourced Database Systems and Optimization

Several crowdsourcing database systems [65, 70, 115, 137, 151] have been proposed to encapsulate the process of interacting with the crowdsourcing platforms. The basic workflow of query processing consists of query parsing, query plan generation, optimization, and execution. Given a query, a parser is first applied and multiple plans can be generated. Then the query optimizer selects the best query plan, and finally executes the plan. Existing crowdsourcing database systems focus on query modeling, query operators, and query optimization techniques. Next we discuss different existing crowdsourced optimization techniques.

• **CrowdDB [70]** extends SQL and defines a new query language, called CrowdSQL, to define which table or attribute should be crowdsourced. In query processing, CrowdDB introduces three crowd operators: CrowdProbe (collect missing information of attributes or new tuples), CrowdJoin (a nested-loop join over two tables), and CrowdCompare (comparison between two elements). CrowdDB proposes rule-based optimization techniques for processing queries with multiple operators.

• **Qurk [137]** uses an SQL-based query language with user-defined functions (UDFs) to enable crowdsourced data management. To facilitate users to implement the UDFs, Qurk has several pre-defined task templates that can generate the UIs for posting different kinds of tasks to the crowd. In query processing, Qurk focuses on implementing join and sort. It implements a block nested loop join and crowdsources the tuples which satisfy join conditions. For sort, it implements comparison-based approaches to execute sort. To further accelerate query processing, it has two important components for cost optimization: task cache and task model. Task cache maintains the crowdsourced answers from previous tasks, while task model trains a model to predict the results for the tasks based on the data that are already collected from the crowd. It uses cost-based optimization to select a good query plan.

• **Deco [151]** separates the user view and system view. The logical relations are specified by a schema designer and queried by an end-user. Raw schema is stored in the RDBMS and it is invisible to the schema designer and users. Deco focuses on crowdsourcing missing values or new tuples based on the defined fetch rules, e.g., Deco specifies resolution rules such as de-duplication and majority voting to resolve inconsistencies in the collected data. Deco also supports other operators, such as Dependent Left Outer Join, Filter and Scan. In query optimization, to find the best query plan with the minimum cost, it considers both cost estimation and optimal query generation. For cost estimation, it proposes an iterative approach to estimate the cost for a query plan. For optimal query plan generation, it enumerates all possible query plans and selects the best query plan with the least estimated cost.

2.7 Focus of the Thesis

As introduced in Chapter 1, the focus of the thesis is mainly on the quality control, which is one of the most fundamental techniques in designing the crowdsourcing applications. Since workers may yield relatively low-quality answers, in order to achieve high quality, it is important to tolerate errors and infer high-quality results from noisy answers. We mainly focus on task assignment

(Chapters 3 and 4) and truth inference (Chapter 5) in the crowdsourcing framework, and study how to combine them together in complex applications, i.e., image tagging (Chapter 6) and question answering (Chapter 7). The design of crowdsourced operators and optimization can definitely benefit from the techniques in task assignment and truth inference, and it will be an interesting and open problem of how to design the applications by integrating the consideration of quality, cost and latency in a unified approach, which we will study as a future work.

Chapter 3

Quality-Aware Online Task Assignment

3.1 Introduction

Crowdsourcing solutions have been proposed to solve problems that are often considered to be hard for computers (e.g., entity resolution [192, 199] and sentiment analysis [127]). Consider the entity resolution problem, where objects in a database referring to the same real-world entity are to be identified. For instance, in a product review database, it is useful to collect all the users' comments about a particular product, which may be named differently by various users (e.g., *iPad Two* and *iPad 2*). To perform this task, crowdsourcing techniques have been developed to generate human understandable tasks (e.g., Are *iPad Two* and *iPad 2* the same or not?) for the database owner (or *requester*) [192]. These tasks, packed into *Human Intelligent Tasks* (HITs), are posted on a crowdsourcing platform (e.g., Amazon Mechanical Turk (AMT)). Internet users (or *workers*) are then invited to answer these tasks, based on which the final result is returned to the requester.

In AMT, every HIT contains a certain number (k) of tasks. Because a worker may give an incorrect answer, a HIT is assigned to a number (z) of workers. The result of each task is then derived based on voting strategies (e.g., Majority Vote). Upon completion of a HIT, the requester may pay a certain amount of money to the worker. A fundamental issue, which we call *task assignment (in the worker-based setting)*, is: Given a pool of n tasks, which of the k tasks should be selected and put to the HIT for the coming worker? In AMT, this issue is usually addressed in an *offline* manner: the tasks assigned to all HITs were decided before they are shown to the workers. As pointed out in [27, 127], the main drawback of this approach is that the difficulty level of a task is not considered: for an "easy" task, its final result can be determined even if the current number of answers received from workers is less than z, whereas a more difficult or controversial task may require answers from more than z workers. Notice that a requester may only have a limited amount of budget to pay the workers. It is thus important to decide the task(s) to be included in a HIT, in order to obtain the best answers under the limited budget. Recent solutions, such as CDAS [127] and AskIt! [27], address this problem through *online assignment strategies* – the HIT is generated *dynamically* when requested by a worker, i.e., the k tasks are chosen for the HIT "on the fly". The statistical confidence of the answers obtained so far for each task is tracked, and tasks whose answers are the least confident are put to the HIT. Thus, the number of times each task is asked can be different. These methods were shown to perform better than the AMT's approach.

However, existing online assignment strategies overlook an important factor – the applications that use the crowdsourced data. Depending on the application semantics, the metric used to gauge the quality of crowdsourced data can be different. In Twitter Sentiment Analysis, for instance, workers are invited to give their sentiments (e.g., "positive", "neutral" or "negative") for each crawled tweet [127]. The *Accuracy* metric, which is the fraction of returned tweets correctly classified, is often used to measure the quality of the sentiment labels [85, 92, 127, 161]. As for entity resolution [192, 199], which asks a worker to judge whether a pair of objects is "equal" or "non-equal", *F-score* [96,132,133,192,195] is often adopted to measure the quality of the entity-resolution results. As our experiments show, considering evaluation metrics in the task assignment process can significantly improve the quality of the crowdsourced results.

Our solutions. In this chapter, we propose a novel online task assignment framework, which takes application-driven evaluation metrics into account. We investigate two popular evaluation metrics, i.e., *Accuracy* and *F-score*, that are widely used by various crowdsourcing applications [85, 92, 127, 161, 192, 195]. Conceptually, our algorithm enumerates all sets of k tasks. For every set of k tasks, our solution estimates the improvement in the quality of the answers, *if* these tasks are really sent to the coming worker. The set of k tasks that maximizes the quality improvement will constitute the new HIT. To realize this, we have to address two key issues:

• **Lack of ground truth.** Evaluation metrics, such as *Accuracy* and *F-score*, assume that each task's *ground truth* (or true answer) is known. However, during the task assignment process, it may not be possible to know the ground truth of a task. Thus, existing evaluation metrics are not readily used to solve the task assignment problem. We represent the possible true answer of a task by the *distribution matrix*, which captures the probability distributions of true answers. We study how to populate this matrix with two models, namely Worker Probability (WP) [80, 100, 127, 211] and Confusion Matrix (CM) [19, 92, 200] that are commonly used to describe the performance of workers. We further incorporate the distribution matrix into *Accuracy* and *F-score*, resulting correspondingly in the proposed two functions: Accuracy* and F-score*.

• **Expensive evaluation.** Finding the best solution for the task assignment problem can be extremely expensive. Given a pool of n tasks, there are $\binom{n}{k}$ sets of candidate tasks for a HIT. Moreover, due to the incorporation of the distribution matrix, measuring the quality of the distribution matrix under Accuracy* or F-score* can be expensive. We explore efficient algorithms for quality mea-

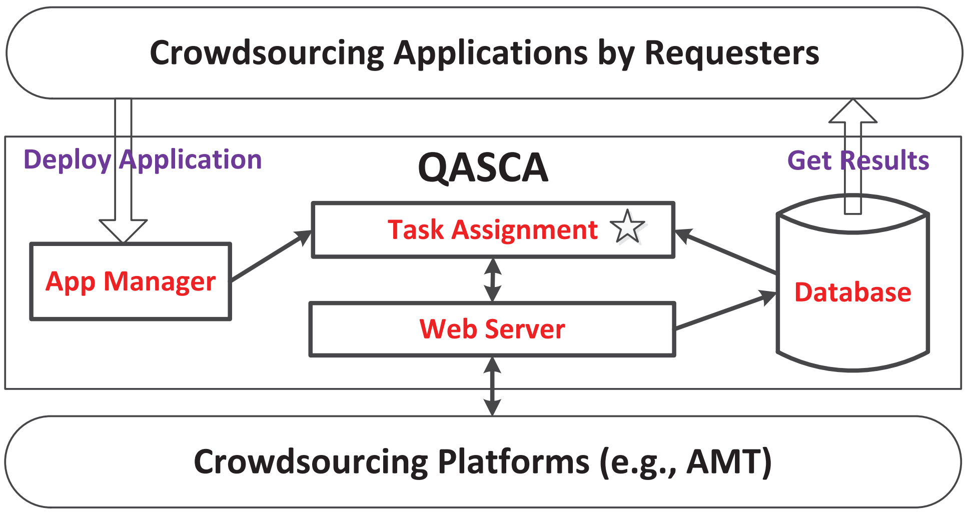


Figure 3.1: The QASCA Architecture.

surement. We also propose two respective linear-time algorithms to find the best set of tasks for assignment.

We have developed a system called QASCA. As shown in Figure 3.1, QASCA is run on top of a crowdsourcing platform (e.g., AMT). The *App Manager* stores the n tasks and other information (e.g., budget) needed by the strategies. The *Task Assignment* runs the strategies and decides the k tasks to be included in the HIT. The *Web Server* then sends the HIT to the workers. The workers' answers are then stored in the *Database* through the *Web Server*, and the derived results are sent back to the requester.

To summarize, we make the following contributions:

(1) We propose a novel task assignment framework by incorporating evaluation metrics into assignment strategies, and formalize the *online task assignment problem* under the proposed framework;

(2) We generalize the definition of evaluation metrics to be able to quantify the result quality w.r.t a distribution matrix, and devise efficient algorithms to identify the optimal result of each task that can maximize the overall quality;

(3) We propose two respective linear online assignment algorithms that can efficiently select the best *k* tasks for a coming worker;

(4) We develop a system called QASCA (`http://i.cs.hku.hk/~ydzheng2/QASCA`), which enables a popular crowdsourcing platform (i.e., AMT) to support our task assignment framework. We evaluate the performance of QASCA on five real applications. Experimental results indicate that QASCA can achieve much better (of more than 8% improvement) result quality compared with five state-of-the-art systems.

The remainder of this chapter is organized as follows. Section 3.2 introduces the architecture. Section 3.3 defines the task assignment problem. We define Accuracy* and F-score*, and explain how to evaluate them in Section 3.4. Efficient online assignment algorithms are devised in Section 3.5. Section 3.6 addresses how to compute distribution matrices. We present our experimental results in Section 3.7. Section 3.8 discusses related works. Finally, we present conclusions and future work in Section 3.9.

3.2 QASCA Architecture

QASCA contains four components, namely APP Manager, Web Server, Task Assignment and Database component in all. To deploy an application, a requester has to configure the three files in APP Manager. We first introduce the four components, and then show how to deploy an entity resolution application in QASCA.

APP Manager: To deploy *n* tasks in QASCA, the requester first needs to create an application folder in APP Manager, which consists of three files: (1) Task File is a JSON-format file which stores the set of *n* tasks and their possible labels; (2) UI Template File is used to render *k* tasks as a user understandable HIT in HTML templates; (3) Configuration File contains all required information about the application, including the number of tasks in each HIT (*k*), the amount of money paid for each HIT (b), the total budget (B), and the evaluation metric.

Web Server: Web Server processes the requests from workers in crowdsourcing platform (AMT). If a worker requests a HIT, Web Server calls Task Assignment, which dynamically generates a HIT and assigns it to the worker; if a worker completes a HIT, Web Server updates the answer set D and parameters (including prior and worker model) in Database.

Task Assignment: Task Assignment is the core component in QASCA. When a worker requests a HIT through Web Server, based on the task model and the worker model stored in Database, Task Assignment identifies k tasks for the worker by considering the evaluation metric specified in APP Manager. Then it dynamically creates a HIT consisting of the identified k tasks, and assigns the HIT to the worker via Web Server. The online task assignment problem is formally defined in Section 3.3.

Database: Database is the component that stores tables containing task and worker model. When a worker requests a HIT through Web Server, tables are queried by Task Assignment; when a worker completes a HIT, tables are updated by Web Server. After all HITs are completed, Database returns the result of each task based on the task model and the evaluation metric.

Suppose a requester wants to deploy an entity resolution application on QASCA and the application has generated $n = 1000$ tasks where each task has the labels L_1 ="equal" and L_2 ="non-equal". The requester first creates an application folder in the APP Manager component. In the created folder, the requester (1) deploys the tasks as JSON-format in the tasks File, (2) specifies the HTML template in the UI Template File, (3) indicates in the Configuration File that each HIT contains $k = 10$ tasks and is paid $b = \$0.02$, and the total invested budget is $B = \$7$, and the evaluation metric is set as *F-score* for "equal" with α=0.5.

When a worker requests a HIT, Web Server acquires worker-id from AMT and passes it to Task Assignment, which identifies $k = 10$ tasks based on the specified evaluation metric (*F-score* for "equal" with α=0.5) in APP Manager, and returns a HIT containing the identified tasks to the worker. When a worker completes a HIT, Web Server updates the answer set and parameters.

The total number of HITs is denoted as $m = B/b = 350$. After obtaining the answers of all $m = 350$ HITs, QASCA terminates and returns the derived result for each task based on considering the task model (stored in Database) and the evaluation metric (*F-score* for "equal" with $\alpha = 0.5$).

3.3 The Task Assignment Problem

We first discuss the task model in Section 3.3.1, and then formally define the task assignment problem in Section 3.3.2. Finally we explain the workflow of QASCA in Section 3.3.3.

3.3.1 Task Model

Let $S = \{q_1, q_2, \ldots, q_n\}$ denote the set of tasks provided by a requester and each task has the same ℓ possible labels (or answers), denoted by $\{L_1, L_2, \ldots, L_\ell\}$. For example, the two labels for all generated tasks in an entity resolution application [195] are $\{L_1$="equal", L_2="non-equal"$\}$. Let $D = \{D_1, D_2, \ldots, D_n\}$ denote the answer set for all tasks. Each D_i contains a set of tuples where each tuple (w, j) denotes that task q_i has been answered by worker w with label L_j. For example, $D_2 = \{(w_1, 1), (w_3, 2)\}$ means that task q_2 is answered twice: worker w_1 has answered task q_2 with label L_1 and worker w_3 has answered task q_2 with label L_2.

When worker w completes a HIT, for each task q_i ($1 \leq i \leq n$), we can compute the probability distribution of task q_i's true label. The probability distributions of all tasks form the task model, called *current distribution matrix*, denoted by Q^c, which is an $n \times \ell$ matrix. The i-th ($1 \leq i \leq n$) row

$Q^c_i = [\ Q^c_{i,1}, Q^c_{i,2}, \ldots, Q^c_{i,\ell}\ ]$ represents the probability distribution for task q_i's true label, and each cell $Q^c_{i,j}$ $(1 \le i \le n, 1 \le j \le \ell)$ denotes the probability that task q_i's true label is L_j. We will discuss how to compute Q^c in Section 3.6.1.

Remarks: For ease of presentation, we assume that (1) the labels are pre-defined, and are the same for all tasks; (2) each task's ground truth is a single label. These assumptions can be relaxed. First, if labels are not pre-defined, [179] addresses how to enumerate possible labels for tasks. Second, if each task has multiple true labels, we can follow [149] to decompose each task into ℓ filter tasks, where each filter task is to decide whether the original task satisfies a corresponding label or not. To handle a domain with continuous values, we can adopt the *bucketing* method [121], which discretizes the domain into different buckets, where each bucket indicates a specific label.

3.3.2 Task Assignment

Note that a worker may request multiple HITs, so we keep track of the history of previously assigned tasks. Let S^w denote the candidate set of tasks for worker w, i.e., the set of tasks that have not been assigned to worker w (each worker has her unique S^w). QASCA will not assign duplicated tasks to the same worker. When worker w requests a HIT, it selects k tasks in S^w and assigns them to her. To select tasks for worker w, QASCA first estimates the probability distribution of each task q_i's $(q_i \in S^w)$ true label *if* worker w answers it. The estimated probability distributions of all tasks in S^w form an $n \times \ell$ matrix Q^w, called *estimated distribution matrix for worker w*. The i-th $(q_i \in S^w)$ row $Q^w_i = [\ Q^w_{i,1}, Q^w_{i,2}, \ldots, Q^w_{i,\ell}\ ]$ represents the estimated probability distribution for task q_i's true label if it is answered by worker w. Each cell $Q^w_{i,j}$ $(q_i \in S^w, 1 \le j \le \ell)$ denotes the estimated probability that task q_i's true label is L_j if it is answered by worker w. Note that row i (or Q^w_i) is empty when $q_i \notin S^w$. We will discuss how to compute Q^w in Section 3.6.3.

Let the vector $X = [x_1, x_2, \ldots, x_n]$ denote an assignment of HIT, where each element $x_i = 1$ (0) indicates that the task q_i will (not) be chosen to assign for the coming worker. When worker w comes, based on S^w, we define X as a *feasible assignment* if it satisfies (1) $\sum_{i=1}^{n} x_i = k$, and (2) if $x_i = 1$, then $q_i \in S^w$. There are k tasks in a HIT and we can only assign tasks in S^w to worker w. Thus the number of feasible X is $\binom{|S^w|}{k} \leq \binom{n}{k}$.

Given Q^c, Q^w, and a feasible X, we construct a matrix Q^X called *assignment distribution matrix for* X. The i-th row, Q_i^X, is the (estimated) probability distribution of task q_i if worker w answers all the assigned tasks in X. We can construct Q^X using Q^c and Q^w: for an unassigned task q_i in X (or $x_i = 0$), its distribution (Q_i^X) remains to be Q_i^c; for an assigned task q_i in X (or $x_i = 1$), its distribution (Q_i^X) is estimated to be Q_i^w, thus

$$Q_i^X = \begin{cases} Q_i^c & \text{if } x_i = 0, \\ Q_i^w & \text{if } x_i = 1. \end{cases} \tag{3.1}$$

Let $F(\cdot)$ be an evaluation metric, which is used to evaluate the quality of a distribution matrix. When worker w requests a HIT, there are $\binom{|S^w|}{k}$ feasible X, and the problem is to choose the optimal feasible X^* that maximizes $F(Q^X)$. (Note that in the extreme case that $|S^w| < k$, we just have to assign the candidates tasks S^w to worker w.) We formally define the **(online) task assignment problem** in Definition 3.1.

Definition 3.1. *When a worker w requests a HIT, given the current distribution matrix (Q^c), the estimated distribution matrix for the worker w (Q^w), and the function $F(\cdot)$, the problem of task assignment for the worker w is to find the optimal feasible assignment vector X^* such that $X^* = argmax_X\ F(Q^X)$.*

3.3.3 The Workflow of QASCA

To deploy an application, a requester needs to set n tasks with ℓ labels and she should also indicate the number of tasks in each HIT (k), the amount of money paid for each HIT (b), the total invested budget (B) and the evaluation

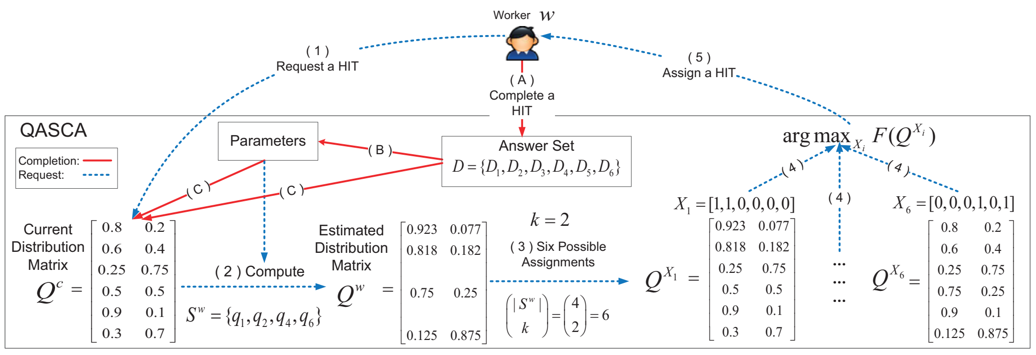


Figure 3.2: The Workflow of QASCA.

metric. In [87, 93], the issues of setting appropriate values of k and b are discussed. The evaluation metric (e.g., *Accuracy* and *F-score*), which depends on the application semantics, will be addressed in Section 3.4.

There are two events from workers that QASCA should process: the event when a worker completes a HIT (called "HIT completion") and the event when a worker requests a HIT (called "HIT request"). Based on the workflow in Figure 3.2, we give an example (Example 1) to show how QASCA processes these two events.

Example 1. *The solid (red) lines and the dotted (blue) lines in Figure 3.2 represent how QASCA processes a HIT completion event and a HIT request event, respectively:*
(1) when a worker w completes a HIT (HIT completion process), QASCA does several updates in the database: it first updates the answer set D [2] *(step A), and then based on the new D, it updates some parameters such as workers' qualities (step B). Finally it uses these new parameters to update Q^c (step C);*
(2) when a worker w requests a HIT (HIT request process), suppose $S = \{q_1, q_2, q_3, q_4, q_5, q_6\}$, and each HIT contains $k = 2$ tasks. QASCA first extracts Q^c (step 1) and parameters from database to compute Q^w for worker w (step 2). Assume worker w has answered q_3 and q_5 previously, now the candidate set of tasks for her is

[2]Note that the answer set D is continuously updated. That is, in the HIT request process, the current D is used to decide which tasks should be assigned; while in the HIT completion process, QASCA updates D based on worker's answers.

$S^w = \{q_1, q_2, q_4, q_6\}$. Since $|S^w| = 4$ and $k = 2$, there are $\binom{4}{2} = 6$ feasible assignments for worker w (step 3). Consider the first feasible assignment $X_1 = [1,1,0,0,0,0]$, which assigns q_1 and q_2 to worker w. We construct Q^{X_1} by Equation 3.1, so $Q_i^{X_1} = Q_i^w$ for $i = 1, 2$ and $Q_i^{X_1} = Q_i^c$ for $i = 3, 4, 5, 6$. Similarly, we can construct Q^{X_2}, Q^{X_3}, Q^{X_4}, Q^{X_5}, Q^{X_6} for other feasible assignments. Based on the chosen function $F(\cdot)$ (Accuracy or F-score), assume that $F(Q^{X_i})$ $(1 \le i \le 6)$ is maximized on Q^{X_6} (step 4) and since $X_6 = [0,0,0,1,0,1]$, QASCA batches tasks $\{q_4, q_6\}$ in a HIT and assigns it to worker w (step 5).

To help readers better understand our chapter, we summarize notations in Table 3.1. There are three challenges in Figure 3.2. Firstly, how can we define $F(\cdot)$ for different evaluation metrics; secondly, given Q^c, Q^w and $F(\cdot)$, how can we efficiently compute the optimal assignment in Definition 3.1 (step 4); thirdly, how can we compute Q^c when a HIT is completed (step C) and estimate Q^w for worker w when a HIT is requested (step 2). We respectively address these challenges in the following three sections.

3.4 Evaluating Quality Metrics

In this section, we study the two popular evaluation metrics (*Accuracy* and *F-score*) in Sections 3.4.1 and 3.4.2, respectively. For each evaluation metric, we first introduce its original definition assuming the known ground truth, then we study its variants by incorporating a distribution matrix Q. Finally we define $F(\cdot)$ by discussing how to evaluate the quality of Q w.r.t. an evaluation metric.

We first clarify some notations. We denote the result vector by $R = [r_1, r_2, \ldots, r_n]$, where $1 \le r_i \le \ell$ and L_{r_i} is the returned label (or result) for q_i. We also denote the ground truth vector by $T = [t_1, t_2, \ldots, t_n]$, where $1 \le t_i \le \ell$ and L_{t_i} is the ground truth label for q_i. The formal definition of function *Accuracy* or *F-score* is $F(T, R)$, which evaluates the quality of R based on known T. In task assignment scenarios, the ground truth vector T is unknown, but we can obtain

Table 3.1: Notations Used in Chapter 3.

Notation	Description
Task Model	
q_i	The i-th task ($1 \leq i \leq n$)
L_j	The j-th label ($1 \leq j \leq \ell$)
t_i	The index of true label for q_i ($1 \leq i \leq n$), $1 \leq t_i \leq \ell$
D_i	Answer set for task q_i ($1 \leq i \leq n$)
S	tasks set: $S = \{q_1, q_2, \ldots, q_n\}$
D	Answer set for all tasks: $D = \{D_1, D_2, \ldots, D_n\}$
Q^c	Current distribution matrix (size $n \times \ell$ matrix)
Task Assignment	
k	The number of tasks per HIT
S^w	Candidate tasks set for worker w
Q^w	Estimated distribution matrix for worker w (size $n \times \ell$ matrix)
X	Assignment vector ($1 \times n$), where each element $x_i = \{0, 1\}$
Q^X	Assignment distribution matrix for X (size $n \times \ell$ matrix)
R	Result vector ($1 \times n$), where each element $1 \leq r_i \leq \ell$
a_i^w	The index of answered label by worker w for q_i
Parameters	
m^w	Worker Probability (WP) for worker w
M^w	Confusion Matrix (CM) for worker w (size $\ell \times \ell$ matrix)
p_j	Prior probability for label L_j ($1 \leq j \leq \ell$)

a distribution matrix Q based on the crowd's answers. Thus, we generalize the evaluation metrics to be able to quantify the quality of result vector R (called "result quality") w.r.t the distribution matrix Q, i.e., $F^*(Q, R)$. Since given a distribution matrix Q, the requesters want the best results $R^* = \text{argmax}_R \ F^*(Q, R)$ to be returned. So in order to evaluate the quality of Q, we consider the choice of R^* and use the best quality that Q can reach to evaluate the quality of Q, i.e., $F(Q) = \max_R \ F^*(Q, R) = F^*(Q, R^*)$.

3.4.1 Accuracy

Accuracy is an evaluation metric used by many crowdsourcing applications [44, 91, 92, 127, 161]. It aims to measure the overall classification quality among all labels. For example, in a sentiment analysis application, if a requester focuses on the overall quality among all three labels (i.e., "positive", "neutral" and "negative"), *Accuracy* can be used as the evaluation metric. It is defined as the fraction of returned labels that are correct. Let $\mathbb{1}_{\{\cdot\}}$ denote an indicator function which returns 1 if its argument is true; 0, otherwise. For example, $\mathbb{1}_{\{5=2\}} = 0$ and $\mathbb{1}_{\{5=5\}} = 1$. Then we derive

$$Accuracy(T,R) = \frac{\sum_{i=1}^{n} \mathbb{1}_{\{t_i = r_i\}}}{n}. \tag{3.2}$$

Consider an example where $n = 4$, $\ell = 3$, $T = [2,1,3,2]$ and $R = [2,1,3,1]$. Since it correctly identifies the labels of the 1st, 2nd, and 3rd tasks, we have $Accuracy(T,R) = \frac{3}{4} = 0.75$.

Accuracy*

As shown above, the definition of $Accuracy(T,R)$ requires to know the ground truth T. In practice, however, we only have the distribution matrix Q, which records the probability distribution of each task's true label. Therefore, we use the expected accuracy to measure the result quality. Since Q_i (the i-th row of Q) represents the probability distribution of task q_i's true label, then we have $P(t_i = j) = Q_{i,j}$ and $\mathbb{E}[\mathbb{1}_{\{t_i=j\}}] = P(t_i = j)\cdot 1 + P(t_i \neq j)\cdot 0 = Q_{i,j}$. Thus $\text{Accuracy}^*(Q,R)$ is defined as

$$\text{Accuracy}^*(Q,R) = \mathbb{E}[\, Accuracy(T,R)\,] = \frac{\sum_{i=1}^{n} Q_{i,r_i}}{n}. \tag{3.3}$$

Note that the assumptions made here are that the distribution matrix Q can be known in advance. In general, many existing truth inference works (see [221], or Chapter 5) have focused on computing the matrix Q. Basically it leverages the inherent relationships between each worker's quality and each task's truth, and iteratively derives the two sets of parameters. Then for each task, it considers the answers given by workers and finally derives such Q.

Specifically, $\text{Accuracy}^*(Q, R)$ represents the *expected* number of correctly answered tasks out of all tasks. For example, consider the distribution matrix Q^c in Figure 3.2. Given a result vector of $R = [1, 2, 2, 1, 1, 1]$, its accuracy (w.r.t Q^c) is defined as $\text{Accuracy}^*(Q^c, R) = \frac{0.8+0.4+0.75+0.5+0.9+0.3}{6} = 60.83\%$.

Identify the Optimal Result for Accuracy*

In order to measure the quality of a distribution matrix Q, we need to determine the optimal result vector R^* that maximizes $\text{Accuracy}^*(Q, R)$, i.e., $R^* = \text{argmax}_R \ \text{Accuracy}^*(Q, R)$. To compute R^*, an intuitive idea is to return the most likely label for each task, i.e., $R^* = [r_1^*, r_2^*, \ldots, r_n^*]$ where $r_i^* = \text{argmax}_j \ Q_{i,j}$. We next prove that the idea is correct in Theorem 3.1.

Theorem 3.1. *For Accuracy*, the optimal result r_i^* ($1 \leq i \leq n$) of a task q_i is the label with the highest probability, i.e., $r_i^* = \text{argmax}_j \ Q_{i,j}$.*

Proof. We prove the theorem by proof of contradiction. Suppose the theorem does not hold. Then in the optimal result vector $R^* = [r_1^*, r_2^*, \ldots, r_n^*]$, there exists an index t ($1 \leq t \leq n$), such that $r_t^* \neq \text{argmax}_j \ Q_{t,j}$. So we can construct a result vector $R' = [r'_1, r'_2, \ldots, r'_n]$ where $r'_t = \text{argmax}_j \ Q_{t,j}$ and $r'_i = r_i^*$ for $i \neq t$. Then we have $\text{Accuracy}^*(Q, R') - \text{Accuracy}^*(Q, R^*) = (Q_{t,r'_t} - Q_{t,r_t^*})/n > 0$, which contradicts that R^* is the optimal result vector. Thus the theorem is correct. □

Based on Theorem 3.1, we know that for Accuracy*, the optimal result of a task q_i only depends on its own distribution Q_i. Take Q^c in Figure 3.2 as an example. Consider the first task. Since $Q^c_{1,1}$ (0.8) $>$ $Q^c_{1,2}$ (0.2), the optimal result for q_1 is $r_1^* = 1$. Similarly we can derive the optimal result vector R^* for all tasks, i.e., $R^* = [1, 1, 2, 1, 1, 2]$ (or $[1, 1, 2, 2, 1, 2]$ as $Q^c_{4,1} = Q^c_{4,2}$). Thus the quality of Q^c for the metric *Accuracy*, is measured as $F(Q^c) = \text{Accuracy}^*(Q^c, R^*) = 70.83\%$

For *Accuracy*, the definition (Equation 5.3) and the optimal result selection (Theorem 3.1) are not very difficult. But for *F-score*, these problems become more challenging.

3.4.2 F-score

In applications such as text classification [42], sentiment analysis [132, 133], entity resolution [192, 195, 199] and fact finding [155, 215], a requester may only be interested in a particular label (which we call *target label*). In this situation, a task can be simplified as a two-label task (i.e., $\ell = 2$), where the two labels are the "target label" and "non-target label", denoted by L_1 and L_2, respectively. For example, in an sentiment analysis application, if the requester wants to pick out the "positive" sentiment tweets with a high confidence, then each task is to identify whether a sentence's sentiment is L_1="positive" or L_2="non-positive".

The *F-score*, introduced in [181], can be used to capture the quality of answers to the two-label tasks above. This measure (*F-score*) is formally defined as the weighted harmonic mean of two commonly known metrics, namely *Precision* and *Recall*:

$$\textit{F-score} = \frac{1}{\alpha \cdot \frac{1}{\textit{Precision}} + (1-\alpha) \cdot \frac{1}{\textit{Recall}}}, \tag{3.4}$$

where

(1) ***Precision*** is the faction of tasks with returned results L_1 that are actually correct, i.e.,

$$\textit{Precision}(T, R) = \frac{\sum_{i=1}^{n} \mathbb{1}_{\{t_i=1\}} \cdot \mathbb{1}_{\{r_i=1\}}}{\sum_{i=1}^{n} \mathbb{1}_{\{r_i=1\}}}; \tag{3.5}$$

(2) ***Recall*** is the fraction of tasks with ground truth L_1 that are actually returned with results L_1, i.e.,

$$\textit{Recall}(T, R) = \frac{\sum_{i=1}^{n} \mathbb{1}_{\{t_i=1\}} \cdot \mathbb{1}_{\{r_i=1\}}}{\sum_{i=1}^{n} \mathbb{1}_{\{t_i=1\}}}; \tag{3.6}$$

(3) $\alpha \in (0, 1)$ is a parameter that controls the degree of emphasis: $\alpha \in (\frac{1}{2}, 1)$ emphasizes *Precision*; $\alpha \in (0, \frac{1}{2})$ emphasizes *Recall*; and we call the *balanced F-score* if $\alpha = \frac{1}{2}$.

Note that besides *F-score*, there are some other combinations of *Precision* and *Recall*, e.g., the average $(\textit{Precision} + \textit{Recall})/2$ and the geometric mean $\sqrt{\textit{Precision} \cdot \textit{Recall}}$. Due to the widespread use of *F-score*, we only consider optimizing *F-score* in this chapter, and an interesting future work might be how the techniques can be applied to other combinations.

By plugging Equations 3.5 and 3.6 into Equation 3.4, we obtain:

$$\textit{F-score}(T, R, \alpha) = \frac{\sum_{i=1}^{n} \mathbb{1}_{\{t_i=1\}} \cdot \mathbb{1}_{\{r_i=1\}}}{\sum_{i=1}^{n} [\, \alpha \cdot \mathbb{1}_{\{r_i=1\}} + (1-\alpha) \cdot \mathbb{1}_{\{t_i=1\}}]}. \tag{3.7}$$

For the parameter α, if a requester wants to select "positive" sentiment sentences with a high confidence, she can set α to a large value (e.g., 0.75) which emphasizes Precision more. On the other hand, if a company manager wants to collect as many "positive" comments for their products as possible, she can give more attention to *Recall* by setting α to a lower value (e.g., 0.25).

Challenges

As the numerator and denominator of $\textit{F-score}(T, R, \alpha)$ both have the random variable $\mathbb{1}_{\{t_i=1\}}$, its expectation cannot be computed as easily as for *Accuracy* (Equation 5.3), so we resort to another way of computing its expectation. We denote τ as the set of all possible ground truth vectors, i.e., $\tau = \{1, 2\}^n$ and $|\tau| = 2^n$. Given Q, the probability that $T' = [t'_1, t'_2, \ldots, t'_n] \in \tau$ is the ground truth vector is $\prod_{i=1}^{n} Q_{i,t'_i}$. Then, we have

$$\mathbb{E}[\, \textit{F-score}(T, R, \alpha) \,] = \sum_{T' \in \tau} \textit{F-score}(T', R, \alpha) \cdot \prod_{i=1}^{n} Q_{i,t'_i}. \tag{3.8}$$

There are two challenges related to this equation:

One is how to efficiently compute or estimate Equation 3.8. It is computationally expensive to enumerate all 2^n possible T'. But given α and R, for different $T' \in \tau$, the numerator and denominator of $\textit{F-score}(T', R, \alpha)$ can only take at most $n+1$ possible values respectively, thus *F-score*(T', R, α) can only have at most $(n+1)^2$ possible values. Based on this observation, for a given Q and α, Equation 3.8 can be accurately calculated in $\mathcal{O}(n^3)$ time [95]. Nevertheless, this complexity is still very high to a task assignment system as it is a key step in task assignment. Thus it is better to find an accurate approximation that is efficient to compute.

Another challenge is that if we directly apply Equation 3.8 to derive the optimal result for each task, we observe two interesting facts that are different from *Accuracy*:

Observation 1: Returning the label with the highest probability in each task may not be optimal (even for $\alpha = 0.5$);

Observation 2: Deriving the optimal result of a task q_i does not only depend on the task's distribution (or Q_i) itself.

In order to verify these two observations, we next give two counter-examples in Example 2.

Example 2. *Consider* $\alpha = 0.5$ *and* $Q = \begin{bmatrix} 0.35 & 0.65 \\ 0.55 & 0.45 \end{bmatrix}$. *If we return the label with the highest probability for each task, then we obtain a result vector* $\widetilde{R} = [2, 1]$ *and the corresponding* $\mathbb{E}[$ *F-score*$(T, \widetilde{R}, \alpha)$ $] = \frac{1}{0.5 \cdot 1 + 0.5 \cdot 2} \cdot 0.35 \cdot 0.55 + 0 \cdot 0.35 \cdot 0.45 + \frac{1}{0.5 \cdot 1 + 0.5 \cdot 1} \cdot 0.65 \cdot 0.55 + 0 \cdot 0.65 \cdot 0.45 = 48.58\%$ *(Equation 3.8). However, by enumerating all possible* $R \in \{1, 2\}^2$, *we can derive the optimal* $R^* = [1, 1]$ *and* $\mathbb{E}[$ *F-score*(T, R^*, α) $] = 53.58\%$. *This example verifies our first observation.*

Consider $\widehat{\alpha} = 0.5$ *and* $\widehat{Q} = \begin{bmatrix} 0.35 & 0.65 \\ 0.9 & 0.1 \end{bmatrix}$. *Using the same method as above, we can obtain the optimal* $\widehat{R}^* = [2, 1]$. *Compared with the above example, we can see that for the same* $\alpha = \widehat{\alpha} = 0.5$ *in F-score, even if* Q_1 *and* $\widehat{Q}_1$ *are the same (i.e.,* $[0.35, 0.65]$*), the computed* $r_1^* = 1$ *and* $\hat{r}_1^* = 2$ *are different. This example shows that the optimal result for each task is not only dependent on the task's distribution itself.*

To address the above two challenges, we first give an approximation for

$\mathbb{E}[\,\textit{F-score}(T, R, \alpha)\,]$ in Section 3.4.2 and then discuss how to compute R^* for the approximated function in Section 3.4.2.

F-score*

Following Equation 3.7, we approximate $\mathbb{E}[\,\textit{F-score}(T, R, \alpha)\,]$ as $\frac{\mathbb{E}[\,\sum_{i=1}^{n} \mathbb{1}_{\{t_i=1\}} \cdot \mathbb{1}_{\{r_i=1\}}\,]}{\mathbb{E}[\,\sum_{i=1}^{n} [\,\alpha \cdot \mathbb{1}_{\{r_i=1\}} + (1-\alpha) \cdot \mathbb{1}_{\{t_i=1\}}]\,]}$ to boost efficiency, which is the ratio of the expectation of numerator and denominator of $\textit{F-score}(T, R, \alpha)$. By plugging Q inside, we can formally define F-score$^*(Q, R, \alpha)$ as:

$$\text{F-score}^*(Q, R, \alpha) = \frac{\sum_{i=1}^{n} Q_{i,1} \cdot \mathbb{1}_{\{r_i=1\}}}{\sum_{i=1}^{n} [\,\alpha \cdot \mathbb{1}_{\{r_i=1\}} + (1-\alpha) \cdot Q_{i,1}\,]}. \tag{3.9}$$

For example, consider the values of $\widehat{\alpha}$, $\widehat{Q}$ and $\widehat{R}^*$ in Example 2. Based on Equations 3.8 and 5.4, we can obtain $\mathbb{E}[\,\textit{F-score}(T, \widehat{R}^*, \widehat{\alpha})\,] = 79.5\%$ and F-score$^*(\widehat{Q}, \widehat{R}^*, \widehat{\alpha}) = \frac{0.9}{0.5 \cdot 1 + 0.5 \cdot (0.35 + 0.9)} = 80\%$, respectively. The error in the example is 0.5%.

Approximation Error. Let A and B denote two random variables. The same approximation ($\mathbb{E}\left[\frac{A}{B}\right] \approx \frac{\mathbb{E}[A]}{\mathbb{E}[B]}$) has also been used by other works [34, 114, 146, 163] for efficiency's sake. Furthermore, [163] gives a general formula by expanding Taylor series: $\mathbb{E}\left[\frac{A}{B}\right] = \frac{\mathbb{E}[A]}{\mathbb{E}[B]} + \sum_{v=1}^{\infty} \phi_v$, where $\phi_v = (-1)^v \cdot \frac{\mathbb{E}[A] \cdot <^v B> + <A,^v B>}{(\mathbb{E}[B])^{v+1}}$, $<^v B> = \mathbb{E}[\,(B - \mathbb{E}[B])^v\,]$, and $< A,^v B > = \mathbb{E}[\,(A - \mathbb{E}[A]) \cdot (B - \mathbb{E}[B])^v\,]$. The standard approximation formula (derived by delta method) used in [34, 146] is $\mathbb{E}\left[\frac{A}{B}\right] = \frac{\mathbb{E}[A]}{\mathbb{E}[B]} + \mathcal{O}(n^{-1})$ for $n \to \infty$. By setting A and B as the numerator and denominator of $\textit{F-score}(T, R, \alpha)$, we can generally derive $\mathbb{E}[\,\textit{F-score}(T, R, \alpha)\,] = \text{F-score}^*(Q, R, \alpha) + \mathcal{O}(n^{-1})$. We also verify this in experiments (Section 3.7.1) and show that the error is small ($\leq 0.01\%$) when $n \geq 10^3$, which means that our approximation is reasonable.

Identify the Optimal Result for F-score*

Unlike Accuracy*, to identify the optimal result for F-score*, the method that chooses the most possible label of each task is not correct anymore. However, the intuition that it is preferable to return a label with a large probability value may still hold. Based on this idea, we conjecture that there exists a threshold w.r.t each task that can determine whether a target label should be returned or not. That is, a target label should be returned only when its probability is not lower than the threshold; otherwise the non-target label is returned. More formally, let θ_i be the threshold w.r.t a task q_i. If $Q_{i,1} \geq \theta_i$, we return the target label (i.e., $r_i^* = 1$); otherwise, the non-target label is returned (i.e., $r_i^* = 2$). We prove the correctness of the conjecture in Theorem 3.2. An interesting observation from the theorem is that every task has the same threshold, which is equal to $\lambda^* \cdot \alpha$, where λ^* represents the optimal value for F-score* (or the quality evaluation of Q), i.e., $\lambda^* = \max_R$ F-score$^*(Q, R, \alpha)$.

Theorem 3.2. *Given Q and α, for F-score*, the optimal result r_i^* ($1 \leq i \leq n$) of a task q_i can be derived by comparing $Q_{i,1}$ with the threshold $\theta = \lambda^* \cdot \alpha$, i.e., $r_i^* = 1$ if $Q_{i,1} \geq \theta$ and $r_i^* = 2$ if $Q_{i,1} < \theta$.*

Proof. In the proof for Theorem 2, we assume that λ^* is known, and we try to exploit how the optimal result vector R^* can be constructed with the known λ^*. As $\lambda^* = \max_R$ F-score$^*(Q, R, \alpha)$, which means that for any $R \in \{1, 2\}^n$, the inequality $\lambda^* \geq$ F-score$^*(Q, R, \alpha)$ holds, i.e., we have

$$\lambda^* \geq \frac{\sum_{i=1}^n Q_{i,1} \cdot \mathbb{1}_{\{r_i=1\}}}{\alpha \cdot \sum_{i=1}^n \mathbb{1}_{\{r_i=1\}} + (1-\alpha) \cdot \sum_{i=1}^n Q_{i,1}},$$

then we can further derive

$$\sum_{i=1}^n (Q_{i,1} - \lambda^* \cdot \alpha) \cdot \mathbb{1}_{\{r_i=1\}} \leq \lambda^* \cdot (1-\alpha) \cdot \sum_{i=1}^n Q_{i,1}. \tag{3.10}$$

From another perspective, the optimal result vector R^* satisfies the following formula: $\lambda^* =$ F-score$^*(Q, R^*, \alpha)$, thus similarly we can derive

$$\sum_{i=1}^n (Q_{i,1} - \lambda^* \cdot \alpha) \cdot \mathbb{1}_{\{r_i^*=1\}} = \lambda^* \cdot (1-\alpha) \cdot \sum_{i=1}^n Q_{i,1}. \tag{3.11}$$

As λ^*, Q, and α are known, for ease of representation let us denote a fixed constant A and a function $h(R)$ as follows

$$\begin{cases} A = \lambda^* \cdot (1-\alpha) \cdot \sum_{i=1}^{n} Q_{i,1}, \\ h(R) = \sum_{i=1}^{n} (Q_{i,1} - \lambda^* \cdot \alpha) \cdot \mathbb{1}_{\{r_i=1\}}. \end{cases}$$

Then Equation 3.10 and 3.11 can be represented as: (1) for any $R \in \{1,2\}^2$, $h(R) \le A$; and (2) the optimal R^* satisfies $h(R^*) = A$. Next we prove that if we can derive $R' = \text{argmax}_R \; \{ \, h(R) \, \}$, then $R' = [r'_1, r'_2, \ldots, r'_n]$ is the optimal result vector (i.e., $R' = R^*$). From Equation 3.10, since $R' \in \{1,2\}^2$, we can derive $h(R') \le A$.

From $R' = \text{argmax}_R \; \{ \, h(R) \, \}$, we know that $\max_R\{ \, h(R) \, \} = h(R')$ and from Equation 3.11, we know that $\max_R\{ \, h(R) \, \} \ge h(R^*) = A$. So we have $h(R') \ge A$. As $h(R') \le A$, we have $h(R') = A$, or $\sum_{i=1}^{n}(Q_{i,1} - \lambda^* \cdot \alpha) \cdot \mathbb{1}_{\{r'_i=1\}} = \lambda^* \cdot (1-\alpha) \cdot \sum_{i=1}^{n} Q_{i,1}$, and finally we can derive λ^* from the above Equation:

$$\lambda^* = \frac{\sum_{i=1}^{n} Q_{i,1} \cdot \mathbb{1}_{\{r'_i=1\}}}{\alpha \cdot \sum_{i=1}^{n} \mathbb{1}_{\{r'_i=1\}} + (1-\alpha) \cdot \sum_{i=1}^{n} Q_{i,1}}.$$

As $\lambda^* = \text{F-score}^*(Q, R', \alpha)$, i.e., R' derives the optimal λ^*, we know $R' = R^*$. Then $R^* = \text{argmax}_R\{ \, h(R) \, \}$, i.e., $R^* = \text{argmax}_R \; \left\{ \sum_{i=1}^{n}(Q_{i,1} - \lambda^* \cdot \alpha) \cdot \mathbb{1}_{\{r_i=1\}} \right\}$. In order to maximize $h(R)$, we can set $\mathbb{1}_{\{r_i=1\}} = 1$ (or $r_i = 1$) if $Q_{i,1} \ge \lambda^* \cdot \alpha$ and $\mathbb{1}_{\{r_i=1\}} = 0$ (or $r_i = 2$) if $Q_{i,1} < \lambda^* \cdot \alpha$. Then we get

$$r_i^* = \begin{cases} 1 & \text{if} \quad Q_{i,1} \ge \lambda^* \cdot \alpha, \\ 2 & \text{if} \quad Q_{i,2} < \lambda^* \cdot \alpha. \end{cases}$$

Thus we have proved that there exists a threshold $\theta = \lambda^* \cdot \alpha$, such that $R^* = [\, r_1^*, r_2^*, \ldots, r_n^* \,]$ can be constructed as $r_i^* = 1$ if $Q_{i,1} \ge \lambda^* \cdot \alpha$ and $r_i^* = 2$ if otherwise. □

As $\lambda^* = \max_R \; \text{F-score}^*(Q, R, \alpha)$, Theorem 3.2 shows that for F-score*, a task's optimal result is related to the optimal value for F-score* w.r.t. Q, which takes all the tasks' distributions into consideration. This explains that why the claim for Accuracy* (Theorem 3.1), i.e., "a task's optimal result is only dependent on its own distribution" does not hold. Next we focus on the problem of how to efficiently derive λ^*.

We first show that this problem can be reduced to a 0-1 fractional programming (FP) problem, and then present an iterative algorithm to identify the optimal value (λ^*). Let $B = [b_1, b_2, \ldots, b_n]$, $D = [d_1, d_2, \ldots, d_n]$ denote two coefficient vectors, and β, γ denote two scalars. The 0-1 FP problem is defined as:

$$\max \quad f(\mathbf{z}) = \frac{\sum_{i=1}^{n}(z_i \cdot b_i) + \beta}{\sum_{i=1}^{n}(z_i \cdot d_i) + \gamma} \quad s.t. \quad \mathbf{z} \in \Omega \subseteq \{0,1\}^n$$

Comparing F-score* (Equation 5.4) with $f(\mathbf{z})$, we find that F-score* can actually be rewritten in the form of $f(\mathbf{z})$ as follows:

$$\begin{cases} z_i = \mathbb{1}_{\{r_i=1\}},\ b_i = Q_{i,1},\ d_i = \alpha \ \text{ for } 1 \le i \le n; \\ \beta = 0,\ \gamma = \sum_{i=1}^{n}(1-\alpha)\cdot Q_{i,1},\ \ \Omega = \{0,1\}^n. \end{cases} \tag{3.12}$$

Thus the problem of computing λ^* can be reduced to a 0-1 FP problem. The 0-1 FP problem can be efficiently solved based on the Dinkelbach framework [52]. We apply it to our problem, which consists of the following three steps:

Initialization: It first constructs a new function $g(\mathbf{z}, \lambda) = \sum_{i=1}^{n} (b_i - \lambda \cdot d_i) \cdot z_i$. Then it will iteratively update λ.

In our case, b_i, d_i and z_i ($1 \le i \le n$) can be represented following Equation 3.12 and $g(\mathbf{z}, \lambda) = \sum_{i=1}^{n}(Q_{i,1} - \lambda \cdot \alpha) \cdot \mathbb{1}_{\{r_i=1\}}$.

Iteration: Let λ_t denote the λ for the t-th iteration. For the first iteration, the algorithm initializes λ as a constant value λ_{init} (for example, 0): $\lambda_1 = \lambda_{init}$, and then computes $\mathbf{z}' = \text{argmax}_{\mathbf{z}}\ g(\mathbf{z}, \lambda_1)$. By plugging the newly computed $\mathbf{z}'$ into $f(\mathbf{z})$, the algorithm updates λ_1 to $\lambda_2 = f(\mathbf{z}')$. Then it repeats the above iteration with the updated λ (i.e., λ_2).

In our case, in the first iteration, with initial $\lambda_1 = \lambda_{init}$, we have to compute $R' = \text{argmax}_R \ \sum_{i=1}^{n}(Q_{i,1} - \lambda_1 \cdot \alpha) \cdot \mathbb{1}_{\{r_i=1\}}$, and $R' = [r'_1, r'_2, \ldots, r'_n]$ can be derived by setting

$$\mathbb{1}_{\{r'_i=1\}} = \begin{cases} 1 \quad \text{(i.e., } r'_i = 1\text{)} \ \text{ if } \ Q_{i,1} \geq \lambda_1 \cdot \alpha, \\ 0 \quad \text{(i.e., } r'_i = 2\text{)} \ \text{ if } \ Q_{i,1} < \lambda_1 \cdot \alpha. \end{cases}$$

Then λ_1 is updated as $\lambda_2 = \text{F-score}^*(Q, R', \alpha)$, and we repeat the above iteration with the updated λ_2.

<u>*Termination:*</u> The algorithm terminates at the c-th iteration when λ converges (or it is unchanged), i.e., $\lambda_{c+1} = \lambda_c$, and it returns λ_{c+1} which is computed for the c-th iteration.

The Dinkelbach framework guarantees that the iterative process will finally converge to the optimal objective value [52]. Thus our algorithm can return the optimal value for F-score*, i.e., $\lambda^* = \lambda_{c+1}$. The detailed algorithm is in Algorithm 1. For a given Q, In order to derive the optimal λ^* such that $\lambda^* = \max_R \ \text{F-score}^*(Q, R, \alpha)$, following the discussions in Section 3.4.2, we design Algorithm 1. It iteratively updates λ until convergence. Let λ_t denote the λ for the t-th iteration, so initially $\lambda_1 = \lambda_{init} = 0$. In the t-th iteration (λ_t is known), it first constructs a new result vector R' using the known λ_t (lines 5-7) and then update λ_t to $\lambda_{t+1} = \text{F-score}^*(Q, R', \alpha)$ (line 8) for the next iteration. The way to construct each r'_i ($1 \leq i \leq n$) in R' is based on comparing $Q_{i,1}$ with the threshold $\lambda_t \cdot \alpha$, i.e., $r'_i = 1$ if $Q_{i,1} \geq \lambda_t \cdot \alpha$ and $r_i = 2$ if otherwise. Finally it decides whether it converges (i.e., $\lambda_{t+1} = \lambda_t$) or not (lines 9-12). We next run the two counter-examples demonstrated in Example 2, and analyze the time complexity in the end.

Example 3. *Consider $\widehat{\alpha}$ and $\widehat{Q}$ in Example 2. In the 1st iteration, suppose $\lambda_1 = \lambda_{init} = 0$. As $\widehat{Q}_{1,1}$ and $\widehat{Q}_{2,1}$ are both $\geq \lambda_1 \cdot \widehat{\alpha} = 0$, then $R' = [1,1]$ and λ_1 is updated to $\lambda_2 = \text{F-score}(\widehat{Q}, R', \widehat{\alpha}) = \frac{0.35+0.9}{0.5 \cdot 2 + 0.5 \cdot (0.35+0.9)} = 0.77$. As $\lambda_2 \neq \lambda_1$, we continue. In the 2nd iteration, by comparing $\widehat{Q}_{1,1}$ and $\widehat{Q}_{2,1}$ with $\lambda_2 \cdot \widehat{\alpha} = 0.385$, we can*

Algorithm 1 Measure the Quality of Matrix Q for *F-score* (Chapter 3).

```
Input:  $Q$, $\alpha$
Output: $\lambda$
1:  $\lambda = 0$ ; // initialized as 0 ($\lambda_{init} = 0$)
2:  $R' = [\,]$ ;
3:  while True do
4:      $\lambda_{pre} = \lambda$; // record $\lambda$ for this iteration
5:      // construct new $R' = [r'_1, r'_2 \dots r'_n]$
6:      for $i = 1$ to $n$ do
7:          if $Q_{i,1} \geq \lambda \cdot \alpha$ then $r'_i = 1$ else $r'_i = 2$
8:      end for
9:      $\lambda = \frac{\sum_{i=1}^{n} Q_{i,1} \cdot \mathbb{1}_{\{r'_i=1\}}}{\sum_{i=1}^{n} [\, \alpha \cdot \mathbb{1}_{\{r'_i=1\}} + (1-\alpha) \cdot Q_{i,1} \,]}$; // F-score$^*(Q, R', \alpha)$
10:     if $\lambda_{pre} == \lambda$ then
11:         break
12:     else
13:         $\lambda_{pre} = \lambda$
14:     end if
15: end while
16: return $\lambda$
```

construct $R' = [2,1]$, and $\lambda_3 =$ F-score$(\widehat{Q}, R', \widehat{\alpha}) = \frac{0.9}{0.5 \cdot 1 + 0.5 \cdot (0.35+0.9)} = 0.8 \neq \lambda_2$. In the 3rd iteration, as the updated $\lambda_3 \cdot \widehat{\alpha} = 0.4$, we can construct $R' = [2,1]$. Since $\lambda_4 = \lambda_3 = 0.8$, it converges and $\widehat{\lambda}^* = 0.8$ is returned. Following Theorem 3.2, we can obtain the threshold $\widehat{\theta} = \widehat{\lambda}^* \cdot \widehat{\alpha} = 0.4$. Since $\widehat{Q}_{1,1} < \theta$ and $\widehat{Q}_{2,1} \geq \theta$, we have $\widehat{R}^* = [2,1]$.

If we consider α and Q in Example 2, then similarly we derive $\lambda^* = 0.62$, $\theta = \lambda^* \cdot \alpha = 0.31$, and $R^* = [1,1]$. The above two examples show that the approximation function F-score$^*(\cdot)$ also conforms to the two observations verified in Example 2. Moreover, F-score$^*(\cdot)$ gives us an intuitive explanation of why two observations occur: in the two examples, the optimal values of F-score* (λ^* and $\widehat{\lambda}^*$) affect the individual threshold (θ and $\widehat{\theta}$), and thus affecting the optimal result vectors (especially r_1^* and $\hat{r}_1^*$).

Time Complexity. As each iteration requires $\mathcal{O}(n)$ time and there are c iterations in total, the time complexity is $\mathcal{O}(c \cdot n)$. To evaluate the time complexity in practice, we conduct extensive simulated experiments by randomly generating Q and $\alpha \in [0,1]$, which shows that the time complexity of the algorithm linearly increases w.r.t. n. It converges very fast, and $c \leq 15$ when $n = 2000$ (Section 3.7.1).

3.5 Online Assignment Algorithms

Recall Definition 3.1, the quality of a given distribution matrix Q is measured as its maximal quality (or quality w.r.t R^*), i.e., $F(Q) = \max_R F^*(Q,R) = F^*(Q,R^*)$, where F can be *Accuracy* or *F-score*. To address the task assignment problem, one simple solution is to enumerate all feasible assignment vectors. For each one (X), Q^X can be constructed via Equation 3.1, and we compute the optimal result vector for Q^X, denoted as $R^X = \text{argmax}_R \; F^*(Q^X, R)$. Finally $X^* = \text{argmax}_X \; F(Q^X, R^X)$.

Obviously, this simple method is very expensive. To avoid enumerating $\binom{|S^w|}{k}$ feasible assignments, we propose two efficient algorithms: a Top-k Benefit Algorithm for Accuracy* (Section 3.5.1) and an Online Assignment Algorithm for F-score* (Section 3.5.2).

3.5.1 Accuracy*: Top-K Benefit Algorithm

Given Q^c and Q^w, let $R^c = [r_1^c, r_2^c, \ldots, r_n^c]$ and $R^w = [r_1^w, r_2^w, \ldots, r_n^w]$ denote their respective optimal result vectors, i.e., $r_i^c = \text{argmax}_j \; Q_{i,j}^c$ for $1 \leq i \leq n$ and $r_i^w = \text{argmax}_j \; Q_{i,j}^w$ for $q_i \in S^w$. As shown in Theorem 3.1, the choice of respective optimal result r_i^c (r_i^w) only depends on Q_i^c (Q_i^w). Therefore, based on the definition of Q^X (Equation 3.1), the optimal result vector for Q^X, denoted by $R^X = [r_1^X, r_2^X, \ldots, r_n^X]$, can be represented using R^c and R^w as follows:

$$r_i^X = \begin{cases} r_i^c & \text{if } x_i = 0, \\ r_i^w & \text{if } x_i = 1. \end{cases} \tag{3.13}$$

According to Definition 3.1, we aim to find the optimal feasible assignment X such that $\text{Accuracy}^*(Q^X, R^X) = \sum_{i=1}^n Q_{i,r_i^X}/n = \sum_{i=1}^n [\ (Q_{i,r_i^c}/n) \cdot \mathbb{1}_{\{x_i=0\}} + (Q_{i,r_i^w}/n) \cdot \mathbb{1}_{\{x_i=1\}}\ ]$ is maximized. We can further derive $\text{Accuracy}^*(Q^X, R^X) =$

$$\sum_{i=1}^{n} \frac{Q^c_{i,r_i^c}}{n} + \sum_{i=1}^{n} \frac{Q^w_{i,r_i^w} - Q^c_{i,r_i^c}}{n} \cdot \mathbb{1}_{\{x_i=1\}}. \tag{3.14}$$

As for each X, the value $\sum_{i=1}^n Q^c_{i,r_i^c}/n$ is fixed. Then for each task q_i, we can define the benefit of assigning it to worker w as $\mathit{Benefit}(q_i) = Q^w_{i,r_i^w} - Q^c_{i,r_i^c}$, which indicates that the function Accuracy* will be increased by $\mathit{Benefit}(q_i)/n$ if q_i is assigned to worker w. Therefore, the optimal assignment consists of k tasks with the largest benefits, and selecting them needs $\mathcal{O}(|S^w|) = \mathcal{O}(n)$ time[3]. Example 4 illustrates how to apply the *Top-k Benefit Algorithm* to assign tasks in Figure 3.2 when the function is set as Accuracy*.

Example 4. *Consider Q^c and Q^w in Figure 3.2. We can obtain $R^c = [1, 1, 2, 1, 1, 2]$ (or $[1, 1, 2, 2, 1, 2]$) and $R^w = [1, 1, 0, 1, 0, 2]$.*[4] *For each $q_i \in S^w$, we compute its benefit as follows: $\mathit{Benefit}(q_1) = Q^w_{1,r_1^w} - Q^c_{1,r_1^c} = 0.123$, $\mathit{Benefit}(q_2) = 0.212$, $\mathit{Benefit}(q_4) = 0.25$ and $\mathit{Benefit}(q_6) = 0.175$. So q_2 and q_4 which have the highest benefits will be assigned to worker w.*

3.5.2 F-score*: Online Assignment Algorithm

Compared with Accuracy*, the online assignment for F-score* is more challenging. The main reason is that as shown in Section 3.4.2, based on F-score*, the optimal result for each task is not only dependent on its own distribution. Given a feasible X (Q^X can be constructed), deriving the optimal result vector R^X for Q^X is not as straightforward as Equation 3.13 for Accuracy*.

Next we show how to efficiently solve the task assignment problem for

[3]The problem that finds the top k elements in an array can be solved linearly using the PICK algorithm [26].

[4]Note that for $q_i \notin S^w$, Q_i^w does not need to be computed, so we set $r_i^w = 0$ for $q_i \notin S^w$ and it will never be used.

F-score*. Recall that X^* denotes the optimal assignment and let δ^* denote the optimal objective value, i.e.,

$$\delta^* = \max_R \text{ F-score}^*(Q^{X^*}, R, \alpha). \tag{3.15}$$

The basic idea of our solution is to first initialize $\delta_{init} \leq \delta^*$ (say $\delta_{init} = 0$), and then iteratively update the initial δ_{init} to δ^* until convergence. Since δ^* is unknown, the main problem is how to ensure that δ_{init} is always *increasing* until it reaches δ^*. More formally, let δ_t denote the δ at the t-th iteration. Given δ_t, the updated δ_{t+1} should satisfy the following two properties:

Property 1: if $\delta_t < \delta^*$, then the updated δ_{t+1} should satisfy $\delta_t < \delta_{t+1} \leq \delta^*$;

Property 2: if $\delta_t = \delta^*$, then the updated δ_{t+1} should satisfy $\delta_t = \delta_{t+1} = \delta^*$.

Intuitively, Property 1 guarantees that starting from $\delta_{init} < \delta^*$, δ_{init} will be iteratively increased until δ^*. Property 2 guarantees that at convergence ($\delta_t = \delta_{t+1}$), we can get $\delta_{t+1} = \delta^*$. There are two challenges in solving the problem:
(1) Given δ_t, how can we construct δ_{t+1} such that the above two properties hold?
(2) The update should be solved efficiently to satisfy the performance requirement for task assignment.

To address the first challenge, we present our designed update in Definition 3.2 as follows:

Definition 3.2 (Update). *Given δ_t, Q^c, Q^w, α, and S^w, the update from δ_t to δ_{t+1} is defined as*

$$\delta_{t+1} = \max_X \textit{ F-score}^*(Q^X, \widehat{R}^X, \alpha), \tag{3.16}$$

where for each feasible X (Q^X can be constructed via Equation 3.1), $\widehat{R}^X = [\widehat{r}_1^X, \widehat{r}_2^X, \ldots, \widehat{r}_n^X]$ is constructed based on δ_t, i.e.,

$$\widehat{r}_i^X = \begin{cases} 1 & \textit{if } \; Q_{i,1}^X \geq \delta_t \cdot \alpha, \\ 2 & \textit{if } \; Q_{i,1}^X < \delta_t \cdot \alpha. \end{cases} \tag{3.17}$$

To help understand the meaning of δ_{t+1} in Definition 3.2, we will present a brute-force method to obtain δ_{t+1}. This method enumerates all possible feasible assignment vectors. For each feasible X, as Q^X and $\widehat{R}^X$ can be constructed following Equation 3.1 and 3.17 respectively, then F-score$^*(Q^X, \widehat{R}^X, \alpha)$ can be computed. By comparing the computed values for all assignment vectors, we can obtain the maximum value, i.e., δ_{t+1}. Theorem 3.3 formally proves that the updated δ_{t+1} following Definition 3.2 satisfies the two properties.

Theorem 3.3. *The defined δ_{t+1} (in Definition 3.2) satisfies Property 1 and Property 2.*

Proof. As mentioned before, in the definition of computing δ_{t+1} (Definition 3.2), the construction of R^X (Equation 3.17) is similar to the construction of R' in choosing the optimal result vector for a given Q (Algorithm 1). Thus the basic idea of the proof is to make a comparison with Algorithm 1.

Before making the comparison, we present some theoretical results proved in [52] for the Dinkelbach framework (which applies to Algorithm 1). It has proved that starting from the initial $\lambda_{init} \leq \lambda^*$, the λ_{init} will be iteratively *increased* to λ^* until convergence. It means that the update from λ_t to λ_{t+1} in Algorithm 1 also conforms to our two properties, i.e., (1) if $\lambda_t < \lambda^*$, then $\lambda_t < \lambda_{t+1} \leq \lambda^*$ and (2) if $\lambda_t = \lambda^*$, then $\lambda_t = \lambda_{t+1} = \lambda^*$.

Recall that δ^* and X^* respectively denote the optimal value and the optimal assignment vector, and we can derive $\delta^* = \max_R$ F-score$^*(Q^{X^*}, R, \alpha)$. Thus, if Algorithm 1 takes Q^{X^*} and α as the input, then $\lambda^* = \max_R$ F-score$^*(Q^{X^*}, R, \alpha)$, which is exactly δ^*, i.e., $\lambda^* = \delta^*$.

The comparison is conducted based on comparing our online assignment algorithm with Algorithm 1, which takes Q^{X^*} and α as the input. As derived above, the optimal value for both algorithms are the same (i.e., δ^*).

To prove Property 1, suppose both algorithms start with $\delta_t < \delta^*$, and they update their respective values (denoted as δ_{t+1} and λ_{t+1} respectively) for the next iteration as follows

$$\begin{cases} \delta_{t+1} = \max_X \text{ F-score}^*(Q^X, \widehat{R}^X, \alpha), \\ \lambda_{t+1} = \text{F-score}^*(Q^{X^*}, R', \alpha). \end{cases} \tag{3.18}$$

Note that for a feasible X, $\widehat{R}^X$ is constructed by comparing each $Q^X_{i,1}$ ($1 \le i \le n$) with the threshold $\delta_t \cdot \alpha$. And R' is similarly constructed by comparing each $Q^{X^*}_{i,1}$ ($1 \le i \le n$) with the same threshold $\delta_t \cdot \alpha$. As X^* is also a feasible assignment, we have $(Q^{X^*}, R') \in \{(Q^X, \widehat{R}^X) \mid X \text{ is feasible }\}$. Then by considering Equation 3.18, we derive $\delta_{t+1} \ge \lambda_{t+1}$. As mentioned above, since $\delta_t < \delta^*$, the properties in [52] guarantee that $\lambda_{t+1} > \delta_t$, then we derive $\delta_{t+1} > \delta_t$. As δ^* is the optimal objective value, we can finally derive $\delta_t < \delta_{t+1} \le \delta^*$, which proves Property 1.

To prove Property 2, we apply the same comparison. Suppose both algorithms start with $\delta_t = \delta^*$, then they are updated by Equation 3.18. Similarly we have $\delta_{t+1} \ge \lambda_{t+1}$. As $\delta_t = \delta^*$, following the properties in [52], we have $\lambda_{t+1} = \delta_t = \delta^*$. Since $\delta_{t+1} \le \delta^*$, we derive $\delta_t = \delta_{t+1} = \delta^*$, which proves Property 2. □

To address the second challenge above, as shown in Theorem 3.4, we find that this problem (computing δ_{t+1} in Definition 3.2) can actually be reduced to a 0-1 FP problem and efficiently solved by leveraging the Dinkelbach framework (similar to Section 3.4.2).

Theorem 3.4. *The problem of computing δ_{t+1} (Definition 3.2) can be reduced to a 0-1 FP problem.*

Proof. Given Q^c, Q^w and a feasible X, from Equation 3.1 we know that Q^X can be expressed by X, Q^c and Q^w, then we have

$$Q^X_{i,1} = x_i \cdot Q^w_{i,1} + (1 - x_i) \cdot Q^c_{i,1}.$$

Given Q^X and δ_t, we know that $\widehat{R}^X$ is constructed by setting $\widehat{r}^X_i = 1$ if $Q^X_{i,1} \ge \delta_t \cdot \alpha$ and $\widehat{r}^X_i = 2$ if $Q^X_{i,1} < \delta_t \cdot \alpha$. It means that if we construct $\widehat{R}^c$ ($\widehat{R}^w$) by setting $\widehat{r}^c_i = 1$

($\widehat{r}_i^w = 1$) if $Q_{i,1}^c \geq \delta_t \cdot \alpha$ ($Q_{i,1}^w \geq \delta_t \cdot \alpha$) and $\widehat{r}_i^c = 2$ ($\widehat{r}_i^w = 2$) if $Q_{i,1}^c < \delta_t \cdot \alpha$ ($Q_{i,1}^w < \delta_t \cdot \alpha$), then $\widehat{R}^X$ can be expressed with the given $\widehat{R}^c$ and $\widehat{R}^w$ as follows:

$$\mathbb{1}_{\{\widehat{r}_i^X=1\}} = x_i \cdot \mathbb{1}_{\{\widehat{r}_i^w=1\}} + (1-x_i) \cdot \mathbb{1}_{\{\widehat{r}_i^c=1\}}.$$

As $x_i = \{0,1\}$, if we plug the above derived $Q_{i,1}^X$ and $\mathbb{1}_{\{\widehat{r}_i^X=1\}}$ into

$$\text{F-score}^*(Q^X, \widehat{R}^X, \alpha) = \frac{\sum_{i=1}^n Q_{i,1}^X \cdot \mathbb{1}_{\{\widehat{r}_i^X=1\}}}{\sum_{i=1}^n [\, \alpha \cdot \mathbb{1}_{\{\widehat{r}_i^X=1\}} + (1-\alpha) \cdot Q_{i,1}^X \,]},$$

and set the parameters b_i, d_i ($1 \leq i \leq n$), β and γ following
(1) $b_i = d_i = 0$ for $q_i \notin S^w$, and
(2) b_i, d_i ($q_i \in S^w$), β and γ are set as:

$$\begin{cases} b_i = Q_{i,1}^w \cdot \mathbb{1}_{\{\widehat{r}_i^w=1\}} - Q_{i,1}^c \cdot \mathbb{1}_{\{\widehat{r}_i^c=1\}} \\ d_i = \alpha \cdot (\mathbb{1}_{\{\widehat{r}_i^w=1\}} - \mathbb{1}_{\{\widehat{r}_i^c=1\}}) + (1-\alpha) \cdot (Q_{i,1}^w - Q_{i,1}^c) \\ \beta = \sum_{i=1}^n Q_{i,1}^c \cdot \mathbb{1}_{\{\widehat{r}_i^c=1\}} \\ \gamma = \sum_{i=1}^n [\, \alpha \cdot \mathbb{1}_{\{\widehat{r}_i^c=1\}} + (1-\alpha) \cdot Q_{i,1}^c \,], \end{cases}$$

then the problem is to maximize

$$\text{F-score}^*(Q^X, \widehat{R}^X, \alpha) = \frac{\sum_{i=1}^n (x_i \cdot b_i) + \beta}{\sum_{i=1}^n (x_i \cdot d_i) + \gamma}. \tag{3.19}$$

s.t. X is a feasible assignment vector. As Q^c, Q^w, $\widehat{R}^c$, $\widehat{R}^w$, S^w, α, and δ_t are known, then x_i ($1 \leq i \leq n$) are the only unknown variables. Let all the feasible assignment vectors form the subspace $\Omega \subseteq \{0,1\}^n$ where $|\Omega| = \binom{|S^w|}{k}$. If we set $z_i = x_i$ ($1 \leq i \leq n$), then each $\mathbf{z} \in \Omega$ corresponds to a feasible assignment vector X. So the problem is to maximize $\frac{\sum_{i=1}^n (z_i \cdot b_i) + \beta}{\sum_{i=1}^n (z_i \cdot d_i) + \gamma}$ s.t. $\mathbf{z} \in \Omega \subseteq \{0,1\}^n$, which is a 0-1 FP problem. $\square$

Our designed algorithm, called *F-score Online Assignment Algorithm*, computes the optimal assignment. In each iteration, it calls the *Update Algorithm*, which leverages the Dinkelbach framework to compute the updated δ_{t+1}. Note that the Dinkelbach framework not only returns the updated δ_{t+1}, but also derives the corresponding assignment X' such that $\delta_{t+1} = \text{F-score}^*(Q^{X'}, \widehat{R}^{X'}, \alpha)$. Thus, at the time that δ converges, the δ^* and its corresponding optimal assignment X^* are both returned by the *Update Algorithm*. Algorithms 2 and 3 give the pseudo-codes of *F-score Online Assignment Algorithm* and *Update Algorithm*.

Algorithm 2 is the *F-score Online Assignment Algorithm*, which iteratively updates δ_{init} until convergence. In each iteration (lines 3-9), it first calls the *Update Algorithm* (the details are introduced in the following two paragraphs) to update δ (line 5), and then decide whether it converges or not (lines 6-9). Finally, it uses the assignment vector X corresponding to the converging δ to construct a HIT (lines 10-14). The converging δ and its corresponding assignment vector X are both optimal (i.e., respectively δ^* and X^*).

As we have proved that that the problem of computing λ_{t+1} can be reduced to a 0-1 FP problem (Theorem 4), following the Dinkelbach framework [52] as discussed in Section 3.4.2, the key is to solve the sub-problem $\mathbf{z}' = argmax_{\mathbf{z}}$ $\sum_{i=1}^{n}(b_i - \lambda \cdot d_i) \cdot z_i$. In Theorem 3.4, in order to get $\mathbf{z}'$, due to the constraint of Ω (containing all feasible assignments), we should select k tasks in S^w ($q_i \in S^w$) with the largest values of $(b_i - \lambda \cdot d_i)$.

We present the detailed *Update Algorithm* in Algorithm 3, which leverages Dinkelbach framework [52] to efficiently compute δ_{t+1} based on δ_t. For efficiency's sake it first constructs $\widehat{R}^c$, $\widehat{R}^w$ (lines 5-9), and b_i, d_i ($b_i \in S^w$), β, γ (line 10) following the details in Theorem 3.4 . Then it iteratively updates λ_{init} until convergence (lines 11-21). In each iteration (lines 13-21), with known λ, it selects k tasks in S^w ($q_i \in S^w$) with the largest values of $(b_i - \lambda \cdot d_i)$ (line 14), which needs $\mathcal{O}(n)$ time by PICK algorithm [26]. Then it updates λ following Equation 3.19 (line 17). Finally it decides whether it converges or not (lines 18-21).

Time Complexity: We analyze the time complexity of *F-score Online Assignment*

Algorithm 2 F-score Online Assignment (Chapter 3).

Input: Q^c, Q^w, α, k, S^w
Output: HIT

```
1: $\delta = 0$ ; // initialized as 0 ($\delta_{init} = 0$)
2: while True do
3:    $\delta_{pre} = \delta$
4:    // get the updated $\delta_{t+1}$ and its corresponding $X$
5:    $X, \delta = Update(Q^c, Q^w, \alpha, k, S^w, \delta)$ (Algorithm 3)
6:    if $\delta_{pre} == \delta$ then
7:       break
8:    else
9:       $\delta_{pre} = \delta$
10:   end if
11: end while
12: // construct HIT based on the returned $X$
13: for $i = 1$ to $n$ do
14:    if $x_i == 1$ then
15:       HIT = HIT $\cup \{q_i\}$
16:    end if
17: end for
18: return HIT
```

Algorithm. It is an iterative algorithm, where it runs *Update Algorithm* in each iteration. Following Dinkelbach framework, *Update Algorithm* adopts an iterative approach and takes $\mathcal{O}(n)$ in each iteration. Suppose *F-score Online Assignment Algorithm* requires u iterations to converge and *Update Algorithm* requires v iterations to converge, then the total time complexity is $\mathcal{O}(u \cdot v \cdot n)$. We conduct extensive simulated experiments by randomly generating Q^c and Q^w (n = 2000), and varying $\alpha \in [0, 1]$, which shows that the bound $u \cdot v <= 10$ in practice (Section 3.7.1).

Example 5. *Given S^w, k, Q^c and Q^w in Figure 3.2, if the evaluation metric is set as F-score* with α = 0.75, we next derive the optimal assignment for worker w. With an initial $\delta_1 = \delta_{init} = 0$, we need to get δ_2. The brute-force way*[5] *is to enumerate all 6 feasible assignments, where for the first $X = [1, 1, 0, 0, 0, 0]$, we construct Q^X. As $Q^X_{i,1} \geq \delta_1 \cdot \alpha = 0$ for all $1 \leq i \leq n$, thus $\widehat{R}^X = [1, 1, 1, 1, 1]$ and*

[5]Here we present the brute-force way for illustration purpose.

Algorithm 3 Update (Chapter 3).

```
Input:  $Q^c$, $Q^w$, $\alpha$, $k$, $S^w$, $\delta$
Output:  $X, \lambda$
1: $\lambda = 0$ ; // initialized as 0 ($\lambda_{init} = 0$)
2: $X = [\,]$ ;
3: $\widehat{R}^c = [\,]$; $\widehat{R}^w = [\,]$;
4: $b = d = [0, 0, \ldots, 0]$; $\beta = 0$; $\gamma = 0$;
5: // construct $\widehat{R}^c$ ($\widehat{R}^w$) by comparing $Q^c$ ($Q^w$) with $\delta \cdot \alpha$; (lines 6-9)
6: for $i = 1$ to $n$ do
7:    if $Q^c_{i,1} \geq \delta \cdot \alpha$ then $\widehat{r}^c_i = 1$ else $\widehat{r}^c_i = 2$
8: end for
9: for $q_i \in S^w$ do
10:    if $Q^w_{i,1} \geq \delta \cdot \alpha$ then $\widehat{r}^w_i = 1$ else $\widehat{r}^w_i = 2$
11: end for
12: Compute $b_i$, $d_i$ ($1 \leq i \leq n$) and $\beta$, $\gamma$ following the proof in Theorem 3.4;
13: // Update $\lambda$ from $\lambda_{init}$ until convergence; (line 12-21)
14: while True do
15:    $\lambda_{pre} = \lambda$
16:    compute $TOP$, a set which contains $k$ tasks in $S^w$ that correspond to the
       highest value of $b_i - \lambda \cdot d_i$;
17:    for $i = 1$ to $n$ do
18:       if $q_i \in TOP$ then $x_i = 1$ else $x_i = 0$
19:    end for
20:    $\lambda = \frac{\sum_{i=1}^{n}(\ x_i \cdot b_i\ ) + \beta}{\sum_{i=1}^{n}(\ x_i \cdot d_i\ ) + \gamma}$ ;
21:    if $\lambda_{pre} == \lambda$ then
22:       break
23:    else
24:       $\lambda_{pre} = \lambda$
25:    end if
26: end while
27: return $X, \lambda$
```

F-score$^*(Q^X, \widehat{R}^X, \alpha) = 0.68$. By considering all 6 feasible X, we derive the maximal F-score* value, i.e., 0.7, which corresponds to $X = [0, 1, 0, 1, 0, 0]$. Then $\delta_2 = 0.7$ and as $\delta_2 \neq \delta_1$, we continue with δ_2. Again consider $X = [1, 1, 0, 0, 0, 0]$, as $\delta_2 \cdot \alpha = 0.525$, by comparing each $Q^X_{i,1}$ ($1 \leq i \leq n$) with 0.525, we derive $\widehat{R}^X = [1, 1, 0, 0, 1, 0]$ and F-score$^*(Q^X, \widehat{R}^X, \alpha) = 0.832$. By considering all 6 feasible X, the assignment $X = [1, 1, 0, 0, 0, 0]$ corresponds to the maximal F-score*, and $\delta_3 = 0.832 \neq \delta_2$. In* *the third iteration, similarly the assignment* $X = [1, 1, 0, 0, 0, 0]$ *corresponds to the* $\delta_4 = 0.832$*. As* $\delta_4 = \delta_3$*, we have* $\delta^* = 0.832$ *and return* $X^* = [1, 1, 0, 0, 0, 0]$*.*

Compared with Accuracy* *in Example 4, which assigns* q_2 *and* q_4 *with the highest benefits, here the optimal assignment is* q_1 *and* q_2 *if the evaluation metric is F-score** *with* $\alpha = 0.75$*. The reason is that* $\alpha = 0.75$ *focuses on Precision, and it tries to assign tasks such that the estimated probability for the target label* L_1 *is of high confidence (or* $Q_{i,1}$ *is high). Thus it is more beneficial to assign* q_1 *compared with* q_4*, as* $Q^w_{1,1}$ *(0.818)* $\geq Q^w_{4,1}$ *(0.75).*

3.6 Computing the Current and Estimated Distribution Matrices

In this section, we examine how to compute Q^c and Q^w in Section 3.6.1 and 3.6.3 respectively. Since computing these two matrices requires some parameters, Section 3.6.2 discusses the parameters and the existing heuristics to compute them.

3.6.1 Current Distribution Matrix

When a worker completes a HIT, based on $D = \{D_1, D_2, \ldots, D_n\}$, we compute the parameters (including prior probability and worker model) and Q^c. As $Q^c_{i,j}$ represents the probability that task q_i's true label is L_j, we compute $Q^c_{i,j}$ based on the answer set of q_i, i.e., D_i. From Bayes' theorem we get

$$Q^c_{i,j} = P(t_i = j \mid D_i) = \frac{P(D_i \mid t_i = j) \cdot P(t_i = j)}{P(D_i)}. \tag{3.20}$$

Thus $Q^c_{i,j} \propto P(D_i \mid t_i = j) \cdot P(t_i = j)$. It means that if we derive $P(D_i \mid t_i = j) \cdot P(t_i = j)$ for each label L_j ($1 \leq j \leq \ell$), then we can normalize and finally get $Q^c_{i,j}$ for $1 \leq j \leq \ell$. We next discuss how to compute $P(t_i = j)$ and $P(D_i \mid t_i = j)$:

(1) $P(t_i = j)$ is the prior probability which represents that a task's true label is

L_j, and it is commonly regarded as the proportion of tasks whose true label is L_j, which is the same among different tasks, so w.l.o.g., we denote $p_j = P(t_i = j)$. Existing works [19,91,92,192] usually estimated the prior as the expected fraction of tasks whose ground truth is label L_j, i.e.,

$$p_j = \mathbb{E}[\,\frac{\sum_{i=1}^{n} \mathbb{1}_{\{t_i=j\}}}{n}\,] = \frac{\sum_{i=1}^{n} Q^c_{i,j}}{n}.$$

(2) $P(D_i \mid t_i = j)$ is the probability that the answer set for task q_i is D_i given that the task q_i's true label is L_j. Assume that the answers in D_i are independently answered by different workers (which was also adopted in [49,80,92,127,211]). Let a_i^w denote the index of answered label for q_i by worker w, then

$$P(D_i \mid t_i = j) = \prod_{(w,j') \in D_i} P(a_i^w = j' \mid t_i = j),$$

where $P(a_i^w = j' \mid t_i = j)$ is the probability that worker w answers label $L_{j'}$ given the true label is L_j, which can be expressed by worker model (Section 3.6.2). Initially, Q^c is set as uniform distribution for each task q_i, i.e., $Q^c_{i,j} = \frac{1}{\ell}$ for $1 \leq j \leq \ell$.

3.6.2 Parameters

For the issue about how to model a worker's quality, several works [80,100, 127,211] define a worker w's quality as a single value $m^w \in [0,1]$ called Worker Probability (WP) and

$$P(a_i^w = j' \mid t_i = j) = \begin{cases} m^w, \text{ for } j = j' \\ \frac{1-m^w}{\ell-1}, \text{ for } j \neq j' \end{cases}.$$

Some other works [19,92,200] define a worker w's quality as a $\ell \times \ell$ matrix M^w called Confusion Matrix (CM) and

$$P(a_i^w = j' \mid t_i = j) = M^w_{j,j'} \text{ for } 1 \leq j, j' \leq \ell.$$

For example, if an application has two labels, an example of WP for worker w is $m^w = 0.6$ and an example of CM for worker w is $M^w = \begin{bmatrix} 0.6 & 0.4 \\ 0.3 & 0.7 \end{bmatrix}$. In experiments (Section 3.7.2) we study the properties of different worker models on real datasets. For the initialization of WP and CM, each worker can be assumed as a perfect worker in the beginning (the assumption made by some prior work, e.g., Ipeirotis et al. [92]). Then for WP, $m^w = 1$; while for CM, $M^w_{j,j} = 1$ for $1 \le j \le \ell$ and $M^w_{j,j'} = 0$ for $j \neq j'$. Next Example 6 shows how to compute Q^c based on WP and prior.

Example 6. *Suppose a task q_2 with three labels ($\ell = 3$) has been answered by two workers: $D_2 = \{(w_1, L_3), (w_2, L_1)\}$, where the worker models are $m^{w_1} = 0.7$, $m^{w_2} = 0.6$ and the priors are $p_1 = p_2 = p_3 = \frac{1}{3}$. In order to compute Q^c_2, based on Equation 3.20, we get $Q^c_{2,1} = P(t_2 = 1 \mid D_2) \propto P(a_2^{w_1} = 3 \mid t_2 = 1) \cdot P(a_2^{w_2} = 1 \mid t_2 = 1) \cdot P(t_2 = 1) = \frac{1-m^{w_1}}{\ell-1} \cdot m^{w_2} \cdot p_1 = 0.03$ and similarly $Q^c_{2,2} \propto 0.01$, $Q^c_{2,3} \propto 0.0467$. By normalization, we get $Q^c_2 = [0.346, 0.115, 0.539]$.*

To compute the worker model and prior parameters, we leverage the EM [49] algorithm, which uses the answer set as input, and adopts an iterative approach (in each iteration, E-step and M-step are applied) to update all parameters until convergence. The EM algorithm has been widely used [19,91,92,192] to infer the parameters and has a fast convergence rate. There are some other works [91,92,169,182,183,200] studying how to derive worker model and prior parameters, and they can be easily adapted to our system. Note that the EM algorithm is run regularly offline (e.g., every 100 answers), thus it could detect the change of workers' qualities gradually (e.g., at first a worker is a bad worker, and as time goes by, the worker improves his/her quality). It might also be an interesting future work to see whether more recent input from workers should be given a higher weight (e.g., use a time-decaying function to weigh the past input of the worker).

3.6.3 Estimated Distribution Matrix

We next discuss how to compute Q^w based on Q^c and w's worker model. For each $q_i \in S^w$, the computation of Q_i^w consists of two steps: in the first step, we estimate l_i^w, which is denoted as the index of the label that worker w will answer for q_i; in the second step, we construct an estimated answer set (denoted as D_i^w) for q_i by adding a tuple containing l_i^w into D_i, i.e., $D_i^w = D_i \cup \{(w, l_i^w)\}$, and then Q_i^w can be computed based on D_i^w. We next talk about the details.

First Step: In order to estimate l_i^w, we first compute the label distribution that worker w will answer q_i. We can derive $P(a_i^w = j' \mid D_i)$ by considering all possible true labels of task q_i:

$$P(a_i^w = j' \,|D_i) = \sum_{j=1}^{\ell} P(a_i^w = j' \mid t_i = j, D_i) \cdot P(t_i = j \,|D_i). \tag{3.21}$$

(1) Given $t_i = j$ is known, $a_i^w = j'$ is independent of D_i, so $P(a_i^w = j' \mid t_i = j, D_i) = P(a_i^w = j' \mid t_i = j)$, which can be derived by w's worker model. (2) We have $Q_{i,j}^c = P(t_i = j \mid D_i)$. So we can compute $P(a_i^w = j' \mid D_i)$ for $1 \le j' \le \ell$.

Next, we predict l_i^w by capturing the label distribution, and the weighted random sampling method [59] is leveraged to select a label index $l_i^w \in \{1, 2, \ldots, \ell\}$, where the label index j has the probability $P(a_i^w = j \mid D_i)$ to be sampled out.

Second Step: To derive Q_i^w, we first construct $D_i^w = D_i \cup \{(w, l_i^w)\}$, then $Q_{i,j}^w = P(t_i = j \mid D_i^w) \propto P(D_i^w \mid t_i = j) \cdot p_j$. Similar to Section 3.6.1, we get $P(D_i^w \mid t_i = j) = P(a_i^w = l_i^w \mid t_i = j) \cdot \prod_{(w,j') \in D_i} P(a_i^w = j' \mid t_i = j)$. Take a further step, by considering Equation 3.20, we get

$$Q_{i,j}^w \propto Q_{i,j}^c \cdot P(a_i^w = l_i^w \mid t_i = j), \tag{3.22}$$

thus Q_i^w can be computed by normalization if we get $Q_{i,j}^c \cdot P(a_i^w = l_i^w \mid t_i = j)$ for $1 \le j \le \ell$. We next give an example (Example 7) to show how to compute

Q^w based on Q^c in Figure 3.2.

Example 7. *Given Q^c in Figure 3.2, suppose worker w with $m^w = 0.75$ requests a HIT. To compute Q^w, we take Q^w_1 as an example. In the first step we derive the label distribution to predict ℓ^w_1. Following Equation 3.21, $P(a^w_1 = 1 \,|D_1) = \sum_{j=1}^{2} P(a^w_1 = 1 \mid t_1 = j) \cdot P(t_1 = j \,|D_1) = 0.75 \cdot 0.8 + 0.25 \cdot 0.2 = 0.65$ and $P(a^w_1 = 2 \,|D_1) = 0.35$, thus the label distribution is $[0.65, 0.35]$. After weighted random sampling, suppose we get $\ell^w_1 = 1$. In the second step, following Equation 3.22, the proportion can be derived by multiplying $P(a^w_1 = 1 \mid t_1 = j)$ to $Q^c_{1,j}$ $(1 \leq j \leq 2)$, which is $(0.8 \cdot 0.75)$: $(0.2 \cdot 0.25)$. To normalize the proportion, we get $Q^w_2 = [0.923, 0.077]$. Similarly other Q^w_i for $q_i \in S^w$ can be computed and Q^w can be constructed in Figure 3.2.*

3.7 Experiments

We have conducted experiments on both simulated datasets (Section 6.1) and real datasets (Section 6.2).

3.7.1 Experiments on Simulated Data

We first describe the experimental settings (Section 6.1.1), then evaluate the performance of F-score* (Section 6.1.2), and finally examine the online assignment algorithms (Section 6.1.3)

Settings for Simulated Data

We show how to generate distribution matrices in simulated experiments. For *F-score*, as it focuses on a target label (L_1), to generate an $n \times 2$ distribution matrix Q, each task q_i's ($1 \leq i \leq n$) distribution is generated as follows: first the probability for target label is randomly generated as $Q_{i,1} \in [0, 1]$, then $Q_{i,2} = 1 - Q_{i,1}$. For *Accuracy*, to generate an $n \times \ell$ distribution matrix Q, for each task q_i ($1 \leq i \leq n$), we randomly generate $Q_{i,j} \in [0, 1]$ for $1 \leq j \leq \ell$, and then normalize Q_i to get a distribution. To achieve statistical significance, we perform each experiment for 1000 times and record its average value. Experiments are implemented in Python on a 16GB memory Ubuntu server.

Evaluating Our Techniques for F-Score*

As directly computing $\mathbb{E}[\ \textit{F-score}(T, R, \alpha)\ ]$ is inefficient, an approximation function F-score$^*(Q, R, \alpha)$ is proposed in Section 3.4.2. In this section, we evaluate this approximation on randomly generated distribution matrices. We first evaluate the approximation error, and then show the performance of its optimal result vector selection algorithm.

Approximation Error. We study how the proposed function F-score$^*(Q, R, \alpha)$ is approximate to $\mathbb{E}[\ \textit{F-score}(T, R, \alpha)\ ]$ from Figure 3.3(a)-(c). In Figure 3.3(a), we vary $\alpha \in [0, 1]$ and $n \in [20, 50]$ to observe the approximation error. For a fixed α and n, we randomly generate Q and result vector R, and record the approximation error $\epsilon = |\ \text{F-score}^*(Q, R, \alpha) - \mathbb{E}[\ \textit{F-score}(T, R, \alpha)\ ]\ |$ in the figure (note that the error is averaged over 1000 repeated trials). It shows that as n varies in $[20, 50]$, the approximation error decreases, and when $n = 20$, the errors for different α are smaller than 0.5%. As α varies in $[0, 1]$, the average error reaches the highest at around $\alpha = 0.5$. Moreover, an interesting observation is that the curve is not symmetric, especially when $\alpha = 1$ (which is *Precision*) and $\alpha = 0$ (which is *Recall*). The reason is as follows:

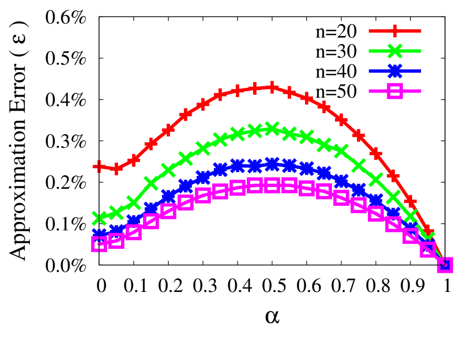
(1) For *Precision*, we can express $\mathbb{E}[\ \textit{Precision}(T, R)\ ]$ as

$$\mathbb{E}[\ \frac{\sum_{i=1}^{n} \mathbb{1}_{\{t_i=1\}} \cdot \mathbb{1}_{\{r_i=1\}}}{\sum_{i=1}^{n} \mathbb{1}_{\{r_i=1\}}}\ ] = \frac{\sum_{i=1}^{n} Q_{i,1} \cdot \mathbb{1}_{\{r_i=1\}}}{\sum_{i=1}^{n} \mathbb{1}_{\{r_i=1\}}} = \text{F-score}^*(Q, R, 1),$$

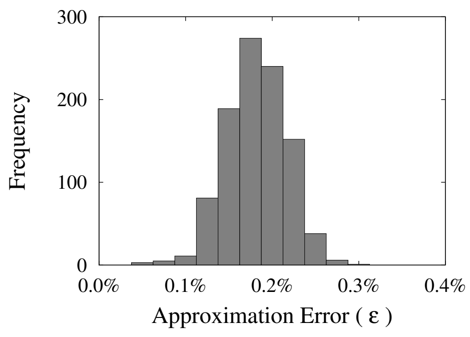
thus we have $\epsilon = 0$ as $\alpha = 1$;

(2) For *Recall*, we can express $\mathbb{E}[\ \textit{Recall}(T, R)\ ]$ as

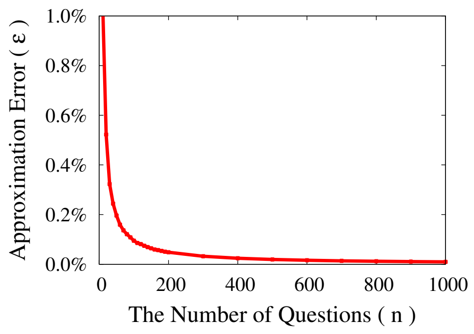
$$\mathbb{E}[\ \frac{\sum_{i=1}^{n} \mathbb{1}_{\{t_i=1\}} \cdot \mathbb{1}_{\{r_i=1\}}}{\sum_{i=1}^{n} \mathbb{1}_{\{t_i=1\}}}\ ] \approx \frac{\sum_{i=1}^{n} Q_{i,1} \cdot \mathbb{1}_{\{r_i=1\}}}{\sum_{i=1}^{n} Q_{i,1}} = \text{F-score}^*(Q, R, 0),$$



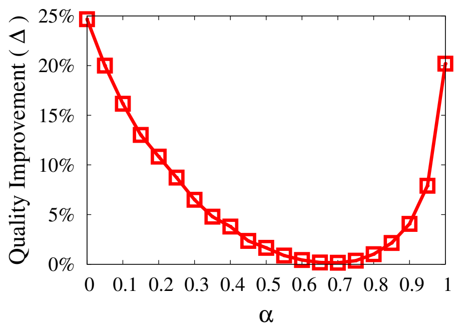
(a) Error: Varying α



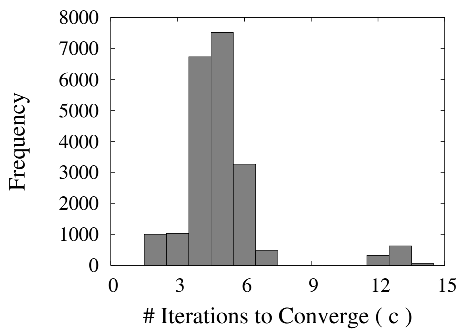
(b) Error: Frequency



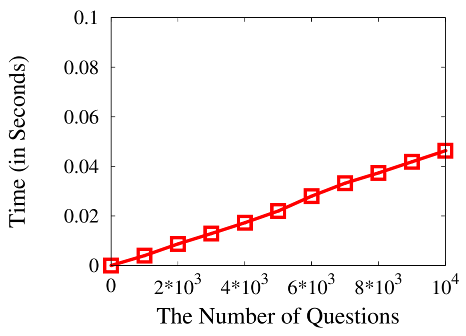
(c) Error: Varying n



(d) Improvement: Varying α



(e) Efficiency: Convergence



(f) Efficiency: Varying n

Figure 3.3: Evaluating F-Score* (Simulation).

thus we have $\epsilon \neq 0$ as $\alpha = 0$.

Furthermore, to observe the distribution of ϵ, in Figure 3.3(b), we set $n = 50$ and $\alpha = 0.5$, and record the frequency of ϵ over 1000 trials. We observe that the error is centered around the mean 0.19%, and ranges from 0% to 0.31%, which is small and stable. To observe how ϵ changes for a larger n, in Figure 3.3(c) we fix $\alpha = 0.5$ and record ϵ by varying $n \in [0, 10^3]$. The curve shows that ϵ consistently decreases and it conforms to the error bound (i.e., $\epsilon = \mathcal{O}(n^{-1})$) as derived in Section 3.4.2. Especially when $n = 1000$, the average $\epsilon \leq 0.01\%$, which is fairly small.

To summarize, the proposed function F-score$^*(Q, R, \alpha)$ (Equation 5.4) is a good approximation of $\mathbb{E}[\, \textit{F-score}(T, R, \alpha) \,]$ in practice.

Optimal Result Vector Selection. In Figure 3.3(d)-(f), we study the result vector selection. Existing works [83,166,192] evaluate *F-score* by choosing the result of each task as the label with the highest probability. We first study the quality improvement by using our optimal returned result vector algorithm. Let R^* denote the optimal result vector, i.e., $R^* = \text{argmax}_R$ F-score$^*(Q, R, \alpha)$. Let $\widetilde{R} = [\tilde{r}_1, \tilde{r}_2, \ldots, \tilde{r}_n]$ denote the result vector chosen by the existing works [83,166,192], i.e., $\tilde{r}_i = 1$ if $Q_{i,1} \geq Q_{i,2}$ (or $Q_{i,1} \geq 0.5$) and $\tilde{r}_i = 2$ if otherwise. We set $n = 2000$ and vary $\alpha \in [0, 1]$, where for a fixed α, we randomly generate Q, and respectively compute R^* and $\widetilde{R}$ based on Q. Then we define the quality improvement Δ as follows:

$$\Delta = \mathbb{E}[\, \textit{F-score}(T, R^*, \alpha) \,] - \mathbb{E}[\, \textit{F-score}(T, \widetilde{R}, \alpha) \,], \tag{3.23}$$

and we record Δ in Figure 3.3(d). It can be observed that R^* improves the quality of $\widetilde{R}$ a lot. As α varies in $[0, 1]$, about 25% of the values of α result in an improvement of over 10%. We also observe that the curve is asymmetric bowl-shaped, especially when α is around 0.65, Δ becomes zero. Next we apply the approximated function F-score* to verify this interesting phenomenon in theory. For some unknown α', if $\widetilde{R}$ is equal to R^* (or $\widetilde{R} = R^*$),
(1) as $\widetilde{R}$ is constructed by comparing with the threshold 0.5, thus from Theorem 3.2 we know the threshold $\theta = \lambda^* \cdot \alpha' = 0.5$ and

(2) as $\lambda^* = \text{F-score}^*(Q, R^*, \alpha')$, and $R^* = \widetilde{R}$, we have

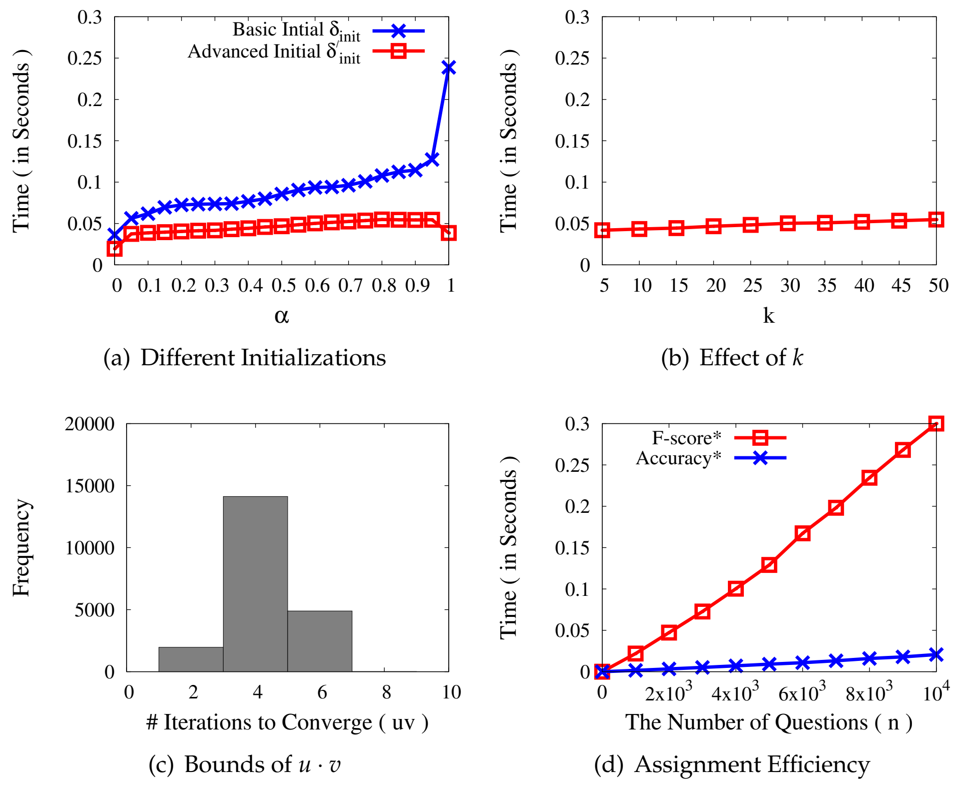
$$\lambda^* = \frac{\sum_{i=1}^{n} \mathbb{1}_{\{Q_{i,1} \geq 0.5\}} \cdot Q_{i,1}}{\alpha' \cdot \sum_{i=1}^{n} \mathbb{1}_{\{Q_{i,1} \geq 0.5\}} + (1-\alpha') \cdot \sum_{i=1}^{n} Q_{i,1}}.$$

Taking $\lambda^* \cdot \alpha' = 0.5$ inside, we can obtain $\sum_{i=1}^{n} Q_{i,1} \cdot \mathbb{1}_{\{Q_{i,1} \geq 0.5\}} = 0.5 \cdot \left[\sum_{i=1}^{n} \mathbb{1}_{\{Q_{i,1} \geq 0.5\}} + (\frac{1}{\alpha'} - 1) \cdot \sum_{i=1}^{n} Q_{i,1}\right]$. Note that as we randomly generate $Q_{i,1}$ ($1 \leq i \leq n$) for all tasks, it makes $Q_{i,1}$ ($1 \leq i \leq n$) uniformly distributed in $[0, 1]$. Thus if we take the expectation on both sides of the obtained formula, and apply the properties of uniform distribution, we can derive $0.75 \cdot \frac{n}{2} = 0.5 \cdot \left[\ \frac{n}{2} + (\frac{1}{\alpha'} - 1) \cdot 0.5 \cdot n\ \right]$, and then get $\alpha' = 0.667$, which verifies our observation (around 0.65).

As the time complexity of optimal vector selection algorithm (Section 3.4.2) is $\mathcal{O}(c \cdot n)$, we next evaluate the bound of c (#iterations to converge) in Figure 3.3(e). For a fixed $n = 2000$, we vary α from 0 to 1 with a step of 0.1, and for each α, we randomly generate Q and record c by running our algorithm. Figure 3.3(e) illustrates the frequency of c for all α. It can be seen that $c \leq 15$ in general (for different α). To evaluate the efficiency with a larger n, we vary $n \in [0, 10^4]$ and record the running time in Figure 3.3(f), which shows that the time linearly increases with n, and our algorithm finishes within 0.05s even when n is large ($n = 10^4$).

Evaluating Online Assignment's Efficiency

We evaluate the online assignment algorithm's efficiency in Figure 3.4(a)-(d). First we compare different ways of initialization in our algorithm. A basic way is to initialize $\delta_{init} = 0$. By considering that the optimal F-score* for Q^{X^*} may not be far away from the optimal F-score* for Q^c (as they are only different in k tasks' distributions), we propose another way of initialization: $\delta'_{init} = \max_R \text{ F-score}^*(Q^c, R, \alpha)$, which can be computed using the optimal result vector selection algorithm (Section 3.4.2). To compare their efficiency, in



(a) Different Initializations (b) Effect of k

(c) Bounds of $u \cdot v$ (d) Assignment Efficiency

Figure 3.4: Evaluating Efficiency of Assignments (Simulation).

Figure 3.4(a) we set $n = 2000$, $k = 20$, and vary $\alpha \in [0, 1]$. For a fixed α, we randomly generate Q^c and a confusion matrix, then Q^w can be computed (Equation 3.22). We run our algorithm with respective δ_{init} and δ'_{init}, and record the time in Figure 3.4(a). It can be seen that both initializations are efficient (≤ 0.3s), but there is a sudden increase as $\alpha \geq 0.95$ for δ_{init} (mainly due to the fact that big α focuses on *Precision*, that is, only confident tasks for L_1 will be returned, which leads to a big δ^*, and $\delta_{init} = 0$ is far away from δ^*). For δ'_{init}, it is steady with different α, as the initialization is computed by considering both Q^c and α. So we use δ'_{init} to initialize in later experiments.

We next evaluate the performance of different k in Figure 3.4(b). We set $n = 2000$ and vary $k \in [5, 50]$ to observe the assignment's efficiency, which shows that it is invariant with k. The main reason is that our algorithm iteratively updates δ, and the update solves a 0-1 FP problem via Dinkelbach framework, which is invariant of the size k. To evaluate the bound for $u \cdot v$, in Figure 3.4(c), we set $n = 2000$ and vary α from 0 to 1 with a step of 0.1. For each α, we generate Q^c and Q^w, and record the frequency of $u \cdot v$ in Figure 3.4(c). It showed that generally $u \cdot v \leq 10$ for different α.

We then evaluate the assignment's efficiency for Accuracy* and F-score* with a larger n. We set $k = 20$, $\alpha = 0.5$, and vary $n \in [0, 10^4]$ in Figure 3.4(d), which shows that both algorithms are efficient. For example, both of them finish within 0.3s when $n = 10^4$. As reported in [93], the number of HITs for a certain task is usually $\leq$ 5000, even for the top invested requesters. Therefore, our assignment algorithms can work well in a real crowdsourcing platform. Moreover, the assignment time both linearly increases for Accuracy* and F-score*, but with a larger factor in F-score*, as it has to deal with the converging complexity (i.e., $u \cdot v$).

To summarize, the assignment algorithms are both efficient for Accuracy* and F-score*, and they work well even for a large number of tasks (e.g., $n = 10^4$).

3.7.2 Experiments for Real Datasets

We first discuss the experimental setup (Section 6.2.1), then evaluate the properties of different worker models (Section 6.2.2), and finally compare with existing systems (Section 6.2.3).

Settings for Real Datasets

We generate five applications from real-world datasets (their ground truth is known for evaluation purposes). The result quality for the first two applications are evaluated in *Accuracy*, which is introduced as follows:

Films Poster (FS): *FS* uses the top 500 films (with posters) with the highest ratings from IMDB[6]. We generate 1,000 tasks, where each task asks the crowd to compare two different films and the coming worker should decide whether one film is published earlier ($<$) or later ($\geq$) than the other.

Sentiment Analysis (SA): *SA* uses the Twitter[7] dataset to label the sentiments of tweets w.r.t. different companies. It contains 5,152 hand-classified tweets. We select 1,000 tweets from them and each task asks the crowd to label the sentiment ("positive", "neutral" or "negative") of a tweet w.r.t. the related company.

The result quality for the other three applications is evaluated in *F-score* (with different α), which is introduced as follows:

Entity Resolution (ER): *ER* uses Product[8] dataset to evaluate whether the descriptions of two products refer to the same product or not. It contains 1,180,452 product pairs. For each pair (r_1,r_2), we compute the Jaccard similarity ($\frac{r_1 \cap r_2}{r_1 \cup r_2}$) and select 2,000 pairs with similarity ≥ 0.7 as our tasks. Each task contains a product pair and the coming worker should decide whether they are "equal" or "non-equal". It is evaluated using balanced *F-score* for the "equal" label ($\alpha = 0.5$).

Positive Sentiment Analysis (PSA): Based on the demand that a company manager may want to select positive sentiment comments for their products with high confidence (emphasizing *Precision*), we generate *PSA* by selecting 1,000 tweets related to Apple company from the Twitter[7] dataset. Each task is to ask the crowd about whether the sentiment of a tweet related to Apple is "positive" or "non-positive". As it emphasizes *Precision*, we set the evaluation metric as *F-score* for "positive" where $\alpha = 0.75$.

Negative Sentiment Analysis (NSA): From another perspective, a company manager may want to collect as many negative sentiment comments as possible for their products (emphasizing *Recall*), then *NSA* is generated by selecting 1,000 tweets related to Apple company from the Twitter[7] dataset. Each task is to ask

[6]http://www.imdb.com/
[7]http://www.sananalytics.com/lab/twitter-sentiment/
[8]http://dbs.uni-leipzig.de/file/Abt-Buy.zip

Table 3.2: The Statistics of Each Application.

Data	Labels	n	k	m	B	Evaluation Metric
FS	$L_1 =$"$<$", $L_2 =$"$\geq$"	1000	4	750	\$15	Accuracy
SA	$L_1 =$"positive", $L_2 =$"neutral", $L_3 =$"negative"	1000	4	750	\$15	Accuracy
ER	$L_1 =$"equal", $L_2 =$"non-equal"	2000	4	1500	\$30	F-score for L_1 ($\alpha = 0.5$)
PSA	$L_1 =$"positive", $L_2 =$"non-positive"	1000	4	750	\$15	F-score for L_1 ($\alpha = 0.75$)
NSA	$L_1 =$"negative", $L_2 =$"non-negative"	1000	4	750	\$15	F-score for L_1 ($\alpha = 0.25$)

the crowd about whether the sentiment of a tweet related to Apple is "negative" or "non-negative". It is evaluated using *F-score* for "negative" where $\alpha = 0.25$ (emphasizing *Recall*).

We compare QASCA with other five systems, i.e., Baseline, CDAS [127], AskIt! [27], MaxMargin and ExpLoss:

(1) Baseline: It randomly assigns k tasks to a coming worker.

(2) CDAS [127]: It adopts a quality-sensitive answering model to measure the confidence of tasks' current results, and terminates assigning the tasks which have already got confident results. It assigns k non-terminated tasks.

(3) AskIt! [27]: It uses an entropy-like method to define the uncertainty of each task, and assigns k most uncertain tasks.

(4) MaxMargin: It selects the next task with the highest expected marginal improvement, disregarding the characteristics of the worker, and it assigns k selected tasks.

(5) ExpLoss: It selects the next task by considering the expected loss, defined as $\min_j \sum_{j'=1}^{\ell} p_{j'} \cdot \mathbb{1}_{\{j \neq j'\}}$ $(1 \leq j \leq \ell)$, and it assigns k selected tasks.

As workers may vary in quality for different rounds, we need to use the same set of workers in the comparison of all six systems. To achieve this, when worker w comes, we use each system to assign $k = 4$ tasks, then $k \times 6 = 24$ tasks are batched into a HIT in a random order and the HIT will be assigned to worker w. Following this way, we can evaluate six systems in a "parallel" way with the same set of workers. We pay each worker $0.12 for doing a HIT, and assign each task to $z = 3$ workers on average, which are typical experimental settings in AMT [192,195]. The detailed settings are listed in Table 3.2. Take *FS* application as an example: there are $n = 1000$ generated tasks, and the "parallel" way assigns $m = \frac{n \times z}{k} = \frac{1000 \times 3}{4} = 750$ HITs in total for all six systems, where each system corresponds to $k = 4$ tasks in a HIT and each system takes $B = \frac{m \times \$0.12}{6} =$ $15 for *FS*. When HITs are finished, we compute the result quality w.r.t. the corresponding evaluation metric based on the tasks' ground truth and report their comparison results. All the HITs are published during 5:00 pm $\sim$ 10:00 pm (PDT) on weekdays, and all the crowdsourcing applications are finished within 24 hours. The number of workers participated in five applications *FS*, *SA*, *ER*, *PSA* and *NSA* are respectively 97, 101, 193, 104, and 101.

We implement QASCA in Python using the Django web framework, and deploy it on a 16GB memory Ubuntu server. We conduct experiments on a widely-used crowdsourcing platform: AMT. We use the "external-HIT" way provided by AMT which embeds the generated HTML pages (by our server) into its frame and workers directly interact with our server through the frame. When a worker comes, we can identify the worker from the individual worker-id provided by AMT. Then we can dynamically batch the selected tasks (from different systems) in a HIT for the coming worker.

Evaluating Worker Models

In this section, we especially study the selection of worker models in computing the distribution matrix. Recall that Section 3.6.2 reviews two typical

Table 3.3: Comparisons Between Worker Models.

	FS	*SA*	*ER*	*PSA*	*NSA*
CM	93.33%	87.09%	82.02%	91.73%	88.43%
WP	93.33%	78.64%	72.80%	92.61%	87.01%

worker models used in existing works: Worker Probability (WP) and Confusion Matrix (CM). Intuitively CM is better than WP as it is more complex. Even though what we observe from real datasets validates the intuition, we try to explain some interesting observations. We first collect the crowd's answers from the five published applications on AMT.

In Table 3.3, we compare WP and CM on five applications by leveraging the ground truth (t_i) of each task. Based on the collected answer set $D = \{D_1, D_2, \ldots, D_n\}$, for each worker w we compute the real WP and CM as follows

$$\begin{cases} \widetilde{m}^w = \frac{\sum_{i=1}^n \mathbb{1}_{\{(w,t_i)\in D_i\}}}{\sum_{i=1}^n \sum_{j=1}^{\ell} \mathbb{1}_{\{(w,j)\in D_i\}}}, \\ \widetilde{M}^w_{j,j'} = \frac{\sum_{i=1}^n \mathbb{1}_{\{t_i=j\}} \cdot \mathbb{1}_{\{(w,j')\in D_i\}}}{\sum_{i=1}^n (\mathbb{1}_{\{t_i=j\}} \cdot \sum_{z=1}^{\ell} \mathbb{1}_{\{(w,z)\in D_i\}})}. \end{cases} \tag{3.24}$$

We leverage the real priors (computed as $\widetilde{p}_j = \frac{\sum_{i=1}^n \mathbb{1}_{\{t_i=j\}}}{n}$ for $1 \le j \le \ell$) and two respective real worker models to derive the distribution matrix based on Equation 3.20, and then the optimal result vector R^* w.r.t. corresponding evaluation metric can be computed. Finally we evaluate the quality of R^* using the ground truths. To avoid overfitting, we randomly select 80% of the tasks answered by each worker to compute WP and CM, respectively. The random process is repeated over 1000 trails and we record the average quality of the results in Table 3.3. We observe that CM performs better than WP for *SA* and *ER* applications, while they perform approximately the same for the other three applications.

To explain this, we know that $\widetilde{m}^w$ performs equally with $\widetilde{M}^w$ only if the following requirements hold: (1) $\widetilde{M}^w_{j,j'} = \widetilde{M}^w_{j,j''}$ for $j' \neq j \neq j''$ (non-triangular elements in $\widetilde{M}^w$, which expresses the relations between pairwise labels); (2)

$\widetilde{M}^w_{j,j} = \widetilde{M}^w_{j',j'}$ (triangular elements in $\widetilde{M}^w$, which expresses the difficulty of individual label).

CM is better than WP on *SA* and *ER*, as they do not satisfy the above requirements: for *SA*, if a task's true label is "positive", a worker is more likely to answer "neutral" compared with "negative", i.e., $\widetilde{M}^w_{1,2} > \widetilde{M}^w_{1,3}$, which violates the first requirement, as there exists relations between pairwise labels; for *ER*, two product descriptions are "equal" only if all features (brand, color, etc.) are the same, while they are "non-equal" once a different feature is spotted, which implies that identifying a "non-equal" pair is easier than correctly identifying an "equal" pair, i.e., $\widetilde{M}^w_{1,1} < \widetilde{M}^w_{2,2}$ (*easier* means a higher probability), which violates the second requirement, as different labels have different difficulties.

To summarize, the above observations among labels in the two applications (*SA* and *ER*) can only be captured by CM other than WP. However, the labels in the other three applications (*FS*, *PSA* and *NSA*) do not have explicit relations or difficulties, which perform similarly for WP and CM. As CM performs at least as good as WP, we use the CM worker model in later experiments.

End-to-End Experiments

We compare QASCA with five other systems on five crowdsourcing applications in AMT.

Improvement of Optimal Result Selection. As we have validated the improvement in Section 3.7.1 on simulated datasets, here we further explore its benefit on real datasets. Recall the definition of R^* and $\widetilde{R}$ in Section 3.7.1.

As the ground truth vector T is known on real datasets, we define the real quality improvement w.r.t. different result vectors (R^* and $\widetilde{R}$) as

$$\widehat{\Delta} = \textit{F-score}(T, R^*, \alpha) - \textit{F-score}(T, \widetilde{R}, \alpha). \tag{3.25}$$

As HITs are completed, we compute R^* and $\widetilde{R}$ based on Q by time, and record $\widehat{\Delta}$

Table 3.4: The Average Quality Improvement ($\widehat{\Delta}$).

	Baseline	CDAS	AskIt!	MaxMargin	ExpLoss
ER ($\alpha = 0.5$)	2.59%	2.69%	4.56%	5.49%	4.32%
PSA ($\alpha = 0.75$)	4.14%	2.96%	1.26%	2.08%	1.66%
NSA ($\alpha = 0.25$)	14.12%	10.45%	12.44%	14.26%	9.98%

at each time-stamp. Finally after all HITs are completed, we report the average $\widehat{\Delta}$ for three applications (i.e., *ER*, *PSA* and *NSA*)[9] with the evaluation metric *F-score* in Table 3.4. Recall that the three applications *ER*, *PSA* and *NSA* have different α (i.e., 0.5, 0.75 and 0.25), and their corresponding Δ in simulated experiments (Figure 3.3(d)) are around 2%, 1% and 9%, respectively. In Table 3.4, we observe that the average $\widehat{\Delta}$ are larger than zero on all applications, which means that all systems can benefit from choosing the optimal result vector R^*. We also observe that *NSA* ($\alpha = 0.25$) has a bigger improvement compared with *ER* and *PSA*, which conforms to the observation in simulated experiments. Note that the main reason that $\widehat{\Delta}$ may have some differences from Δ is that in our simulated experiments, the $Q_{i,1}$ for $1 \leq i \leq n$ is uniformly distributed. While in reality, as a task q_i gets more answers, the computed distribution Q_i may become more confident (either $Q_{i,1}$ is close to 1 or 0). To summarize, for *F-score*, all systems can benefit from choosing the optimal result vector R^* rather than returning a label with the highest probability ($\widetilde{R}$).

System Comparison Results. We next show the main results, i.e., the real result quality compared with other systems on all applications in Figure 3.5(a)-(e). We collect the completed HITs by time and calculate the corresponding result quality based on the derived results of different systems. Since we have shown that the quality improves a lot by selecting the optimal result vector R^* for F-score* both in simulated datasets (Figure 3.3(d)) and real datasets (Table 3.4). To make a fair comparison, we apply this optimization (i.e., selecting R^*) to all systems

[9]In Theorem 3.1 we have proved that $R^* = \widetilde{R}$ for Accuracy*, so we did not include the applications with the evaluation metric *Accuracy* (i.e., *FS* and *SA*).

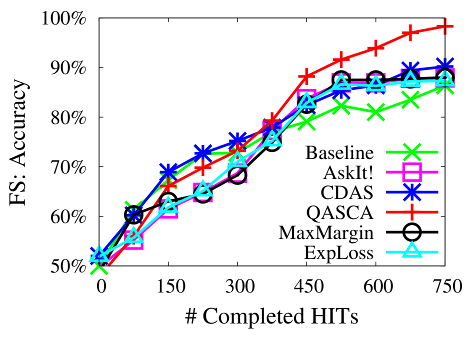
Table 3.5: Overall Result Quality (All HITs completed).

Dataset	*FS*	*SA*	*ER*	*PSA*	*NSA*
Baseline	86.40%	72.20%	71.07%	80.85%	74.73%
CDAS	90.20%	73.80%	73.75%	81.58%	75.68%
AskIt!	87.90%	72.20%	71.78%	81.95%	73.16%
QASCA	**98.30%**	**84.60%**	**85.96%**	**95.70%**	**86.65%**
MaxMargin	88.00%	73.30%	72.02%	83.92%	75.96%
ExpLoss	87.30%	72.90%	71.36%	82.43%	73.38%

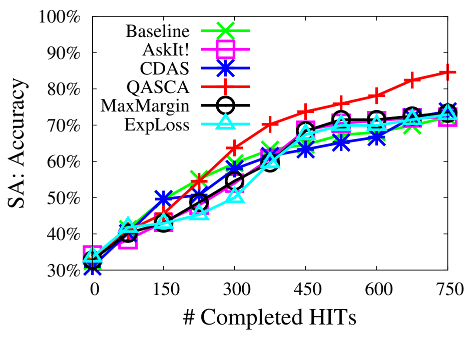
when the evaluation metric is *F-score*.

From Figure 3.5(a)-(e), we can observe that in the beginning, when the number of completed HITs is small, all systems have similar performance, as they all do not know much information about tasks or workers. However, QASCA dominates other systems as time goes by. This is because QASCA assigns the best k tasks for each coming worker to maximize the evaluation metric value (Accuracy* or F-score*), while other systems do not explicitly consider the impact of evaluation metrics on result quality in their task assignment process. For a clear comparison, Table 3.5 shows the final result quality value when all HITs are completed, and QASCA leads other systems over 8% for all applications. To be specific, QASCA leads the second best system by 8.1%, 10.8%, 12.21%, 11.78% and 10.69% on five real-world datasets (*FS*, *SA*, *ER*, *PSA* and *NSA*), respectively. We can also observe that MaxMargin outperforms ExpLoss, as for the inherently ambiguous tasks, they will have high expected loss (ExpLoss will continuously assign them); while the marginal benefit of assigning these tasks will be much lower as more answers are collected (MaxMargin can save the assignments for more beneficial tasks).

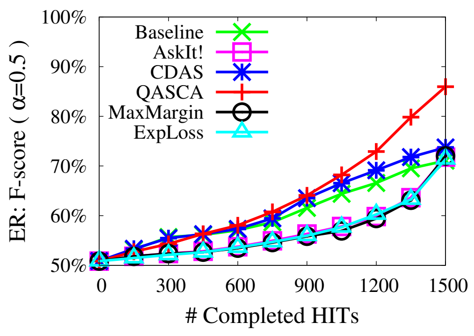
We compare the efficiency of different systems in Figure 3.6(a). To make a fair comparison, for an application in each system, we record the worst case assignment time during the assignment process for all HITs. It is shown that all systems are efficient, and the worst case assignment of all systems can be fin-



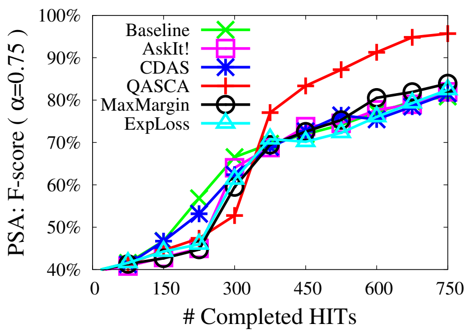
(a) *FS*: *Accuracy*



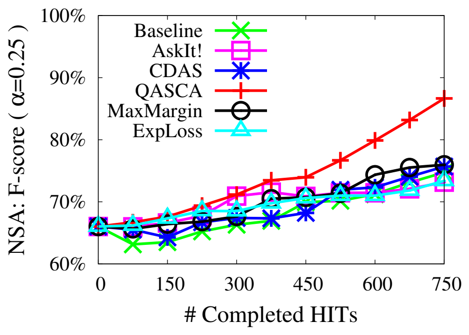
(b) *SA*: *Accuracy*



(c) *ER*: *F-score*



(d) *PSA*: *F-score*



(e) *NSA*: *F-score*

Figure 3.5: End-to-End System Comparisons.

ished within 0.06s, which is fairly acceptable in real applications. Even though QASCA is less efficient than other systems due to its complexity in assignments, it can significantly improve the result quality (over 8%). The reason why *ER* runs slower is that it contains 2000 tasks while other applications contain 1000 tasks.

Estimation of Worker Quality. We next examine how the estimated quality of workers is different from the real quality of workers[10] by ranging the percentage of completed HITs. Let us denote the real CM as $\widetilde{M}^w$ and the estimated CM as M^w. For a worker w, we further define the *estimation deviation* of worker quality as the absolute difference of estimated quality and real quality, i.e., $\frac{1}{\ell \times \ell} \cdot \sum_{j=1}^{\ell} \sum_{j'=1}^{\ell} |M^w_{j,j'} - \widetilde{M}^w_{j,j'}|$. Then we calculate the *mean estimation deviation* by averaging the calculated estimation deviation among all workers. The smaller the mean estimation deviation is, the closer the estimated worker quality (M^w) is to the real one ($\widetilde{M}^w$). We report the mean estimation deviation by ranging the percentage of completed HITs for all datasets in Figure 3.6(b). It shows that as more HITs are completed, the estimated worker quality gets closer to the real one, which may explain why QASCA performs much better compared with other systems as time goes by. That is, as more HITs are completed, the worker's quality is more accurately estimated, then QASCA takes the desired quality metric into consideration and can better leverage the estimated worker's quality to judge how the worker's answers might affect the quality metric if tasks are assigned. Then it selects the assignment that could maximize the quality metric.

3.8 Related Work

Since we have reviewed most of the related works of crowdsourcing in Chapter 2, this section only highlights the part related to evaluation metrics.

[10]The real quality of each worker is calculated by leveraging the ground truth T and the answer set D. We can follow Section 3.7.2 (Equation 3.24) to compute the real CM.

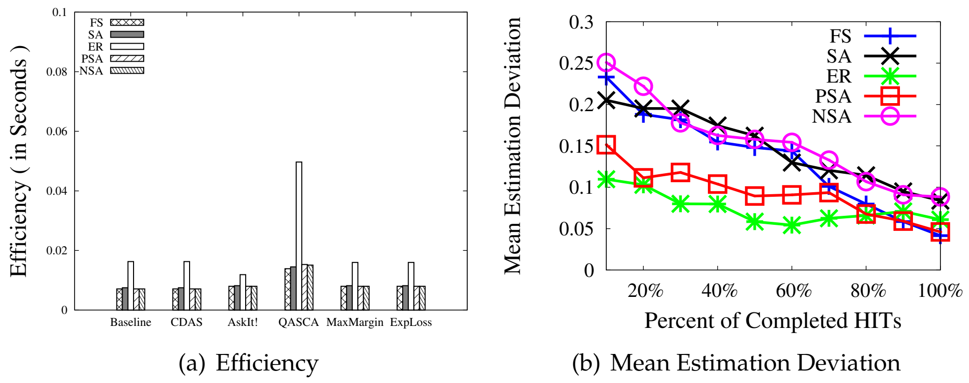


Figure 3.6: Efficiency and Mean Estimation Deviation.

Accuracy [44,91,92,127,161] and *F-score* [96,132,155,192,195,199,215] are two widely used evaluation metrics for crowdsourcing applications, where *Accuracy* evaluates the overall quality and *F-score* focuses on the quality of a specific label. Note that there are other metrics defined based on different purposes. In entity resolution, there are also several cluster-based metrics, such as *K*-measure [111], *GMD* measure [139] and Rand-index [176]. For strings, similarity-based metrics including Jaccard, Dice, Cosine and Edit Distances are defined and used [50, 185,194]. We focus on studying the task-based metrics and propose solutions to estimate the result quality based on distribution matrices.

3.9 Chapter Summary

In this chapter, we have studied the task assignment problem in the task-based setting. To solve the problem, we proposed a novel assignment framework by incorporating evaluation metrics into assignment strategies. We generalized the widely used existing evaluation metrics (*Accuracy* and *F-score*) to be able to quantify the result quality w.r.t a distribution matrix. We designed optimal result vector selection algorithms and two respective efficient online assignment algorithms for *Accuracy* and *F-score*. We built the QASCA system by integrating our novel assignment framework with AMT, and evaluated our system on five real applications. The experimental results showed that QASCA achieved much better result quality than existing approaches.

In next chapter, we will focus on studying the task assignment problem in another setting: worker-based setting, i.e., we want to make a wise decision of selecting workers, such that the tasks in hand can be completed successfully and economically by the selected workers.

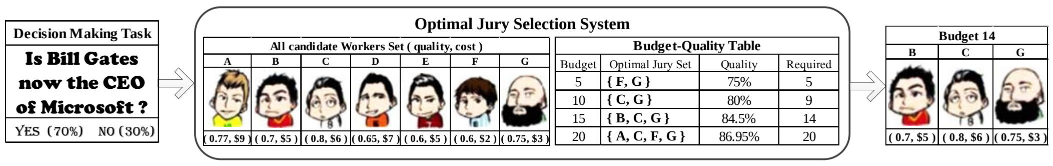
Chapter 4

Optimal Jury Selection Problem

4.1 Introduction

Due to advances in crowdsourcing technologies, computationally challenging tasks (e.g., sentiment analysis, entity resolution, document translation, etc.) can now be easily performed by human workers on the Internet. As reported by the Amazon Mechanical Turk in August 2012, over 500,000 workers from 190 countries worked on human intelligence tasks (HITs). The large number of workers and HITs have motivated researchers to develop solutions to streamline the crowdsourcing process [33, 69, 151].

In general, crowdsourcing a set of tasks involves the following steps: (1) distributing tasks to workers; (2) collecting the workers' answers; (3) deciding final result; and (4) rewarding the workers. An important task assignment problem in the task-base setting is: *how should workers be chosen, so that the tasks in hand can be completed with high quality, while minimizing the monetary budget available?* A related question, called the *Jury Selection Problem* (or JSP), has been recently proposed by Cao et al. [33]. Similar to the concept from law courts, a *jury*, or *jury set* denotes a subset of workers chosen from the available worker pool. Given a monetary budget and a task, the goal of JSP is to find the jury with the highest



Budget	Optimal Jury Set	Quality	Required
5	{ F, G }	75%	5
10	{ C, G }	80%	9
15	{ B, C, G }	84.5%	14
20	{ A, C, F, G }	86.95%	20

Figure 4.1: Optimal Jury Selection System.

expected performance within the budget constraint. The kind of tasks studied in [33] is called the *decision-making task*: a question that requires an answer of either *yes* or *no* (e.g., "Is Bill Gates still the CEO of Microsoft now?") and has a definitive ground truth. Decision-making tasks [33] are commonly used in crowdsourcing systems because of their conceptual simplicity. The authors of [33] were the first to propose a system to address JSP for this kind of tasks.

In this chapter, we go beyond [33] and perform a comprehensive investigation of this problem. Particularly, we ask the following questions: (1) Is the solution in [33] optimal? (2) If not, what is an optimal solution for JSP? To understand these issues, let us first illustrate how [33] solves JSP.

Figure 4.1 shows a decision-making task, to be answered by some of the seven workers labeled from A to G where each worker is associated with a *quality* and a *cost*. The *quality* ranges from 0 to 1, indicating the probability that the worker correctly answers a question. This probability can be estimated by using her background information (e.g., her performance in other tasks) [33,127,183]. The *cost* is the amount of monetary reward the worker can get upon finishing a task. In this example, A has a quality of 0.77 and a cost of 9 units. For a jury, the *jury cost* is defined as the sum of workers' costs in the jury and the *jury quality* (or JQ) is defined as the probability that the result returned by aggregating the jury answers is correct. Given a budget of B units, a feasible jury is a jury whose *jury cost* does not exceed B. For example, if $B = \$20$, then $\{B, E, F\}$ is a feasible jury, since its *jury cost*, or $\$5 + \$5 + \$2 = \$12$, is not larger than \$20.

To solve JSP, a naive solution is to compute the JQ for every feasible jury, and return the one with the highest JQ. In [33], it studies how to compute JQ for a jury where the jury's returned result is decided by *Majority Voting* (MV). In short, MV returns the result as the one corresponding to the most workers. In the following, we consider each worker's answer as a "vote" for either "yes" or "no". Let us consider $\{B, E, F\}$ again, the probability that these workers gives a correct result according to MV is $0.7 \cdot 0.6 \cdot 0.6 + 0.7 \cdot 0.6 \cdot (1 - 0.6) + 0.7 \cdot (1 - 0.6) \cdot 0.6 + (1 - 0.7) \cdot 0.6 \cdot 0.6 = 69.6\%$. Moreover, since $\{A, C, G\}$ yields the highest JQ among all the feasible juries, it is considered to be the best solution by [33].

As illustrated above, MV is used to solve JSP in [33]. In addition to MV, researchers have proposed a variety of *voting strategies*, such as Bayesian Voting (BV) [127], Randomized Majority Voting [110], and Random Ballot Voting [9]. Like MV, these voting strategies decide the final result of a decision-making task based on the workers' votes. For example, BV computes the posterior probability of answers according to Bayes' Theorem [23], based on the workers' votes, and returns the answer having the largest posterior probability.

In this chapter, we investigate an interesting problem: is it possible to find the optimal voting strategy for JSP among all voting strategies? One simple answer to this question is to consider all voting strategies. However, as listed in Table 4.2, the number of existing strategies is very large. Moreover, multiple new strategies may emerge in the future. We address this question by first studying the criteria of a strategy that produce an optimal solution for JSP (i.e., given a jury, the JQ of the strategy is the highest among all the possible voting strategies). This is done by observing that voting strategies can be classified into two major categories: *deterministic* and *randomized*. A deterministic strategy aggregates workers' answers without any degree of randomness; MV is a typical example of this class. For a randomized strategy, each answer is returned with some probability. Using this classification, we present the criteria required for a voting strategy that leads to the optimal solution for JSP. We discover that BV satisfies the requirements of an optimal strategy. In other words, BV is the optimal voting strategy with respect to JQ, and will consistently produce better quality juries than the other strategies.

How to solve JSP with BV then? A straightforward solution is to enumerate all feasible juries, and find the one with the largest value of JQ. However, this approach suffers from two major problems:

1. Computing the JQ of a jury for BV requires enumerating an exponentially large number of workers' answers. In fact, we show that this problem is NP-hard;
2. The number of feasible juries is exponentially large.

To solve Problem 1, we develop a polynomial-time approximation algorithm, which enables a large number of candidate answers to be pruned, without a significant loss of accuracy. We further develop a theoretical error bound of this algorithm. Particularly, our approximate JQ computation algorithm is proved to yield an error of not more than 1%. To tackle Problem 2, we leverage a successful heuristic, the simulated annealing heuristic, by designing local neighborhood search functions. To evaluate our solutions, we have performed extensive evaluation on real and synthetic crowdsourced data. Our experimental results show that our algorithms effectively and efficiently solve JSP. The quality of our solution is also consistently better than that of [33].

We also study how to allow the provider of the tasks to place her confidence information (called *prior*) on the answers of the task. She may associate a "belief score" on the answers to the tasks, before the crowdsourcing process starts. For instance, in Figure 4.1, if she is more confident that Bill Gates is still the CEO of Microsoft, she can assign 70% to *yes*, and 30% to *no*. Intuitively, we prove that under BV, the effect of prior is just the same as regarding the task provider as another worker, having the same quality values as the prior.

Figure 4.1 illustrates our crowdsourcing system, which we called the "Optimal Jury Selection System". In this system, the task provider published a decision-making task. Then, based on the the workers' information (i.e., their individual quality and cost), a "budget-quality table" is generated. In this table, each row contains a budget, the computed optimal jury, its estimated jury quality and the required budget for the jury. Based on this table, the task provider can conveniently decide the best budget-quality combination. For example, she may deem that increasing the budget from 15 units to 20 units is not worthwhile, since the quality increases only by around 2.5%. In this example, the task provider selects the jury set $\{B, C, G\}$ that is the best under a budget of 15 units. This chosen jury set would cost her only 14 units.

Recall that [33] focuses on addressing JSP under MV on decision-making tasks and we address the optimality of JSP on decision-making tasks by considering all voting strategies, where each worker's quality is modeled as a single parameter. In reality, multiple choice tasks [127, 160, 200] are also commonly used in crowdsourcing and several works [19, 160, 182] model each worker as a confusion matrix rather than a single quality score. We also briefly discuss here the optimality of JSP for other task types and worker models, and how our solutions can be extended to these other variants.

The rest of this chapter is arranged as follows. We describe the data model and the problem definition in Section 4.2. In Section 4.3, we examine the requirements of an optimal voting strategy for JSP, and show that BV satisfies these criteria. We present an efficient algorithm to compute JQ of a jury set in Section 4.4 and develop fast solutions to solve JSP in Section 4.5. In Section 4.6, we present our experimental results. We discuss how our solutions can be extended for other task types and worker models in Section 4.7. In Section 4.8, we review the related works and Section 4.9 concludes the chapter.

4.2 Data Model & Problem Definition

We now describe our data model in Section 4.2.1 and define the jury selection problem in Section 4.2.2.

4.2.1 Data Model

In this chapter, we focus on the *decision-making tasks* where each task has two possible answers (either *yes* or *no*). We use 1 and 0 to denote *yes* and *no*, respectively. We assume that each task has a latent true answer (or ground truth) $\mathbf{t} \in \{0, 1\}$, which is unknown in advance. The task provider usually assigns a *prior* on the task, which describes her prior knowledge in the probability distribution of the task's true answer. We denote the prior by α where $\Pr(\mathbf{t} = 0) = \alpha$, and $\Pr(\mathbf{t} = 1) = 1 - \alpha$. If the task provider has no prior knowledge for the task, then we assume $\alpha = 0.5$.

A *jury* (or *jury set*), denoted by J, is a collection of n workers drawn from a set of N candidate workers $W = \{j_1, j_2, \dots, j_N\}$, i.e., $J \subseteq W$, $|J| = n$. Without loss of generality, let $J = \{j_1, j_2, \dots, j_n\}$. In order to infer the ground truth ($\mathbf{t}$), we leverage the collective intelligence of a jury, i.e, we ask each worker to give a vote for the task. We use V, a *voting*, to denote the set of votes (answers) given by a jury J, and so $V = \{v_1, v_2, \dots, v_n\}$ where $v_i \in \{0, 1\}$ is the vote given by j_i. We assume the independence of each worker's vote, an assumption also used in [33,92,127,160].

We follow the worker model in previous works [33,127,211], where each worker j_i is associated with a quality $q_i \in [0, 1]$ and a cost c_i. The quality q_i indicates the probability that the worker conducts a correct vote, i.e., $q_i = \Pr(v_i = \mathbf{t})$, and the cost c_i represents the money (or incentive) required for j_i to give a vote. A few works [33,127,183] have recently addressed how to derive the quality and the cost of a worker by leveraging the backgrounds and answering history of individuals. Thus, similar to [33], we assume that they are known in advance. Also, we have dealt with the simple case that each worker has the same cost requirement in Section 4.5.

We remark that the optimality of JSP and our solutions can be extended to address other task types and worker models used in [19,127,160,160,182,200]. We will briefly discuss these extensions in Section 4.7.

4.2.2 Problem Definition

Let B be the budget of a task provider, i.e., a maximum of B cost units can be given to a jury to collect their votes. Our goal is to solve the *Jury Selection Problem* (denoted by JSP) which selects a jury J under the budget constraint ($\sum_{j_i \in J} c_i \leq B$) such that the jury's collective intelligence is maximized.

The collective intelligence of a jury is closely related to the *Voting Strategy*, denoted by S, which estimates the true answer of the task based on the prior, the jury and their votes. We say the estimated true answer is the *result* of the voting strategy. A detailed discussion about the voting strategy is given in Section 4.3.1.

In order to quantify the jury's collective intelligence, we define the *Jury Quality* (or *JQ* in short) which essentially measures the probability that the result of the voting strategy is correct. The score of JQ is given by function $JQ(J, S, \alpha)$. We will give a precise definition for JQ in Section 4.3.2.

Let Θ denote the set of all voting strategies and $\mathcal{C}$ denote the set of all feasible juries (i.e., $\mathcal{C} = \{J \mid J \subseteq W \wedge \sum_{j_i \in J} c_i \leq B\}$). The aim of JSP is to select the optimal jury J^* such that

$$\textbf{given} \qquad \alpha \text{ and } q_i, c_i \text{ (for } i = 1, 2, \ldots, N) \tag{4.1}$$

$$J^* = \underset{J \in \mathcal{C}}{\operatorname{argmax}} \max_{S \in \Theta} JQ(J, S, \alpha) \tag{4.2}$$

Note that existing work [33] only focuses on majority voting strategy (MV)

Table 4.1: Notations Used in Chapter 4.

Symbol	Description
$\mathbf{t}$	the ground truth for a task, and $\mathbf{t} \in \{0,1\}$
α	prior given by the task provider, and $\alpha = \Pr(\mathbf{t}=0)$
W	a set of all candidate workers $W = \{j_1, j_2, \ldots, j_N\}$
J	a jury, $J \subseteq W$ and $\lvert J\rvert = n$, $J = \{j_1, j_2, \ldots, j_n\}$
V	a voting given by J, and $V = \{v_1, v_2, \ldots, v_n\}$
q_i	the quality of worker j_i and $q_i \in [0,1]$
c_i	the cost of worker j_i
B	the budget provided by the task provider
Θ	a set containing all voting strategies
$\mathcal{C}$	the set of all possible juries within budget constraint

and solves $\text{argmax}_{J\in\mathcal{C}}\, JQ(J, \text{MV}, 0.5)$, which, as we shall prove later, is sub-optimal for JSP.

In the rest of the chapter, we first discuss how to derive the optimal voting strategy S^* such that $JQ(J, S^*, \alpha) = \max_{S\in\Theta} JQ(J, S, \alpha)$ (Section 4.3). We then talk about the computation of $JQ(J, S^*, \alpha)$ (Section 4.4) and finally address of problem of finding J^* (Section 4.5).

Table 4.1 summarizes the symbols used in this chapter.

4.3 Optimal Voting Strategy

In this section, we present a detailed description for the voting strategy in Section 4.3.1. We then formally define JQ in Section 4.3.2. Finally, we give an optimal voting strategy with respect to JQ in Section 4.3.3.

4.3.1 Voting Strategies

As mentioned, a *voting strategy* S gives an estimation of the true answer $\mathbf{t}$ based on the prior α, the jury J and their votes V. Thus, we model a voting strategy as a function $S(V, J, \alpha)$, whose result is an estimation of $\mathbf{t}$. Based on whether the result is given with degree of randomness, we can classify the voting strategies into two categories: *deterministic voting strategy* and *randomized voting strategy*.

Definition 4.1. *A deterministic voting strategy $S(V, J, \alpha)$ returns the result as 0 or 1 without any degree of randomness.*

Definition 4.2. *A randomized voting strategy $S(V, J, \alpha)$ returns the result as 0 with probability p and 1 with probability $1 - p$.*

Example 8. *The majority voting strategy (or MV) is a typical deterministic voting strategy, and it gives result as 0 if more than half of workers vote for 0 (i.e., $\sum_{i=1}^{n}(1 - v_i) \geq \frac{n+1}{2}$); otherwise, the result is 1.*

Its randomized counterpart is called randomized majority voting strategy (or RMV), which returns the result with probability proportional to the number of votes. That is, RMV returns 0 with probability $p = \frac{1}{n}\sum_{i=1}^{n}(1 - v_i)$, and 1 with probability $1 - p$.

Note that randomized strategies are often introduced to improve the error bound for worst-case analysis [123]. And thus, they are widely used when the worst-case performance is the main concern. While in practice, in most of the cases, deterministic strategies are used due to less randomness.

Table 4.2: Classification of Voting Strategies.

Deterministic Voting Strategies	Randomized Voting Strategies
Majority Voting (MV) [33]	Randomized Majority Voting (RMV) [110]
Half Voting [141]	Random Ballot Voting [9]
Bayesian Voting [127]	Triadic Consensus [20]
Weighted MV [123]	Randomized Weighted MV [123]
...	...

Table 4.2 shows a few voting strategies, which are introduced in previous works, and their corresponding category.

4.3.2 Jury Quality

In order to measure the goodness of a voting strategy S for a jury J, we introduce a metric called *Jury Quality* (or *JQ* in short). We model JQ by a function $JQ(J,S,\alpha)$ which gives the quality score as the probability of drawing a correct result under the voting strategy, i.e.,

$$JQ(J,S,\alpha) = \Pr(S(\mathbf{V},J,\alpha) = \mathbf{t}) \tag{4.3}$$

where $\mathbf{V} \in \{0,1\}^n$ and $\mathbf{t} \in \{0,1\}$ are two random variables corresponding to the unknown jury's voting, and the task's latent true answer. For notational convenience, we omit J and α in S when their values are understood and simply write $S(\mathbf{V})$ instead of $S(\mathbf{V},J,\alpha)$.

Let $\mathbb{1}_{\{st\}}$ be the indicator function, which returns 1 if the statement st is true, and 0 otherwise. Let Ω be the domain of $\mathbf{V}$, i.e, $\Omega = \{0,1\}^n$. $JQ(J,S,\alpha)$ can be rewritten as follows.

$$\begin{aligned} JQ(J,S,\alpha) &= 1\cdot\Pr(S(\mathbf{V})=\mathbf{t}) + 0\cdot\Pr(S(\mathbf{V})\neq\mathbf{t}) \\ &= \mathbb{E}[\mathbb{1}_{\{S(\mathbf{V})=\mathbf{t}\}}] \\ &= \sum_{t\in\{0,1\}}\sum_{V\in\Omega}\Pr(\mathbf{V}=V,\mathbf{t}=t)\cdot\mathbb{E}[\mathbb{1}_{\{S(V)=t\}}] \end{aligned}$$

We now give a precise definition for JQ as below.

Definition 4.3 (Jury Quality)**.** *Given a jury J and the prior α, the Jury Quality (or JQ) for a voting strategy S, denoted by $JQ(J,S,\alpha)$, is defined as*

$$\begin{aligned} &\alpha\cdot\textstyle\sum_{V\in\Omega}\Pr(\mathbf{V}=V\mid\mathbf{t}=0)\cdot\mathbb{E}[\mathbb{1}_{\{S(V)=0\}}] \\ +\ &(1-\alpha)\cdot\textstyle\sum_{V\in\Omega}\Pr(\mathbf{V}=V\mid\mathbf{t}=1)\cdot\mathbb{E}[\mathbb{1}_{\{S(V)=1\}}]. \end{aligned} \tag{4.4}$$

For notational convenience, we write $\Pr(V|\mathbf{t}=0)$ instead of $\Pr(\mathbf{V}=V|\mathbf{t}=0)$, and $\Pr(V|\mathbf{t}=1)$ instead of $\Pr(\mathbf{V}=V|\mathbf{t}=1)$. Next, we give two marks in computing JQ.

1. Since workers give votes independently, we have

$$\begin{aligned}\Pr(V \mid \mathbf{t}=0) &= \textstyle\prod_{i=1}^{n} q_i^{(1-v_i)} \cdot (1-q_i)^{v_i} \\ \Pr(V \mid \mathbf{t}=1) &= \textstyle\prod_{i=1}^{n} q_i^{v_i} \cdot (1-q_i)^{(1-v_i)}\end{aligned}$$

2. $\mathbb{E}[\mathbb{1}_{\{S(V)=0\}}]$ and $\mathbb{E}[\mathbb{1}_{\{S(V)=1\}}]$ are either 0 or 1 if S is a deterministic voting strategy; or value of p and $1-p$ if S is a randomized voting strategy (refer to Definition 4.2).

We next give an example to illustrate the computation of JQ.

Example 9. *Suppose $\alpha = 0.5$ and there are 3 workers in J with workers' qualities as $0.9, 0.6, 0.6$ respectively. To compute JQ for MV, we enumerate all possible combinations of V $(\in \{0,1\}^3)$ and t $(\in \{0,1\})$, and show the results in Figure 4.2. The 3rd column in each table represents the probability that a specific combination (V and t) exists. The 4th column shows the result of MV for each V. The symbol $\surd$ indicates whether MV's result is correct or not (according to the value of t). And thus, $JQ(J, MV, \alpha)$ equals to the summation of probabilities where symbol $\surd$ occurs. Take $V = \{1,0,0\}$ and $t = 0$ as an example. First, $\Pr(\mathbf{V} = V, \mathbf{t} = 0) = 0.018$. Since $\sum_{i=1}^{3}(1-v_i) = 2 \geq \frac{n+1}{2} = 2$, we have $MV(V) = 0 = t$ Thus, the probability 0.018 is added to $JQ(J, MV, \alpha)$. Similarly, for $V = \{1,0,0\}$ and $t = 1$, as $MV(V) = 0 \neq t$, then $\Pr(\mathbf{V} = V, \mathbf{t} = 1) = 0.072$ will not be added to $JQ(J, MV, \alpha)$. Considering all V's and t's, the final $JQ(J, MV, \alpha) = 79.2\%$.*

4.3.3 Optimal Voting Strategy

In the last two sections, we present a few voting strategies and define Jury Quality to quantify the goodness of a voting strategy. Thus an interesting question is: does an optimal voting strategy S^* with respect to JQ exist? That is,

No.	*V*	P(t = 0)*P(**V** = *V* $\mid$ **t** = 0)	MV: [compare] (result) [√/×]	BV: [compare] (result) [√/×]
1	{ 0,0,0 }	0.5*0.9*0.6*0.6=0.162	[3 ≥ 2] (0) [√]	[0.162 ≥ 0.008] (0) [√]
2	{ 0,0,1 }	0.5*0.9*0.6*0.4=0.108	[2 ≥ 2] (0) [√]	[0.108 ≥ 0.012] (0) [√]
3	{ 0,1,0 }	0.5*0.9*0.4*0.6=0.108	[2 ≥ 2] (0) [√]	[0.108 ≥ 0.012] (0) [√]
4	{ 0,1,1 }	0.5*0.9*0.4*0.4=0.072	[1 < 2] (1) [×]	[0.072 ≥ 0.018] (0) [√]
5	{ 1,0,0 }	0.5*0.1*0.6*0.6=0.018	[2 ≥ 2] (0) [√]	[0.018 < 0.072] (1) [×]
6	{ 1,0,1 }	0.5*0.1*0.6*0.4=0.012	[1 < 2] (1) [×]	[0.012 < 0.018] (1) [×]
7	{ 1,1,0 }	0.5*0.1*0.4*0.6=0.012	[1 < 2] (1) [×]	[0.012 < 0.018] (1) [×]
8	{ 1,1,1 }	0.5*0.1*0.4*0.4=0.008	[0 < 2] (1) [×]	[0.008 < 0.162] (1) [×]

(a) Enumeration of all $2^3 = 8$ possible votings in Ω ($\mathbf{t} = 0$)

No.	*V*	P(t = 1)*P(**V** = *V* $\mid$ **t** = 1)	MV: [compare] (result) [√/×]	BV: [compare] (result) [√/×]
1	{ 0,0,0 }	0.5*0.1*0.4*0.4=0.008	[3 ≥ 2] (0) [×]	[0.162 ≥ 0.008] (0) [×]
2	{ 0,0,1 }	0.5*0.1*0.4*0.6=0.012	[2 ≥ 2] (0) [×]	[0.108 ≥ 0.012] (0) [×]
3	{ 0,1,0 }	0.5*0.1*0.6*0.4=0.012	[2 ≥ 2] (0) [×]	[0.108 ≥ 0.012] (0) [×]
4	{ 0,1,1 }	0.5*0.1*0.6*0.6=0.018	[1 < 2] (1) [√]	[0.072 ≥ 0.018] (0) [×]
5	{ 1,0,0 }	0.5*0.9*0.4*0.4=0.072	[2 ≥ 2] (0) [×]	[0.018 < 0.072] (1) [√]
6	{ 1,0,1 }	0.5*0.9*0.4*0.6=0.108	[1 < 2] (1) [√]	[0.012 < 0.018] (1) [√]
7	{ 1,1,0 }	0.5*0.9*0.6*0.4=0.108	[1 < 2] (1) [√]	[0.012 < 0.018] (1) [√]
8	{ 1,1,1 }	0.5*0.9*0.6*0.6=0.162	[0 < 2] (1) [√]	[0.008 < 0.162] (1) [√]

(b) Enumeration of all $2^3 = 8$ possible votings in Ω ($\mathbf{t} = 1$)

Figure 4.2: An Example of JQ computation for MV and BV.

given any J and α, $JQ(J, S^*, \alpha) = \max_{S \in \Theta} JQ(J, S, \alpha)$. Note that if S^* exists, we can then solve JSP without enumerating all voting strategies in Θ (refer to Definition 4.2).

To answer this question, let us reconsider Definition 4.3 and Equation 4.4. Let $h(V) = \mathbb{E}[\mathbb{1}_{\{S(V)=0\}}]$. We have (i) $h(V) \in [0,1]$; and (ii) $\mathbb{E}[\mathbb{1}_{\{S(V)=1\}}] = 1 - h(V)$. Also, let $P_0(V) = \Pr(\mathbf{V} = V, \mathbf{t} = 0)$, and $P_1(V) = \Pr(\mathbf{V} = V, \mathbf{t} = 1)$. Hence, $JQ(J, S, \alpha)$ can be rewritten as

$$
\begin{aligned}
& \sum_{V \in \Omega} [\, P_0(V) \cdot h(V) + P_1(V) \cdot (1 - h(V)) \,] \\
= & \sum_{V \in \Omega} [\, h(V) \cdot (P_0(V) - P_1(V)) + P_1(V) \,]
\end{aligned}
$$

This gives us a hint to maximize $JQ(J, S, \alpha)$ and find the optimal voting strategy S^*. Let $h^*(V) = \mathbb{E}[\mathbb{1}_{\{S^*(V)=0\}}]$. It's observed that $P_1(V)$ is constant for a given V and $h(V) \in [0,1]$ for all S's (no matter it's a deterministic one or a randomized one). Thus, to optimize $JQ(J, S, \alpha)$, it's required that

1. if $P_0(V) - P_1(V) < 0$, $h^*(V) = 0$, and so, $S^*(V) = 1$;
2. if $P_0(V) - P_1(V) \geq 0$, $h^*(V) = 1$, and so, $S^*(V) = 0$.

We summarize this observation as below.

Theorem 4.1. *Given α, J, and V, the optimal voting strategy, denoted by S^*, decides the result as follows:*

1. $S^*(V) = 1$ *if* $\alpha \cdot \prod_{i=1}^{n} q_i^{(1-v_i)} \cdot (1-q_i)^{v_i} <$
   $(1-\alpha) \cdot \prod_{i=1}^{n} q_i^{v_i} \cdot (1-q_i)^{(1-v_i)}$*; or*
2. $S^*(V) = 0$*, otherwise.*

Note that S^* is a deterministic voting strategy, and it's essentially a voting strategy based on the Bayes' Theorem [57]. The reason is as follows. According to the Bayes' Theorem, based on the observed voting V, $\Pr(\mathbf{t} = 0 | \mathbf{V} = V) = P_0(V)/\Pr(\mathbf{V} = V)$, and similarly $\Pr(\mathbf{t} = 1 | \mathbf{V} = V) = P_1(V)/\Pr(\mathbf{V} = V)$. Therefore, $P_0(V) - P_1(V) < 0$ indicates $\Pr(\mathbf{t} = 0|\mathbf{V} = V) < \Pr(\mathbf{t} = 1|\mathbf{V} = V)$. And so, 1 has a higher probability to be the true answer than 0. Thus, the voting strategy based on the Bayes' Theorem returns 1 as the result, which is consistent with S^* in Theorem 4.1. Next, we give a formal definition for Bayesian Voting (BV) and summarize the above observation in Theorem 4.1.

Definition 4.4. *The voting strategy based on the Bayes' Theorem, denoted by Bayesian Voting (or BV in short), returns the result as* 1, *if* $\Pr(\mathbf{t} = 0) \cdot \Pr(\mathbf{V} = V|\mathbf{t} = 0) < \Pr(\mathbf{t} = 1) \cdot \Pr(\mathbf{V} = V|\mathbf{t} = 0)$*; or* 0*, otherwise.*

Corollary 4.1. *BV is optimal with respect to JQ, i.e., $S^* = BV$.*

Note that the BV is also used in [19, 92, 127]. In the rest of the chapter, we use S^* and BV interchangeably. We remark that the optimality of BV is based on two assumptions: (1) the prior and workers' qualities are known in advance; (2) JQ (Definition 4.3) is adopted to measure the goodness of a voting strategy.

Example 10. *Let us reconsider Figure 4.2 and see how $JQ(J, BV, \alpha)$ is computed. The 5th column shows results given by BV. The two numbers in bracket correspond to $P_0(V)$ and $P_1(V)$, respectively. The value in parenthesis is the estimated true answer returned by BV. We again use a symbol $\surd$ to indicate the correct voting result. Take $V = \{1, 0, 0\}$ and $t = 0$ as an example. Since $\alpha \cdot (1 - q_1) \cdot q_2 \cdot q_3 = 0.018 < (1 - \alpha) \cdot q_1 \cdot (1 - q_2) \cdot (1 - q_3) = 0.072$, we have $BV(V) = 1 \neq t$, thus 0.018 is not added into $JQ(J, BV, \alpha)$. Otherwise, for $V = \{1, 0, 0\}$ and $t = 0$, similarly we derive that 0.072 is added in $JQ(J, BV, \alpha)$. Recall Example 9, when $V = \{1, 0, 0\}$, if we consider two cases of t, then 0.072 is added into $JQ(J, MV, \alpha)$; but here we have seen in Example 9 that 0.018 is added into $JQ(J, BV, \alpha)$. By considering all V and t, we have $JQ(J, BV, \alpha) = 90\% > JQ(J, MV, \alpha) = 79.2\%$.*

Intuitively, the reason why BV outperforms other voting strategies is that BV considers the prior and worker's qualities in deriving the result of a voting V, and only the one with larger posterior probability is returned. Thus, it is more likely to return a correct answer than other strategies. For example, assume $\alpha = 0.5$ and the voting $V = \{0, 1, 1\}$ is given by workers with individual quality 0.9, 0.6 and 0.6 respectively. As $0.5 \cdot 0.9 \cdot (1 - 0.6) \cdot (1 - 0.6) > 0.5 \cdot (1 - 0.9) \cdot 0.6 \cdot 0.6$, BV returns 0 as the result. However, MV does not leverage either the prior information or workers' qualities, and so, it returns 1 as the result, which is given by two lower quality workers.

Before we go on, we would like to discuss the effect of q_i for voting strategies. Intuitively, $q_i < 0.5$ indicates that worker j_i is more likely to give an incorrect answer than a correct one. Thus, we can either simply ignore this worker in the jury selection process, or modify her answer according to the specific voting strategy. For example, for MV, we can regard vote 0 as 1 and vote 1 as 0 if the vote is given by a worker whose quality is less than 0.5; for BV, according to its definition, it can reinterpret the vote given by a worker with quality $q_i < 0.5$

as an opposite vote given by a worker with quality $1 - q_i > 0.5$. For example, let us consider $V = \{1, 0, 0\}$ and $t = 1$. Since j_2 is a low-quality worker ($q_2 = 0.3 < 0.5$), BV wisely reinterprets her vote by 1 instead of 0 and views it as a vote given by a high-quality worker (it views $V = \{1, 0, 0\}$ as $\{1, 1, 0\}$ voted by q_1, $1 - q_2 = 0.7$ and q_3). And thus, BV returns the correct result 1. The next lemma states this special property for BV.

Lemma 4.1. *Given α and J. Let $J' = J$ and $V' = V$ except that $q'_{i_0} = 1 - q_{i_0}$ and $v'_{i_0} = 1 - v_{i_0}$ for some i_0. The result of BV remains the same, i.e., $BV(V', J', \alpha) = BV(V, J, \alpha)$.*

Proof. First, we have
$q'_{i_0}{}^{(1-v'_i)} \cdot (1 - q'_{i_0})^{v'_i} = q_i{}^{(1-v_i)} \cdot (1 - q_i)^{v_i}$, and similarly
$q'_{i_0}{}^{v'_i} \cdot (1 - q'_{i_0})^{(1-v'_i)} = q_i{}^{v_i} \cdot (1 - q_i)^{(1-v_i)}$. According to Theorem 4.1, we have $\text{BV}(V', J', \alpha) = \text{BV}(V, J, \alpha)$. □

A direct consequence of Lemma 4.1 is as below.

Corollary 4.2. *Given α and J. Let $J' = J$ except that $q'_{i_0} = 1 - q_{i_0}$ for some i_0. The score of JQ for BV remains the same, i.e., $JQ(J', BV, \alpha) = JQ(J, BV, \alpha)$.*

Proof. From Lemma 4.1 we know that for a $V \in \Omega$, $BV(V, J, \alpha) = BV(V', J', \alpha)$ and it is easy to derive
$\Pr(\mathbf{V} = V, \mathbf{t} = 0 \mid J) = \Pr(\mathbf{V} = V', \mathbf{t} = 0 \mid J')$ and
$\Pr(\mathbf{V} = V, \mathbf{t} = 1 \mid J) = \Pr(\mathbf{V} = V', \mathbf{t} = 1 \mid J')$. [11] As the mapping $V \to V'$

[11]Note that $\Pr(\mathbf{V} = V, \mathbf{t} = 0 \mid J)$ is the computation of $\Pr(\mathbf{V} = V, \mathbf{t} = 0)$ by leveraging α and the jury set J, and similarly $\Pr(\mathbf{V} = V', \mathbf{t} = 0 \mid J')$ is the computation of $\Pr(\mathbf{V} = V', \mathbf{t} = 0)$ by leveraging α and the jury set J'. The other two terms $\Pr(\mathbf{V} = V, \mathbf{t} = 1 \mid J)$ and $\Pr(\mathbf{V} = V', \mathbf{t} = 1 \mid J')$ can be derived in the same way.

defines a one-to-one correspondence between Ω and Ω, thus $JQ(J, BV, \alpha) =$

$$
\begin{aligned}
& \sum_{V \in \Omega} \left[\Pr(\mathbf{V} = V, \mathbf{t} = 0 \mid J) + \Pr(\mathbf{V} = V, \mathbf{t} = 1 \mid J) \right] \\
= & \sum_{V \in \Omega} \left[\Pr(\mathbf{V} = V', \mathbf{t} = 0 \mid J') + \Pr(\mathbf{V} = V', \mathbf{t} = 1 \mid J') \right] \\
= & \sum_{V' \in \Omega} \left[\Pr(\mathbf{V} = V', \mathbf{t} = 0 \mid J') + \Pr(\mathbf{V} = V', \mathbf{t} = 1 \mid J') \right] \\
= & \ JQ(J', BV, \alpha).
\end{aligned}
$$

□

Corollary 4.2 is an important result since once a low-quality worker (i.e., $q_i < 0.5$) is involved, we can transform her to the one with quality of $1 - q_i > 0.5$ without affecting JQ for BV. And therefore, in order to simplify our analysis in the next two sections, we assume that $q_i \geq 0.5$ for all workers. In fact, in our experiments with real human workers, we observed that their qualities were generally well above 0.5. We thus assume that $q_i \geq 0.5$ in our subsequent discussions, without loss of generality.

4.4 Computing Jury Quality for Optimal Strategy

In the previous section, we have proved that BV is the optimal voting strategy with respect to JQ. And thus, in order to solve JSP, we only need to figure out J^* such that $JQ(J^*, BV, \alpha)$ is maximized. An immediate question is whether $JQ(J, BV, \alpha)$ can be computed efficiently. Unfortunately, we find that computing $JQ(J, BV, \alpha)$ is NP-hard (Section 4.4.1). To alleviate this, we propose an efficient approximation algorithm with theoretical bounds to compute JQ for BV in this section.

4.4.1 NP-hardness of computing $JQ(J, BV, \alpha)$

Note that [33] has previously proposed an efficient algorithm to compute $JQ(J, MV, 0.5)$ in $\mathcal{O}(n \log n)$. However, this polynomial algorithm cannot be adapted to compute JQ for BV.

The main reason is that computing JQ for BV is an NP-hard problem, which is denoted as below.

Theorem 4.2. *Given α and J, computing JQ for BV, or $JQ(J, BV, \alpha)$, is NP-hard.*

The idea of the proof is that the partition problem [153] (a well-known NP-complete problem) can be reduced to the problem of computing $JQ(J, BV, 0.5)$ for some J. And so, the computation of $JQ(J, BV, 0.5)$ is not easier than the partition problem. As computing $JQ(J, BV, 0.5)$ is not in NP (it is not a decision problem), hence the problem of computing $JQ(J, BV, \alpha)$ for $\alpha \in [0, 1]$ is NP-hard.

The partition problem is a decision problem, which is to decide if a given multi-set W of integer values can be partitioned into two disjoint multi-sets W_1 and W_2 such that the sum of elements in W_1 is equal to the sum of elements in W_2, i.e., $\sum_{e \in W_1} e = \sum_{e \in W_2} e$. The NP-hard proof of $JQ(J, BV, 0.5)$ is based on the detailed proof Lemma 4.2 (which proves that adding a worker will not decrease the JQ). The basic idea of the reduction is that based on the given W, we try to construct a J and $J' = J \cup \{j_{n+1}\}$, then if JQ for BV can be computed (which means that $JQ(J, BV, 0.5)$ and $JQ(J', BV, 0.5)$ can be computed), then the answer ("yes" or "no") to the partition problem of W can be derived based on whether $JQ(J', BV, 0.5) > JQ(J, BV, 0.5)$ or $JQ(J', BV, 0.5) = JQ(J, BV, 0.5)$.

We formally present our proof below.

Proof. Based on the detailed proof in Lemma 4.2, under $\alpha = 0.5$, we know that the JQ will not decrease by adding a worker, and the JQ will increase if there exists a $V \in \Omega$, such that $P(V|\mathbf{t} = 0) \cdot (1 - q_{n+1}) < P(V|\mathbf{t} = 1) \cdot q_{n+1}$ under the condition that $P(V|\mathbf{t} = 0) \geq P(V|\mathbf{t} = 1)$. That is, we have to verify the existence of

$$0 \leq \ln \frac{P(V|\mathbf{t}=0)}{P(V|\mathbf{t}=1)} < \ln \frac{q_{n+1}}{1-q_{n+1}}.$$

By defining $\sigma(q) = \ln \frac{q}{1-q}$, $g(V) = \ln \frac{P(V|\mathbf{t}=0)}{P(V|\mathbf{t}=1)}$ and expand $P(V|\mathbf{t}=0)$, $P(V|\mathbf{t}=1)$, then it is transformed to verify the existence of

$$0 \leq g(V) = \sum_{i=1}^{n} \left[\ (1-2v_i) \cdot \sigma(q_i)\ \right] < \sigma(q_{n+1}). \tag{4.5}$$

That is to say, we need to know if the smallest positive $g(V)$ is less than $\sigma(q_{n+1})$, and we have two observations for $g(V)$:

(1) for a $V \in \Omega$, there always exists a $\overline{V} = \{\bar{v}_1, \ldots, \bar{v}_n\} \in \Omega$, where $\bar{v}_i = 1 - v_i$ for $i \in [1, n]$, and as $1 - 2\bar{v}_i = 2v_i - 1$, then $g(\overline{V}) = -g(V)$, which means that the values for $g(V)$ where $V \in \Omega$ are symmetrically distributed around 0, so we only have to verify the existence of

$$\min_{V \in \Omega} |\ g(V)\ | < \sigma(q_{n+1});$$

(2) for a $V \in \Omega$, we consider two possible cases for each vote in $g(V)$:

(i) if $v_i = 0$, then $(1-2v_i) \cdot \sigma(q_i) = \sigma(q_i)$ is added;

(ii) if $v_i = 1$, then $-\sigma(q_i)$ is added.

This motivates us to define $D = \{d_1, d_2 \cdots d_n\}$ for a V, where $d_i = 1 - 2v_i$ for $i \in [1, n]$. If we consider all possible $V \in \Omega$, then $D \in \{-1, +1\}^n$.

In summary, in order to decide if adding a worker will increase the JQ, we have to verify if

$$G(J) = \min_{d_i = \{-1,+1\}} |\ \sum_{i=1}^{n} d_i \cdot \sigma(q_i)\ | < \sigma(q_{n+1}). \tag{4.6}$$

For example, if $J = \{j_1, j_2\}$ and $q_1 = 0.75$, $q_2 = 0.7$. Then $\sigma(q_1) = \ln \frac{0.75}{1-0.75} \approx 1.099$ and $\sigma(q_2) \approx 0.847$, so $G(J) \approx 0.252$. If we add a worker with quality $q_3 = 0.6$, as $\sigma(q_3) \approx 0.405 > G(J)$, then the JQ will increase, or $JQ(J', BV, 0.5) = 0.765 > JQ(J, BV, 0.5) = 0.75$; otherwise, if the added worker is $q_3 = 0.55$, as $\sigma(q_3) \approx 0.201 \leq G(J)$, then the JQ stays the same, or $JQ(J', BV, 0.5) = JQ(J, BV, 0.5) = 0.75$.

Then we can easily derive by proof of contradiction that
(1) $JQ(J', BV, 0.5) > JQ(J, BV, 0.5) \Rightarrow G(J) < \sigma(q_{n+1})$;
(2) $JQ(J', BV, 0.5) = JQ(J, BV, 0.5) \Rightarrow G(J) \geq \sigma(q_{n+1})$.

Given a multi-set $W = \{w_1, w_2 \cdots w_n\}$ containing n integers, we reduce the partition problem of W to computing JQ by setting $\sigma(q_i) = w_i$ for $i \in [1, n]$, then $q_i = \frac{e^{w_i}}{1+e^{w_i}} \in [0.5, 1]$, and we set $\sigma(q_{n+1}) = 0.5$, then $q_{n+1} = \frac{\sqrt{e}}{1+\sqrt{e}} \in (0.5, 1)$.

Based on our reduction, if JQ is computable, then we can determine whether $JQ(J', BV, 0.5)$ is greater or equal to $JQ(J, BV, 0.5)$, which means that whether the following Equation

$$G(J) = \min_{d_i \in \{-1,+1\}} \mid \textstyle\sum_{i=1}^{n} d_i \cdot w_i \mid < 0.5$$

satisfies or not can be determined. Consider two cases:
(1) $G(J) < 0.5$, as $G(J)$ is a non-negative integer, then $G(J) = 0$, i.e., $\min_{d_i \in \{-1,+1\}} \mid \sum_{i=1}^{n} d_i \cdot w_i \mid = 0$, so the answer to the partition problem of W is "yes", as $d_i = 1$ or -1 can be regarded as the partition;
(2) $G(J) >= 0.5$, then we can derive $G(J) \neq 0$, and the answer to the partition problem of W is "no" (since if the answer is "yes", then $G(J)$ must be 0).

In summary, by the reduction we have
(1) $JQ(J', BV, 0.5) > JQ(J, BV, 0.5) \Rightarrow G(J) < 0.5 \Rightarrow G(J) = 0 \Rightarrow$ the answer to the partition problem of W is "yes";
(2) $JQ(J', BV, 0.5) = JQ(J, BV, 0.5) \Rightarrow G(J) \geq 0.5 \Rightarrow G(J) \neq 0 \Rightarrow$ the answer to the partition problem of W is "no".

Thus we have proved the reduction from computing JQ to the partition problem.

□

Due to this hardness result, we propose an approximation algorithm. We first discuss the computation of $JQ(J, BV, 0.5)$ in Section 4.4.2 and 4.4.3, and give its approximation error bound in Section 4.4.4. Finally, we briefly discuss how to adapt the algorithm to $\alpha \in [0, 1]$ in Section 4.4.5.

4.4.2 Analysis of Computing $JQ(J, BV, 0.5)$

Let us first give some basic analysis for computing $JQ(J, BV, 0.5)$ before we introduce our approximation algorithm. To facilitate our analysis, we first define a few symbols.

- $A_0(V) = 0.5 \cdot \Pr(V \mid \mathbf{t} = 0) \cdot \mathbb{1}_{\{BV(V)=0\}}$;
- $A_1(V) = 0.5 \cdot \Pr(V \mid \mathbf{t} = 1) \cdot \mathbb{1}_{\{BV(V)=1\}}$;
- $\overline{V} = \{\bar{v}_1, \bar{v}_2, \ldots, \bar{v}_n\}$, where $\bar{v}_i = 1 - v_i$ $(1 \leq i \leq n)$.

From Figure 4.2 we observe that $A_0(V) = A_1(\overline{V})$. For example, $A_0(\{0,1,0\}) = A_1(\{1,0,1\}) = 0.108$ and $A_0(\{1,0,1\}) = A_1(\{0,1,0\}) = 0$. The observation motivates us to consider $A_0(V)$ and $A_1(\overline{V})$ together, and we can prove that

$$\begin{aligned} JQ(J, BV, 0.5) &= \textstyle\sum_{V \in \Omega} [\, A_0(V) + A_1(V) \,] \\ &= \textstyle\sum_{V \in \Omega} [\, A_0(V) + A_1(\overline{V}) \,], \end{aligned} \tag{4.7}$$

as $V \to \overline{V}$ defines a one-to-one correspondence between Ω and Ω.

We further define $u(V)$ and $w(V)$ as follows.

$$\begin{aligned} u(V) &= \ln \Pr(V | \mathbf{t} = 0) = \textstyle\sum_{i=1}^{n} [\, (1 - v_i) \ln q_i + v_i \ln(1 - q_i) \,], \\ w(V) &= \ln \Pr(V | \mathbf{t} = 1) = \textstyle\sum_{i=1}^{n} [\, v_i \ln q_i + (1 - v_i) \ln(1 - q_i) \,], \end{aligned}$$

Let $R(V) = u(V) - w(V)$ and $\sigma(q_i) = \ln \frac{q_i}{1-q_i}$ (as $q_i \geq 0.5$, $\sigma(q_i) \geq 0$), we have

$R(V) = u(V) - w(V)$	$A_0(V)$	$A_1(\overline{V})$	$A_0(V) + A_1(\overline{V})$
$R(V) > 0$	$e^{u(V)}/2$	$e^{u(V)}/2$	$e^{u(V)}$
$R(V) = 0$	$e^{u(V)}/2$	0	$e^{u(V)}/2$
$R(V) < 0$	0	0	0

Figure 4.3: Expressing $A_0(V) + A_1(\overline{V})$ using $R(V)$ and $u(V)$.

$$R(V) = \sum_{i=1}^{n}[\ (1-2v_i)\cdot\sigma(q_i)\ ],\ \ e^{u(V)} = \prod_{i=1}^{n} q_i^{(1-v_i)}\cdot(1-q_i)^{v_i}. \tag{4.8}$$

As illustrated in Figure 4.3, we can express $A_0(V) + A_1(\overline{V})$ based on the sign of $R(V)$ and the value of $u(V)$. [12] And therefore, $JQ(J, BV, 0.5) = \sum\limits_{V\in\Omega}[\mathbb{1}_{\{R(V)>0\}}\cdot e^{u(V)} + \mathbb{1}_{\{R(V)=0\}}\cdot\frac{e^{u(V)}}{2}]$.

Motivated by the above formula, we can apply an iterative approach which expands J with one more worker at each iteration and thus compute $JQ(J, BV, 0.5)$ in n total iterations. In the k-th iteration, we consider $V^k \in \{0,1\}^k$. We aim to construct a map structure with $(key, prob)$ pairs, where the domain of key is $\{\ R(V^k) \mid V^k \in \{0,1\}^k\ \}$, and the corresponding value of the key, or $prob$ is

$$prob = \textstyle\sum_{R(V^k)=key\ \wedge\ V^k\in\{0,1\}^k}\ e^{u(V^k)}\ . \tag{4.9}$$

Suppose in the k-th iteration, such a map structure is constructed. Then in the next iteration, we can generate a new map structure from the old map structure: for each $(key, prob)$ in the old map structure, based on the possible choices of v_{k+1} and by considering two formulas in Equation 4.8, we have

1. for $v_{k+1} = 0$, the new key $key + \sigma(q_{k+1})$ is generated and $prob\cdot q_{k+1}$ is added to the prob of the new key;
2. for $v_{k+1} = 1$, the new key $key - \sigma(q_{k+1})$ is generated and $prob\cdot(1-q_{k+1})$

[12]Note that the reason why $A_0(V) \neq A_1(\overline{V})$ when $u(V) = w(V)$ is that as $0.5\cdot e^{u(V)} = 0.5\cdot e^{w(V)}$, based on Theorem 4.1, $BV(V) = BV(\overline{V}) = 0$, so $A_0(V) = 0.5\cdot e^{u(V)}$ and $A_1(\overline{V}) = 0$.

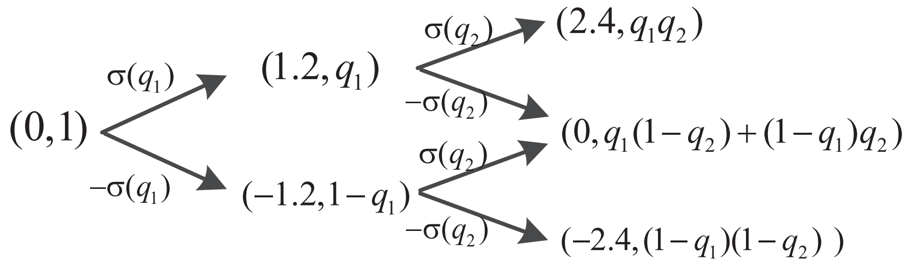


Figure 4.4: Illustrating the Iterative Approach.

is added to the prob of the new key.

Example 11. *We give an example to illustrate the above process in Figure 4.4, where $n = 2$ and $\sigma(q_1) = \sigma(q_2) = 1.2$. In the figure, each pair is represented as "$(key, prob)$". Starting from $(0,1)$, for $v_1 = 0$ and $v_1 = 1$, it respectively creates $(\sigma(q_1) : q_1)$ and $(-\sigma(q_1) : (1 - q_1))$ in the first iteration. Then it leverages the stored $(key, prob)$ pair to generate new pairs in the second iteration by considering different v_2. Note that as $\sigma(q_1) = \sigma(q_2)$, if $(\sigma(q_1), q_1)$ takes $v_2 = 1$ and $(-\sigma(q_1), (1 - q_1))$ takes $v_2 = 0$, then they go to the same $key = 0$, and their new prob, $q_1 \cdot (1 - q_2)$ and $(1 - q_1) \cdot q_2$ are added together.*

4.4.3 Bucket-Based Approximation Algorithm

By our intractability result for JQ we know that the domain of keys, or $\{R(V) \mid V \in \{0,1\}^n\}$ is exponential. In order to address this issue, we set a controllable parameter *numBuckets* and map $\sigma(q_i)$ to a bucket integer $b_i \in [\ 0, numBuckets\ ]$, where the interval between adjacent buckets, called bucket-size (denoted as δ) is the same. Suppose $numBuckets = d \cdot n$, i.e., a constant multiple of the number of jury members, then, for each iteration, the number of possible values in the *key* is bounded by $2dn^2 + 1$ (in the range $[-dn^2, dn^2]$) Considering all n iterations, the time complexity is bounded by $\mathcal{O}(dn^3)$, which is of polynomial order.

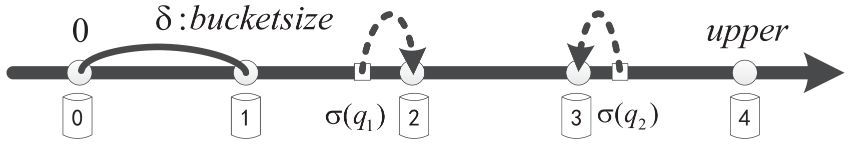


Figure 4.5: Principle of the Bucket Array.

We detail this process in Algorithm 4. To start with, the function GetBucketArray assigns b_i to worker j_i based on $\sigma(q_i)$. The computation of b_i proceeds as follows. At first, we fix a range $[0, upper]$ where $upper = \max_{i\in[1,n]} \{\sigma(q_i)\}$. Then, we divide the range into *numBuckets* of buckets with equal length, denoted by $\delta = \frac{upper}{numBuckets}$. Finally, each worker j_i's bucket number b_i is assigned to its closet bucket: $b_i = \left\lceil \frac{\sigma(q_i)}{\delta} - \frac{1}{2} \right\rceil$. Figure 4.5 illustrates an example where $numBuckets = 4$. Since $\sigma(q_1)$ is the closet to bucket number 2, so $b_1 = 2$, and similarly $b_2 = 3$.

After mapping each worker to a bucket b_i, we iterate over n workers (step 7-20). For a given worker j_i, based on each $(key, prob)$ pair in the stored map SM, we update *key* and *prob*, based on two possible values of v_i (steps 14-19)[13] in the new map M. SM will then be updated as the newly derived map M for next iteration (step 20). Finally, the $(key, value)$ pairs in SM are used in the evaluation of the Jury Quality (steps 21-25), based on the cases in Figure 4.3.

Pruning Techniques. We can further improve the running time of the approximation algorithm by applying some pruning techniques in Algorithm 6, in order to prune redundant computations. For example, assume $n = 5$, and the derived $b = [3, 7, 4, 3, 2]$. In the second iteration, consider the $key = 3 + 7 = 10$ ($v_1 = 0$ and $v_2 = 0$). No matter what the rest of the three votes are, the aggregated buckets cannot be negative (since $4 + 3 + 2 = 9 < 10$), so we can safely prune the search space for $key = 10$ (which takes $2^3 = 8$ computations). To fur-

[13]Note that as we only care about the sign ($+$, 0 or $-$) of $R(V)$, and we approximate $\sigma(q_i)$ as $\delta \cdot b_i$, we can map $\sigma(q_i)$ to b_i and add/subtract the integer b_i.

Algorithm 4 EstimateJQ (Chapter 4).

Input: $J = \{j_1, j_2 \cdots j_n\}$, $numBuckets$, n
Output: $\widehat{JQ}$

```
1: $b = \texttt{GetBucketArray}(J, numBuckets, n)$;
2: $b = \texttt{Sort}(b)$; // sort in decreasing order, for pruning
3: $J = \texttt{Sort}(J)$; // sort based on worker quality, similar as above
4: $aggregate = \texttt{AggregateBucket}(b, n)$; // for pruning
5: $\widehat{JQ} = 0$; // estimated JQ
6: $SM[\,0\,] = 1$; //initialize a map structure
7: for $i = 1$ to $n$ do
8:     $M = map()$; //initialize an empty map structure
9:     for $(key, prob) \in SM$ do
10:        $flag, value = \texttt{Prune}(key, prob, aggregate[i])$;
11:        if $flag$ =true then
12:            $\widehat{JQ} += value$;
13:            continue // for pruning
14:        end if
15:        if $key + b[i] \notin M$ then
16:            $M[\,key + b[i]\,] = 0$;
17:        end if
18:        $M[\,key + b[i]\,] += prob \cdot q_i$; // for $v_i = 0$
19:        if $key - b[i] \notin M$ then
20:            $M[\,key - b[i]\,] = 0$;
21:        end if
22:        $M[\,key - b[i]\,] += prob \cdot (1 - q_i)$; // for $v_i = 1$
23:    end for
24:    $SM = M$;
25: end for
26: for $(key, prob) \in SM$ do
27:    if $key > 0$ then
28:        $\widehat{JQ} += prob$;
29:    end if
30:    if $key = 0$ then
31:        $\widehat{JQ} += 0.5 \cdot prob$;
32:    end if
33: end for
34: return $\widehat{JQ}$;
```

Algorithm 5 GetBucketArray (Chapter 4).

```
Input:  $J = \{j_1, j_2 \cdots j_n\}$, $numBuckets$ , $n$
Output:  $b$
1: $b = [\,0, 0 \cdots 0\,]$; // $n$ elements, all 0
2: $upper = \max_{i \in [1,n]} \sigma(q_i)$; // compute $upper$
3: $\delta = \frac{upper}{numBuckets}$; // bucketsize
4: for $i = 1$ to $n$ do
5:     $b[\,i\,] = \left\lceil \frac{\sigma(q_i)}{\delta} - \frac{1}{2} \right\rceil$; // or $b_i$ in our explanation
6: end for
7: return $b$;
```

Algorithm 6 Pruning Techniques (Chapter 4).

```
def AggregateBucket($b, n$):
    $aggregate = [\,0, 0 \cdots 0\,]$; // $n$ elements, all 0
    for $i = n$ to 1 do
        if $i = n$ then
            $aggregate[i] = b[i]$;
        else
            $aggregate[i] = aggregate[i+1] + b[i]$;
        end if
    end for
    return $aggregate$

def Prune($key, prob, number$):
    $flag$ =false;
    if $key > 0$ and $key - number > 0$ then
        $flag$ =true; $value = prob$;
    end if
    if $key < 0$ and $key + number < 0$ then
        $flag$ =true; $value = 0$;
    end if
    return $flag$, $value$;
```

ther increase the efficiency, in Algorithm 6 we first sort the bucket array and J in decreasing order (step 2-3), guaranteeing that the highest bucket is considered first, and then compute the *aggregate* array via `AggregateBucket` (step 4), which makes the pruning phase (step 10-13) more efficient. The function `Prune` uses *aggregate* to decide whether to prune or not.

4.4.4 Approximation Error Bound

Let $\widehat{JQ}$ denotes the estimated value returned by Algorithm 4, and JQ denotes the real Jury Quality. We evaluate the *additive error* bound on $|JQ - \widehat{JQ}|$ and can prove that:

$$\widehat{JQ} \leq JQ \quad \text{and} \quad JQ - \widehat{JQ} < e^{\frac{n\delta}{4}} - 1, \tag{4.10}$$

where n is the number of workers and $\delta = \frac{upper}{d \cdot n}$ is the bucketsize. We give detailed proof below:

Proof. For a $V \in \Omega$, let $\overline{V} = \{\bar{v}_1, \ldots, \bar{v}_n\}$, where $\bar{v}_i = 1 - v_i$ for $i \in [1, n]$. Recall the definition of $A_0(V)$, $A_1(V)$, $u(V)$, $w(V)$ and $R(V)$ in Section 4.4.2, we express $A_0(V) + A_1(V) + A_0(\overline{V}) + A_1(\overline{V})$ by $u(V)$ and $w(V)$ based on different signs of $R(V)$ in Figure 4.6. It is easy to verify that if we denote $\widetilde{\Omega} = \{0\} \times \{0,1\}^{n-1}$ ($|\widetilde{\Omega}| = 2^{(n-1)}$), then

$$JQ(J, BV, 0.5) = \textstyle\sum_{V \in \widetilde{\Omega}} [\, A_0(V) + A_1(V) + A_0(\overline{V}) + A_1(\overline{V}) \,].$$

For a $V \in \widetilde{\Omega}$, let us denote $T(V) = A_0(V) + A_1(V) + A_0(\overline{V}) + A_1(\overline{V})$, and from Figure 4.6 we can derive that $T(V) = \max\{e^{u(V)}, e^{w(V)}\}$. We can also get

$$JQ(J, BV, 0.5) = \textstyle\sum_{V \in \widetilde{\Omega}} T(V).$$

Recall Algorithm 4, which makes concessions and map $\sigma(q_i)$ to a bucket number b_i, where each bucket is of size δ. So it approximates $\sigma(q_i)$ by δb_i. Thus if we denote

$$\widehat{R}(V) = \textstyle\sum_{i=1}^{n} [\, (1 - 2v_i) \cdot \delta b_i \,],$$

then based on the sign of $\widehat{R}(V)$, the values of $\widehat{A_0}(V)$, $\widehat{A_1}(V)$, $\widehat{A_0}(\overline{V})$ and $\widehat{A_1}(\overline{V})$ are expressed in Figure 4.7. Let us denote $\widehat{T}(V) = \widehat{A_0}(V) + \widehat{A_1}(V) + \widehat{A_0}(\overline{V}) + \widehat{A_1}(\overline{V})$. Thus based on the sign $(+, 0,$ or $-)$ of $\widehat{R}(V)$, the value of $\widehat{T}(V)$ will be

	$A_0(V)$	$A_1(V)$	$A_0(\overline{V})$	$A_1(\overline{V})$	sum
$R(V) > 0$	$e^{u(V)}/2$	0	0	$e^{u(V)}/2$	$e^{u(V)}$
$R(V) = 0$	$e^{u(V)}/2$	$e^{w(V)}/2$	0	0	$\left(e^{u(V)} + e^{w(V)}\right)/2$
$R(V) < 0$	0	$e^{w(V)}/2$	$e^{w(V)}/2$	0	$e^{w(V)}$

Figure 4.6: Illustrating $A_0(V)$, $A_1(V)$, $A_0(\overline{V})$, and $A_1(\overline{V})$.

($e^{u(V)}$, $\frac{e^{u(V)}+e^{w(V)}}{2}$, or $e^{w(V)}$) as Figure 4.7 shows, which is the same as Figure 4.6, as Algorithm 4 only makes concessions to the *key* by approximating $R(V)$ as $\widehat{R}(V)$. Then the estimated

$$\widehat{JQ}(J, BV, 0.5) = \sum_{V \in \widetilde{\Omega}} \widehat{T}(V)$$

So we try to find an error bound for the real $T(V)$ in JQ and computed $\widehat{T}(V)$ in $\widehat{JQ}$, i.e., $|T(V) - \widehat{T}(V)|$.

From the analysis above we know that $T(V) = \max\{e^{u(V)}, e^{w(V)}\}$ and the error occurs when the sign of $\widehat{R}(V)$ is different from the sign of $R(V)$. For example, if $R(V) > 0$ (i.e., $u(V) > w(V)$), then $T(V) = e^{u(V)}$ as Figure 4.6 shows; while if the estimated $\widehat{R}(V) \leq 0$, say $\widehat{R}(V) = 0$, then $\widehat{T}(V) = \frac{e^{u(V)}+e^{w(V)}}{2} < e^{u(V)}$ as Figure 4.7 shows. Since $T(V) = \max\{e^{u(V)}, e^{w(V)}\}$, then $\widehat{T}(V) \leq T(V)$, thus $\widehat{JQ}(J, BV, 0.5) \leq JQ(J, BV, 0.5)$.

Generally, we can derive that
(1) $\widehat{T}(V) \leq T(V)$ and $\widehat{JQ}(J, BV, 0.5) \leq JQ(J, BV, 0.5)$;
(2) the error occurs if there exists a $V \in \widetilde{\Omega}$, such that the sign of $\widehat{R}(V)$ is different from the sign of $R(V)$, and the error, or $T(V) - \widehat{T}(V)$ is bounded by the value $|e^{u(V)} - e^{w(V)}|$.

From Algorithm 4 we know that $|\sigma(q_i) - \delta b_i| \leq \frac{\delta}{2}$ for any worker ($i \in [1, n]$).

	$\widehat{A_0}(V)$	$\widehat{A_1}(V)$	$\widehat{A_0}(\overline{V})$	$\widehat{A_1}(\overline{V})$	*sum*
$\widehat{R}(V) > 0$	$e^{u(V)}/2$	0	0	$e^{u(V)}/2$	$e^{u(V)}$
$\widehat{R}(V) = 0$	$e^{u(V)}/2$	$e^{w(V)}/2$	0	0	$\left(e^{u(V)} + e^{w(V)}\right)/2$
$\widehat{R}(V) < 0$	0	$e^{w(V)}/2$	$e^{w(V)}/2$	0	$e^{w(V)}$

Figure 4.7: Illustrating $\widehat{A_0}(V)$, $\widehat{A_1}(V)$, $\widehat{A_0}(\overline{V})$, and $\widehat{A_1}(\overline{V})$.

As we have

$$\begin{aligned} |R(V) - \widehat{R}(V)| &= | \textstyle\sum_{i=1}^{n}(1-2v_i)\cdot(\sigma(q_i) - \delta b_i) | \\ &\leq \textstyle\sum_{i=1}^{n} | (1-2v_i)\cdot(\sigma(q_i) - \delta b_i) | \\ &= \textstyle\sum_{i=1}^{n} | \sigma(q_i) - \delta b_i | \leq \dfrac{n\delta}{2}, \end{aligned}$$

then we can derive that if $|R(V)| > \frac{n\delta}{2}$, the signs of $R(V)$ and $\widehat{R}(V)$ are the same. So we try to know the bound of $|\ e^{u(V)} - e^{w(V)}\ |$ under the constraint $|R(V)| = |u(V) - w(V)| \leq \frac{n\delta}{2}$. Note that we also have $u(V) + w(V) = \sum_{i=1}^{n}[\ \ln q_i + \ln(1-q_i)\ ] = \sum_{i=1}^{n} \ln\ [\ q_i \cdot (1-q_i)\ ] \leq -n\ln 4$.

So the problem is, what is the bound for $|e^{u(V)} - e^{w(V)}|$ with the constraint below?

$$\begin{cases} -\frac{n\delta}{2} \leq u(V) - w(V) \leq \frac{n\delta}{2}, & (1) \\ u(V) + w(V) \leq -n\ln 4. & (2) \end{cases}$$

Let us denote $u(V) + w(V) = \ln c$, then from (2) we get $c \leq \frac{1}{4^n}$, and as $w(V) = \ln c - u(V)$, we get

$$|e^{u(V)} - e^{w(V)}| = |e^{u(V)} - \frac{c}{e^{u(V)}}|.$$

It is easy to know that $g(x) = x - \frac{c}{x}$ $(c > 0)$ is a strictly increasing function in the domain $x > 0$ (as $g'(x) = 1 + \frac{c}{x^2} > 0$ for $x > 0$). And take $w(V) = \ln c - u(V)$

in (1), we can derive

$$\frac{\ln c}{2} - \frac{n\sigma}{4} \leq u(V) \leq \frac{\ln c}{2} + \frac{n\sigma}{4},$$

then we get

$$|e^{u(V)} - \frac{c}{e^{u(V)}}| \leq \sqrt{c} \cdot e^{\frac{n\delta}{4}} - \sqrt{c} \cdot e^{-\frac{n\delta}{4}}.$$

By setting $e^{\frac{n\delta}{4}} = 1 + \epsilon$, we get

$$\begin{aligned}|e^{u(V)} - e^{w(V)}| &\leq \sqrt{c} \cdot e^{\frac{n\delta}{4}} - \sqrt{c} \cdot e^{-\frac{n\delta}{4}} = \sqrt{c} \cdot \epsilon \cdot (1 + \frac{1}{1+\epsilon}) \\ &< 2\sqrt{c} \cdot (e^{\frac{n\delta}{4}} - 1) \leq \frac{1}{2^{n-1}} \cdot (e^{\frac{n\delta}{4}} - 1).\end{aligned}$$

So given $V \in \widetilde{\Omega}$, we have computed the bound for $T(V) - \widehat{T}(V)$. As $|\widetilde{\Omega}| = 2^{(n-1)}$, then by considering all $2^{(n-1)}$ terms, the error is bounded by

$$\begin{aligned}&JQ(J, BV, 0.5) - \widehat{JQ}(J, BV, 0.5) \\ &< \textstyle\sum_{V \in \widetilde{\Omega}} \frac{1}{2^{n-1}} \cdot (e^{\frac{n\delta}{4}} - 1) = e^{\frac{n\delta}{4}} - 1.\end{aligned}$$

□

We next show that the bound is very small ($<$ 1% by setting $d \geq 200$) in real cases. First we notice that (i) $\sigma(q)$ is a strictly increasing function and (ii) $\sigma(0.99) < 5$. So let us assume $upper < 5$. We can safely make the assumption, since if not, there exists a worker of quality $q_i > 0.99$, and then $JQ \in (0.99, 1]$, as Lemma 4.2 will show. Then we can just return $\widehat{JQ} = q_i > 0.99$, which makes $JQ - \widehat{JQ} < 1\%$. After dividing the interval $[0, upper]$ into $d \cdot n$ equal buckets, we have $\delta < \frac{5}{d \cdot n}$. Using this δ bound in Equation 4.11, we have $JQ - \widehat{JQ} < e^{\frac{5}{4 \cdot d}} - 1$. By setting $d \geq 200$, the bound is $JQ - \widehat{JQ} < 0.627\% < 1\%$.

4.4.5 Incorporation of Prior

In the previous section, we have assumed a prior $\alpha = 0.5$. Here, we drop this assumption and show how we can adapt our approaches to a generalized prior $\alpha \in [0,1]$, given by the task provider. By Theorem 4.3, it turns out this is equivalent to computing $JQ(J', BV, 0.5)$, where J' is obtained by adding a worker (with quality α) to J:

Theorem 4.3. *Given α and J, $JQ(J, BV, \alpha) = JQ(J', BV, 0.5)$, where $J' = J \cup \{j_{n+1}\}$ and $q_{n+1} = \alpha$.*

Proof. Let $\widehat{\Omega} = \{0,1\}^{(n+1)}$, and in order to prove the theorem, we have to verify

$$\begin{aligned}
&\sum_{V\in\Omega} \left[\alpha \cdot P(V|\mathbf{t}=0)\cdot \mathbb{1}_{\{BV(V)=0\}} + (1-\alpha)\cdot P(V|\mathbf{t}=1)\cdot \mathbb{1}_{\{BV(V)=1\}}\right] \\
&= \sum_{V\in\widehat{\Omega}} 0.5 \cdot \left[\, P(V|\mathbf{t}=0)\cdot \mathbb{1}_{\{BV(V)=0\}} + P(V|\mathbf{t}=1)\cdot \mathbb{1}_{\{BV(V)=1\}}\right].
\end{aligned} \tag{4.11}$$

To prove Equation 4.11, for a $V \in \Omega$, we define its two mapping counterparts in $\widehat{\Omega}$ as follows:

(1) $\widehat{V} = \{\hat{v}_1, \hat{v}_2, \ldots \hat{v}_{n+1}\}$ where $\hat{v}_i = v_i$ for $i \in [1,n]$ and $\hat{v}_{n+1} = 0$;

(2) $\widehat{V}' = \{\hat{v}'_1, \hat{v}'_2, \ldots \hat{v}'_{n+1}\}$ where $\hat{v}'_i = 1 - v_i$ for $i \in [1,n]$ and $\hat{v}'_{n+1} = 1$).

For example, if $V = \{1,1,0\}$, then $\widehat{V} = \{1,1,0,0\}$ and $\widehat{V}' = \{0,0,1,1\}$. We argue that if we can prove

$$\begin{aligned}
&\alpha \cdot P(V|\mathbf{t}=0)\cdot \mathbb{1}_{\{BV(V)=0\}} + (1-\alpha)\cdot P(V|\mathbf{t}=1)\cdot \mathbb{1}_{\{BV(V)=1\}} \\
=&0.5 \cdot \left[\, P(\widehat{V}|\mathbf{t}=0)\cdot \mathbb{1}_{\{BV(\widehat{V})=0\}} + P(\widehat{V}|\mathbf{t}=1)\cdot \mathbb{1}_{\{BV(\widehat{V})=1\}}\,\right] + \\
&0.5 \cdot \left[\, P(\widehat{V}'|\mathbf{t}=0)\cdot \mathbb{1}_{\{BV(\widehat{V}')=0\}} + P(\widehat{V}'|\mathbf{t}=1)\cdot \mathbb{1}_{\{BV(\widehat{V}')=1\}}\,\right],
\end{aligned} \tag{4.12}$$

then Equation 4.11 can be proved. As the two sides of Equation 4.12 are respectively $A_0(V) + A_1(V)$ and $[A_0(\widehat{V}) + A_1(\widehat{V})] + [A_0(\widehat{V}') + A_1(\widehat{V}')]$ in Equation 4.11, so we only have to prove that the mapping $V \to \{\widehat{V}, \widehat{V}'\}$ defines a one-to-one correspondence from Ω to $\widehat{\Omega}$. Since

(1) for a $V \in \Omega$, as mentioned, it is mapped to $\widehat{V}, \widehat{V}' \in \widehat{\Omega}$;
(2) for a $\widetilde{V} = \{\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_{n+1}\} \in \widehat{\Omega}$: if $\tilde{v}_{n+1} = 0$, then its mapped voting in Ω is V where $v_i = \tilde{v}_i$ for $i \in [1, n]$, and if $\tilde{v}_{n+1} = 1$, then its mapped voting in Ω is V where $v_i = 1 - \tilde{v}_i$ for $i \in [1, n]$.
So we have proved the one-to-one correspondence, thus we only have to prove that Equation 4.12 is correct.

For a specific $V \in \Omega$, we prove the correctness of Equation 4.12 by considering two cases:
(1) $\alpha \cdot P(V|\mathbf{t} = 0) \neq (1 - \alpha) \cdot P(V|\mathbf{t} = 1)$, and w.l.o.g. we assume $\alpha \cdot P(V|\mathbf{t} = 0) > (1 - \alpha) \cdot P(V|\mathbf{t} = 1)$:

In this case we prove Equation 4.12 by proving the following two equations (Equation 4.13 and 4.14):

$$\begin{aligned}&\alpha \cdot P(V|\mathbf{t} = 0) \cdot \mathbb{1}_{\{BV(V)=0\}}\\ =&0.5 \cdot P(\widehat{V}|\mathbf{t} = 0) \cdot \mathbb{1}_{\{BV(\widehat{V})=0\}} + 0.5 \cdot P(\widehat{V}'|\mathbf{t} = 1) \cdot \mathbb{1}_{\{BV(\widehat{V}')=1\}},\end{aligned} \tag{4.13}$$

$$\begin{aligned}&(1 - \alpha) \cdot P(V|\mathbf{t} = 1) \cdot \mathbb{1}_{\{BV(V)=1\}}\\ =&0.5 \cdot P(\widehat{V}|\mathbf{t} = 1) \cdot \mathbb{1}_{\{BV(\widehat{V})=1\}} + 0.5 \cdot P(\widehat{V}'|\mathbf{t} = 0) \cdot \mathbb{1}_{\{BV(\widehat{V}')=0\}}.\end{aligned} \tag{4.14}$$

Here we only prove Equation 4.13 and Equation 4.14 can be proved similarly. Equation 4.13 can be proved by the followings:

$$\begin{cases} \alpha \cdot P(V|\mathbf{t} = 0) = P(\widehat{V}'|\mathbf{t} = 1) = P(\widehat{V}|\mathbf{t} = 0), & \text{(i)} \\ BV(V) = BV(\widehat{V}) = 0,\ BV(\widehat{V}') = 1. & \text{(ii)} \end{cases}$$

For (i), as $P(\widehat{V}'|\mathbf{t} = 1)$

$$\begin{aligned} &= P(\hat{v}'_{n+1} = 1|\mathbf{t} = 1) \cdot \textstyle\prod_{i=1}^{n} q_i^{\hat{v}'_i} \cdot (1 - q_i)^{1-\hat{v}'_i} \\ &= \alpha \cdot \textstyle\prod_{i=1}^{n} q_i^{(1-v_i)} \cdot (1 - q_i)^{v_i} = \alpha \cdot P(V|\mathbf{t} = 0), \end{aligned}$$

and we can prove similarly that $P(\widehat{V}|\mathbf{t} = 0) = \alpha \cdot P(V|\mathbf{t} = 0)$, then (i) can be proved.

For (ii), as $\alpha \cdot P(V|\mathbf{t}=0) > (1-\alpha)\cdot P(V|\mathbf{t}=1)$, then $BV(V)=0$ and we can prove similarly that $P(\widehat{V}'|\mathbf{t}=0) = (1-\alpha)\cdot P(V|\mathbf{t}=1)$, which means that $0.5 \cdot P(\widehat{V}'|\mathbf{t}=1) > 0.5 \cdot P(\widehat{V}'|\mathbf{t}=0)$, then $BV(\widehat{V}')=1$. And $BV(\widehat{V})=0$ can be proved similarly, then (ii) can be proved.
Thus we have proved Equation 4.13, and Equation 4.14 can be proved similarly.

(2) $\alpha \cdot P(V|\mathbf{t}=0) = (1-\alpha)\cdot P(V|\mathbf{t}=1)$:
In this case we prove Equation 4.12 directly. From the above analysis we can prove that $\alpha \cdot P(V|\mathbf{t}=0) = P(\widehat{V}|\mathbf{t}=0)$ and $(1-\alpha)\cdot P(V|\mathbf{t}=1) = P(\widehat{V}'|\mathbf{t}=0)$. As $\alpha \cdot P(V|\mathbf{t}=0) = (1-\alpha)\cdot P(V|\mathbf{t}=1)$, from Theorem 4.1 we know that all $BV(V) = BV(\widehat{V}) = BV(\widehat{V}') = 0$. Thus we can directly prove Equation 4.12 by taking the above derived formulas into it.

Then we have proved Equation 4.12 for the two cases, thus Equation 4.11, or the theorem can be proved. □

Thus we can use Algorithm 4 for any prior α, by adding to the jury a pseudo-worker of quality α. Moreover, the approximation error bound proved in Section 4.4.5 also holds.

To summarize, to compute $JQ(J, BV, \alpha)$, we have developed an approximation algorithm with time complexity $\mathcal{O}(d \cdot (n+1)^3)$, with an additive error bound within 1%, for $d \geq 200$.

4.5 Jury Selection Problem (JSP)

Now we focus on addressing $J^* = \text{argmax}_{J \in \mathcal{C}} JQ(J, BV, \alpha)$, for $\mathcal{C}$, the set of all feasible juries (i.e., $\mathcal{C} = \{J \mid J \subseteq W \wedge \sum_{i=1}^{n} c_i \leq B\}$).

Before formally addressing JSP, we turn our attention to two monotonicity properties of $JQ(J, BV, \alpha)$: with respect to varying the jury size ($|J|$), and with respect to a worker j_i's quality (q_i). These properties can help us solve JSP under certain cost constraints.

Lemma 4.2 (Monotonicity on jury size)**.** *Given α and J , $JQ(J, BV, \alpha) \leq JQ(J', BV, \alpha)$, where $J' = J \cup \{j_{n+1}\}$.*

Proof. For a $V \in \Omega$, recall our definition of $A_0(V)$ and $A_1(V)$ as below:
• $A_0(V) = \alpha \cdot \Pr(V \mid \mathbf{t} = 0) \cdot \mathbb{E}[\mathbb{1}_{\{BV(V)=0\}}]$;
• $A_1(V) = (1-\alpha) \cdot \Pr(V \mid \mathbf{t} = 1) \cdot \mathbb{E}[\mathbb{1}_{\{BV(V)=1\}}]$.
Based on Theorem 4.1, we can derive that

$$A_0(V) + A_1(V) = \max\{\alpha \cdot \Pr(V|\mathbf{t}=0), (1-\alpha) \cdot \Pr(V|\mathbf{t}=1)\}.$$

And if a worker j_{n+1} is added, $V \in \Omega$ becomes two votings: $V^{(0)}$ and $V^{(1)}$ in $\{0,1\}^{n+1}$:
(1) $V^{(0)} = \{v_1^{(0)}, v_2^{(0)}, \ldots, v_{n+1}^{(0)}\}$ where $v_i^{(0)} = v_i$ for $i \in [1, n]$ and $v_{n+1}^{(0)} = 0$;
(2) $V^{(1)} = \{v_1^{(1)}, v_2^{(1)}, \ldots, v_{n+1}^{(1)}\}$ where $v_i^{(1)} = v_i$ for $i \in [1, n]$ and $v_{n+1}^{(1)} = 1$.
So the corresponding

$$\begin{aligned} A_0(V^{(0)}) + A_1(V^{(0)}) &= \max\{\alpha \cdot \Pr(V^{(0)}|\mathbf{t}=0), (1-\alpha) \cdot \Pr(V^{(0)}|\mathbf{t}=1)\}, \\ A_0(V^{(1)}) + A_1(V^{(1)}) &= \max\{\alpha \cdot \Pr(V^{(1)}|\mathbf{t}=0), (1-\alpha) \cdot \Pr(V^{(1)}|\mathbf{t}=1)\}. \end{aligned}$$

So if we can prove the following inequality

$$\begin{aligned} &\max\{\ \alpha \cdot P(V^{(0)} \mid \mathbf{t}=0)\ ,\ (1-\alpha) \cdot P(V^{(0)} \mid \mathbf{t}=1)\ \}\ + \\ &\max\{\ \alpha \cdot P(V^{(1)} \mid \mathbf{t}=0)\ ,\ (1-\alpha) \cdot P(V^{(1)} \mid \mathbf{t}=1)\ \} \\ \geq\ &\max\{\ \alpha \cdot P(V \mid \mathbf{t}=0)\ ,\ (1-\alpha) \cdot P(V \mid \mathbf{t}=1)\ \}, \end{aligned} \tag{4.15}$$

then we can prove the lemma.

In order to prove the above inequality, let us denote $b = \alpha \cdot P(V|\mathbf{t}=0)$ and $r = (1-\alpha) \cdot P(V|\mathbf{t}=1)$, then Equation 4.15 can be transformed to

$$\max\{b \cdot q_{n+1}, r \cdot (1-q_{n+1})\} + \max\{b \cdot (1-q_{n+1}), r \cdot q_{n+1}\} \geq \max\{b, r\}.$$

Without loss of generality, we assume $b \geq r$, then the right hand side (rhs) of Equation 4.15 is b, and for the left hand side (lhs) there are two summands: for the first summand, $b \cdot q_{n+1} \geq r \cdot (1 - q_{n+1})$, while based on the comparison between $b \cdot (1 - q_{n+1})$ and $r \cdot q_{n+1}$, we consider two cases:
(1) $b \cdot (1 - q_{n+1}) \geq r \cdot q_{n+1}$: in this case the lhs becomes $b \cdot q_{n+1} + b \cdot (1 - q_{n+1}) = b$, which is equal to the rhs b;
(2) $b \cdot (1 - q_{n+1}) < r \cdot q_{n+1}$: in this case the lhs becomes $b \cdot q_{n+1} + r \cdot q_{n+1}$, since from the condition of (2) we know $b \cdot q_{n+1} + r \cdot q_{n+1} > b$, so the lhs is greater than the rhs. ☐

A direct consequence of Lemma 4.2 is that "the more workers, the better JQ for BV". So for the case that each worker will contribute voluntarily ($c_i = 0$ for $1 \leq i \leq N$) or the budget constraint satisfies on all subsets of the candidate workers W (i.e., $B \geq \sum_{i=1}^{N} c_i$), we can select all workers in W.

Lemma 4.3 (Monotonicity on worker quality)**.** *Given α and J. Let $J' = J$ except that $q'_{i_0} \geq q_{i_0} \geq 0.5$ for some i_0, then $JQ(J', BV, \alpha) \geq JQ(J, BV, \alpha)$.*

Proof. To prove the lemma, we just have to prove that given α and J, $JQ(J^{(1)}, BV, \alpha) \geq JQ(J^{(2)}, BV, \alpha)$ where $J^{(1)} = J \cup \{j'_{n+1}\}$, $J^{(2)} = J \cup \{j_{n+1}\}$, and $q'_{n+1} \geq q_{n+1} \geq 0.5$.

Based on the proof of Lemma 4.2, recall the definition of b and r, we know that if we can prove the following inequality

$$\begin{aligned} &\max\{\, b \cdot q'_{n+1}, r \cdot (1 - q'_{n+1}) \,\} + \max\{\, b \cdot (1 - q'_{n+1}), r \cdot q'_{n+1} \,\} \\ \geq &\max\{\, b \cdot q_{n+1}, r \cdot (1 - q_{n+1}) \,\} + \max\{\, b \cdot (1 - q_{n+1}), r \cdot q_{n+1} \,\}, \end{aligned} \tag{4.16}$$

then we can prove the lemma. To prove it, w.l.o.g. we assume $b \geq r$, then

$$\max\{\, b \cdot q'_{n+1}, r \cdot (1 - q'_{n+1}) \,\} = b \cdot q'_{n+1}, \text{ and}$$

$$\max\{\, b \cdot q_{n+1}, r \cdot (1 - q_{n+1}) \,\} = b \cdot q_{n+1}.$$

Based on the comparison between $b \cdot (1 - q_{n+1})$ and $r \cdot q_{n+1}$, we consider two cases:
(1) $b \cdot (1 - q_{n+1}) \geq r \cdot q_{n+1}$, then

$$\begin{aligned}
&\max\{b \cdot q'_{n+1}, r \cdot (1 - q'_{n+1})\} + \max\{b \cdot (1 - q'_{n+1}), r \cdot q'_{n+1}\} \\
&\geq b \cdot q'_{n+1} + b \cdot (1 - q'_{n+1}) = b = \\
&\max\{b \cdot q_{n+1}, r \cdot (1 - q_{n+1})\} + \max\{b \cdot (1 - q_{n+1}), r \cdot q_{n+1}\};
\end{aligned}$$

(2) $b \cdot (1 - q_{n+1}) < r \cdot q_{n+1}$, then

$$\begin{aligned}
&\max\{b \cdot q'_{n+1}, r \cdot (1 - q'_{n+1})\} + \max\{b \cdot (1 - q'_{n+1}), r \cdot q'_{n+1}\} \\
&\geq q'_{n+1} \cdot (b + r) \geq q_{n+1} \cdot (b + r) = \\
&\max\{b \cdot q_{n+1}, r \cdot (1 - q_{n+1})\} + \max\{b \cdot (1 - q_{n+1}), r \cdot q_{n+1}\}.
\end{aligned}$$

Thus we have proved the lemma. $\square$

Lemma 4.3 shows that a worker with higher quality contributes not less in JQ compared with a lower quality worker. For the case that each worker has the same cost requirement c, i.e., $c_i = c_j = c$ for $i, j \in [1, N]$, we can select the top-k workers sorted by their quality in decreasing order, where $k = \min\left\{\left\lfloor \frac{B}{c} \right\rfloor, N\right\}$.

Although the above two properties can indicate us to solve JSP under certain conditions, the case for JSP with arbitrary individual cost is much more complicated as we have to consider not only the worker j_i's quality q_i, but also her cost c_i, and both may vary between different workers.

We can formally prove JSP is NP-hard in Theorem 4.4. Note that JSP, in general, is NP-hard due to the fact that it cannot avoid computing $JQ(J, BV, \alpha)$ at each step, which is an NP-hard problem itself. Moreover, even if we assume access to a polynomial oracle for computing $JQ(J, BV, \alpha)$, e.g., Algorithm 4, the problem still remains NP-hard, by following the reduction to a n-th order Knapsack Problem [33].

Theorem 4.4. *Solving JSP is NP-hard.*

Proof. For ease of presentation, we denote $X = [x_1, x_2, \dots, x_N]$ where $x_i = 1$ or 0 indicates that worker j_i is selected or not. And given α and W, we denote $F(X, W) = JQ(J, BV, \alpha)$ where $J = \{j_i \mid x_i = 1, 1 \le i \le N\}$.

We have proved in Theorem 4.2 that given α and J, computing $JQ(J, BV, \alpha)$ is NP-hard, which means that given a constant $K \in [0, 1]$, a known α and J, verifying the problem $JQ(J, BV, \alpha) \ge K$ (or $K \ge JQ(J, BV, \alpha)$) is NP-hard. Since if not, we can use bisection method to tune K in order to get the exact value of $JQ(J, BV, \alpha)$ in PTIME.[14]

We next prove that JSP is NP-hard by constructing a special case, that is given α, $J = \{j_1, j_2, \dots, j_n\}$ and a constant $K \in [0, 1]$, we reduce the problem of verifying $JQ(J, BV, \alpha) \ge K$ (or $K \ge JQ(J, BV, \alpha)$) to solving JSP.

For a known $K \in [0, 1]$, we construct a multi-set $W' = J \cup \{j_{n+1}\}$, where $q'_i = q_i$ for $i \in [1, n]$ and $q'_{n+1} = K$. Let $C' = \{c'_1, c'_2, \dots, c'_{n+1}\}$ where $c'_i = 1$ for $i \in [1, n]$ and $c'_{n+1} = n$.

Then given W', C' and $B = n$, the result of JSP can be reduced to solving

$$X^* = \operatorname*{argmax}_{X \in \{X^{(1)},\, X^{(2)}\}} \{\ F(X^{(1)}, W'),\ F(X^{(2)}, W')\ \},$$

where $X^{(1)}$ and $X^{(2)}$ are
(i) $X^{(1)} = [x_1^{(1)}, x_2^{(1)}, \dots, x_{n+1}^{(1)}]$ where $x_i^{(1)} = 1$ for $i \in [1, n]$ and $x_{n+1}^{(1)} = 0$;
(ii) $X^{(2)} = [x_1^{(2)}, x_2^{(2)}, \dots, x_{n+1}^{(2)}]$ where $x_i^{(2)} = 0$ for $i \in [1, n]$ and $x_{n+1}^{(2)} = 1$.
The reason is that
(1) if $x_{n+1} = 1$, then any other $x_i = 1$ for $i \ne n+1$ will make the constraint $\sum_{i=1}^{n+1} c_i \le B = n$ invalid;
(2) if $x_{n+1} = 0$, as Lemma 4.2, we know that by setting $x_i = 1$ for $1 \le i \le n$, the highest JQ can be reached in this case. Thus to get the optimal X^*, it is required

[14]bounded by the logarithm of the numerical precision for the product of qualities in J, as the formula of computing $JQ(J, BV, \alpha)$ in Equation 4.4 shows.

compare between $F(X^{(1)}, W') = JQ(J, BV, \alpha)$ and $F(X^{(2)}, W') = K$.
Then we have
(1) $X^* = X^{(1)} \Rightarrow JQ(J, BV, \alpha) \geq K$;
(2) $X^* = X^{(2)} \Rightarrow K \geq JQ(J, BV, \alpha)$.

We know that given $K \in [0, 1]$, α and J, verifying $JQ(J, BV, \alpha) \geq K$ (or $K \geq JQ(J, BV, \alpha)$) is NP-hard, thus we have proved that solving JSP is NP-hard. $\square$

4.5.1 Heuristic Solution

To address the computational hardness issue, we use the *simulated annealing heuristic* [105], which is a stochastic local search method for discrete optimization problems. This method can escape local optima and is proved to be effective in solving a variety of computationally hard problems [28, 55, 228]. Note that there are other heuristics such as iterative improvement and evolution algorithms, however, we find that simulated annealing already works efficiently and effectively in practice.

The simulated annealing heuristic mimics the cooling process of metals, which converge to a final, "frozen" state. A temperature parameter T is used and iteratively reduced until it is small enough. For a specific value of T, the heuristic performs several local neighbourhood searches. There is an objective value on each location, and let Δ denote the difference in objective value between the searched location and the original location. For each local search, the heuristic makes a decision whether to "move" to the new location or not based on T and Δ:

1. if the move will not decrease the objective value (i.e., $\Delta \geq 0$), then the move is accepted;
2. if the move will decrease the objective value (i.e., $\Delta < 0$), the move is accepted with probability $\exp(-\frac{\Delta}{T})$, i.e., by sampling from a Boltzmann

Algorithm 7 JSP (Chapter 4).

```
Input:  $W = \{j_1, j_2, \ldots, j_N\}$, $B$, $N$
Output: $\widehat{J}$
 1: $T = 1.0$; // initial temperature parameter
 2: $X = [\, x_1 = 0, x_2 = 0, \ldots, x_N = 0 \,]$; // all initialized as 0
 3: $\widehat{J} = \emptyset$; // estimated optimal jury set $J^*$
 4: $M = 0$; // the overall monetary incentive for selected workers
 5: $H = \emptyset$; // the set containing indexes for selected workers
 6: while $T \geq \epsilon$ do
 7:   for $i = 1$ to $N$ do
 8:     randomly pick an index $r \in \{1, 2, \ldots, N\}$;
 9:     if $x_r = 0$ and $M + c_r \leq B$ then
10:       $x_r = 1$;  $M = M + c_r$;
11:       $\widehat{J} = \widehat{J} \cup \{j_r\}$;  $H = H \cup \{r\}$;
12:     else
13:       $X, M, \widehat{J}, H = \texttt{Swap}(X, M, \widehat{J}, H, r, B, N)$;
14:     end if
15:   end for
16:   $T = T/2$; // cool the temperature
17: end while
18: return $\widehat{J}$;
```

distribution [112].

The reason for not immediately rejecting the move towards a worse location is that it tries to avoid getting stuck in local optima. Intuitively, when T is large, it is freer to move than at lower T. Moreover, a large Δ restricts the move as it increases the chances of finding a very bad case.

We can apply the simulated annealing heuristic to solve JSP in Algorithm 7 by assuming that each location is a jury set $J \subseteq W$ and its objective value is $JQ(J, BV, \alpha)$. What is important in simulated annealing is the design of local search. Before introducing our design of local search, we first explain some variables to keep in Algorithm 7: H is used to store the indexes of selected workers, M is used to store their aggregated cost, and $X = [x_1, x_2, \ldots, x_N]$ is used to keep the current state of each worker ($x_i = 1$ indicates that worker j_i is selected and 0 otherwise). Starting from an initial X, we iteratively decrease

T (step 14) until T is small enough (step 6). In each iteration, we perform N local searches (steps 7-13), by randomly picking an index r out of the N worker indexes. Based on the randomly picked x_r, we either select the worker if adding the worker does not violate the budget B (steps 9-11), or execute `Swap`, which is described in Algorithm 8. The decision to swap is made based on different x_r values:

1. if $x_r = 0$, a randomly picked worker $k \in H$ is replaced with worker r if the replacement does not violate the budget constraint and the move is accepted based on Δ and T;

2. if $x_r = 1$, the algorithm performs similarly to the above case, and it replaces worker r with a randomly picked worker $k \in \{1, 2, \cdots, N\} \backslash H$ if the budget constraint still satisfies and the move is accepted as above.

While the heuristic does not have any bound on the returned jury ($\widehat{J}$) versus the optimal jury (J^*), we show in the experiments (Section 4.6) that it is close to the optimal by way of comparing the real and estimated JQ (i.e., $JQ(\widehat{J}, BV, \alpha)$ and $JQ(J^*, BV, \alpha)$).

4.6 Experiments

In this section we present the experimental evaluation of JQ and JSP, both on synthetic data and real data. For each dataset, we first evaluate the solution to JSP first, and then give detailed analysis on the computation of JQ. The algorithms were implemented in Python 2.7 and evaluated on a 16GB memory machine with Windows 7 (64 bit).

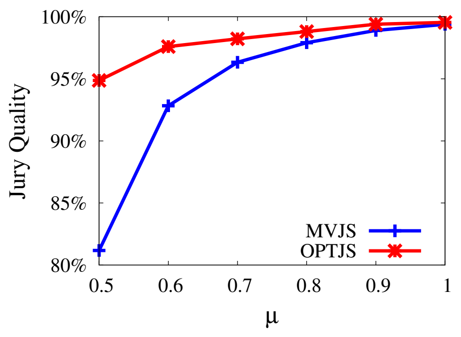
Algorithm 8 Swap (Chapter 4).

```
Input:  $X,\ M,\ \widehat{J},\ H,\ r,\ B,\ N$
Output: $X,\ M,\ \widehat{J},\ H$
1: if $x_r = 0$ then
2:     randomly pick an index $k \in H$;
3:     $a = k$ ; $b = r$ ; // store the index
4: else
5:     randomly pick an index $k \in \{1, 2, \ldots, N\} \backslash H$;
6:     $a = r$ ; $b = k$ ; // store the index
7: end if
8: if $M - c_a + c_b \leq B$ then
9:     $\Delta = \texttt{EstimateJQ}(\ \widehat{J} \setminus \{j_a\} \cup \{j_b\}\ ) - \texttt{EstimateJQ}(\widehat{J})$;
10:    if $\Delta \geq 0$ or $random(0,1) \leq \exp(-\frac{\Delta}{T})$ then
11:        $x_a = 0;\ x_b = 1;\ \ M = M - c_a + c_b;$
12:        $\widehat{J} = \widehat{J} \setminus \{j_a\} \cup \{j_b\};\ \ H = H \backslash \{a\} \cup \{b\};$
13:    end if
14: end if
15: return $X,\ M,\ \widehat{J},\ H$
```

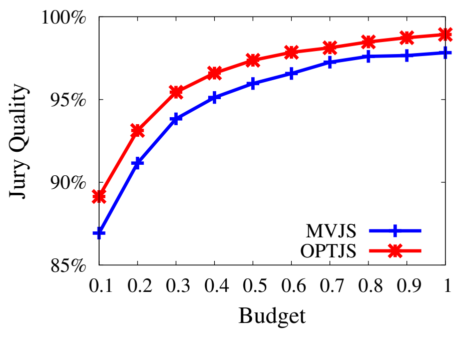
4.6.1 Synthetic Dataset

Setup

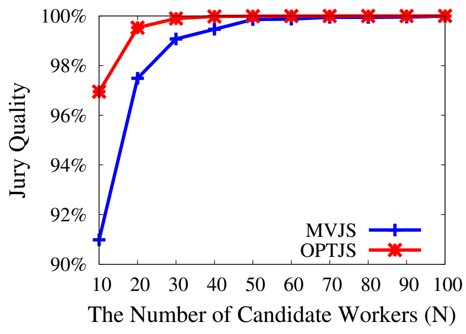
First, we describe our default settings for the experiments. Similar to the settings in [33], we generate each worker j_i's quality q_i and cost c_i via Gaussian distributions, i.e., $q_i \sim \mathcal{N}(\mu, \sigma^2)$ and $c_i \sim \mathcal{N}(\widehat{\mu}, \widehat{\sigma}^2)$. We also set parameters following [33], i.e., $\mu = 0.7$, $\sigma^2 = 0.05$, $\widehat{\mu} = 0.05$ and $\widehat{\sigma}^2 = 0.2$. By default, $B = 0.5$, $\alpha = 0.5$ and the number of candidate workers in W is $N = 50$. For JSP (Algorithm 7), we set $\epsilon = 10^{-8}$; for JQ computation (Algorithm 4), we set *numBuckets* $= 50$. To achieve statistical significance of our results, we repeat the results 1,000 times and report the average values.



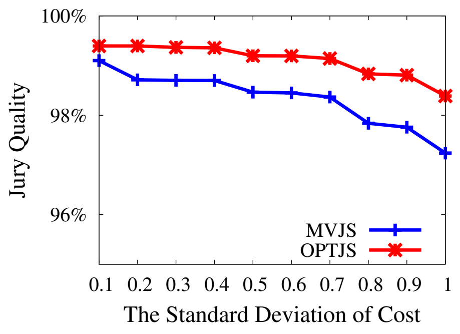
(a) Varying μ



(b) Varying B



(c) Varying N



(d) Varying $\widehat{\sigma}$

Figure 4.8: End-to-End System Comparisons.

System Comparison

We first perform the comparison of JSP with previous works, in an end-to-end system experiment. Cao et al. [33] is the only related algorithm we are aware of, which solves JSP under the MV strategy in an efficient manner. Formally, it addresses JSP as $\text{argmax}_{J\in\mathcal{C}}\, JQ(J, MV, 0.5)$. We denote their system as *MVJS* (Majority Voting Jury Selection System) and our system (Figure 4.1) as *OPTJS* (Optimal Jury Selection System). We compare the two systems by measuring the JQ on the returned jury sets.

The results are presented in Figure 4.8. We first evaluate the performance of the two systems by varying $\mu \in [0.5, 1]$ in Figure 4.8(a), which shows that *OPTJS* always outperforms *MVJS*, and *OPTJS* is more robust with low-quality workers. For example, when $\mu = 0.6$, the JQ of *OPTJS* leads that of *MVJS* for 5%. By fixing $\mu = 0.7$, Figure 4.8(b)-(d) respectively vary $B \in [0.1, 1]$, $N \in [10, 100]$, $\widehat{\sigma} \in [0.1, 1]$ and compare the performance of *MVJS* and *OPTJS*, which all show that *OPTJS* consistently performs better than *MVJS*. In Figure 4.8(b), *OPTJS* on average leads around 3% compared with *MVJS* for different B; in Figure 4.8(c), *OPTJS* is better than *MVJS*, especially when the number of workers is limited (say when $n = 10$, *OPTJS* leads *MVJS* for more than 6%); in Figure 4.8(d), compared with *MVJS*, *OPTJS* is more robust with the change of $\widehat{\sigma}$.

In summary, *OPTJS* always outperforms *MVJS* and, moreover, it is more robust with (1) lower-quality workers, (2) limited number of workers and (3) different cost variances.

Evaluating *OPTJS*

Next, we test the approximation error of Algorithm 7 by fixing $N = 11$ and varying $B \in [0.05, 0.5]$. Because of its NP-hardness, J^* is obtained by enumerating all feasible juries. We record the optimal $JQ(J^*, BV, 0.5)$ and the returned $JQ(\widehat{J}, BV, 0.5)$ in Figure 4.9(a). It shows that the two curves almost coincide with each other. As mentioned in Section 4.6.1, each point in the graph is averaged over repeated experiments. Thus, we also give statistics of the difference $JQ(J^*, BV, 0.5) - JQ(\widehat{J}, BV, 0.5)$ on all the 10,000 experiments considering different B (B changes in $[0.05, 0.5]$ with step size 0.05) in Table 4.3, which shows that more than 90% of them have a difference less than 0.01% and the maximum error is within 3%.

Our next experiment is to test the efficiency of Algorithm 7. We set $B = 0.5$ and vary $N \in [100, 500]$. The results are shown in Figure 4.9(b). We observe that the running time increases linearly with N, and it is less that 2.5 seconds even for high numbers of workers ($N = 500$). It is fairly acceptable in real situations as the JSP can be done offline.

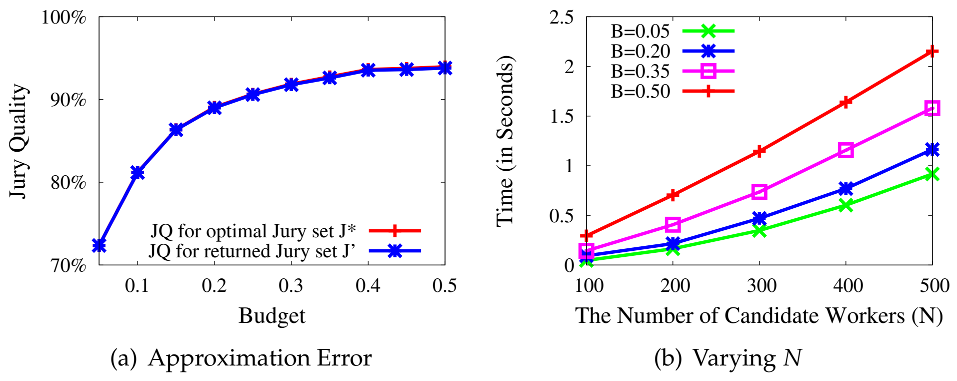


Figure 4.9: Evaluating Efficiency and Effectiveness of *OPTJS*.

Table 4.3: Statistics in Different Error Ranges.

%	$[0, 0.01]$	$(0.01, 0.1]$	$(0.1, 1]$	$(1, 3]$	$(3, +\infty)$
Counts	9301	231	408	60	0

JQ Computation

We now turn our attention to the computation of JQ, which is an essential part of *OPTJS*. We denote here by n the jury size.

We first evaluate the optimality of BV with respect to JQ. Due to the fact the computing JQ in general is NP-hard, we set $n = 11$ and evaluate JQ for four different strategies: two deterministic ones (MV-Majority Voting, and BV-Bayesian Voting), and two randomized ones (RBV-Random Ballot Voting[15] and RMV-Randomized Majority Voting). We vary $\mu \in [0.5, 1]$ and illustrate the resulting JQ in Figure 4.10(a). It can be seen that the JQ for BV outperforms the others. Moreover, unsurprisingly, all strategies have their worst performance for $\mu = 0.5$ as the workers are purely random in that case. But when $\mu = 0.5$, BV also performs robust (with JQ 93.3%), the reason is that other strategies are sensitive to low-quality workers, while BV can wisely decides the result by leveraging the workers' qualities. Finally, the randomized version of MV, i.e., RMV,

[15]RBV randomly returns 0 or 1 with 50%.

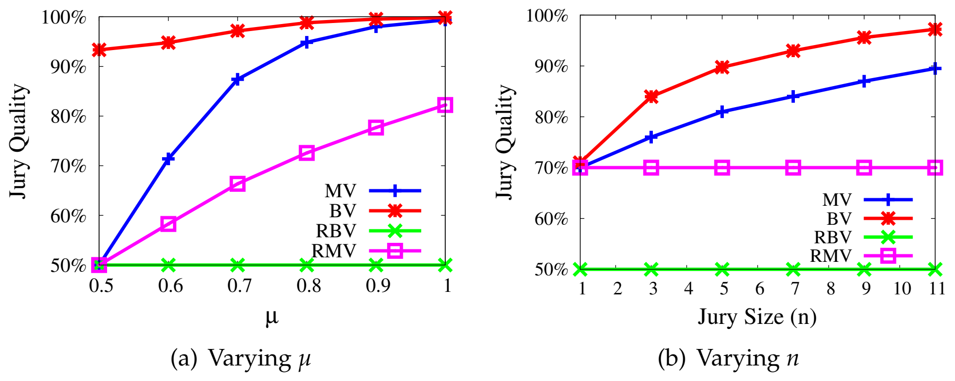


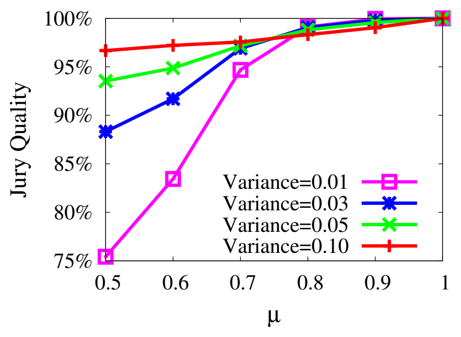
Figure 4.10: JQ for Different Strategies.

performs not better than MV for $\mu \geq 0.5$, as randomized strategies may improve the error bound in the worst case [123]. The JQ under RBV always keeps at 50% since it is purely random.

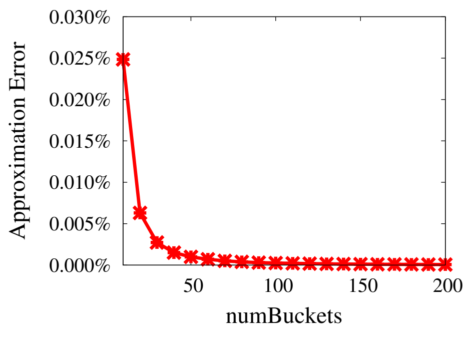
To further evaluate the performance of different strategies for different jury sizes, and for a fixed $\mu = 0.7$, we vary $n \in [1, 11]$ and plot the resulting qualities in Figure 4.10(b). The results show that as n increases, the JQ for the two randomized strategies stay the same and BV is the highest among all strategies. To be specific, when $n = 7$, the BV is about 10% better than MV. In summary, BV performs the best among all strategies.

Having compared the JQ between different strategies, we now focus on addressing the computation of JQ for BV, i.e., $JQ(J, BV, 0.5)$ in Figure 4.11. We first evaluate the effect of the quality variance σ^2 with varying mean μ in Figure 4.11(a). It can be seen that JQ has the highest value for a high variance when $\mu = 0.5$. It's because under a higher variance, worker qualities are more likely to deviate from the mean (0.5), and so, it's likely to have more high-quality workers.

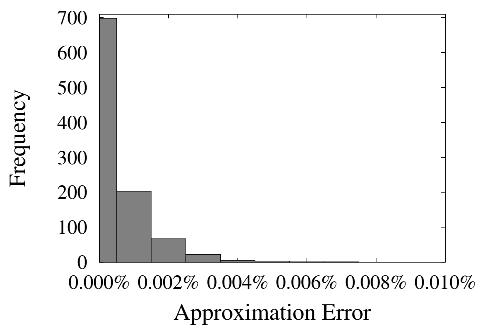
Then we address the effectiveness of Algorithm 4 for approximating the real JQ. We first evaluate the approximation error in Figure 4.11(b) by varying *numBuckets* $\in [10, 200]$. As can be seen, the approximation error drops sig-



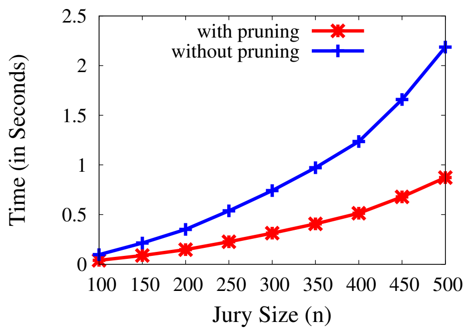
(a) Varying μ and σ^2



(b) Varying $numBuckets$



(c) Approximation Error



(d) Varying n

Figure 4.11: Evaluating $JQ(J, BV, 0.5)$ Computation.

nificantly with $numBuckets$, and is very close to 0 if we have enough buckets. In Figure 4.11(c) we plot the histogram of differences between the accurate JQ and the approximated JQ (or $JQ - \widehat{JQ}$) over all repeated experiments by setting $numBuckets = 50$. It is heavily skewed towards very low errors. In fact, the maximal error is within 0.01%.

Finally, we evaluate the computational savings of the pruning techniques of Algorithm 4 by varying the number of workers $n \in [100, 500]$ in Figure 4.11(d). The pruning technique is indeed effective, saving more than half the computational cost. Moreover, it can be seen that our method scales very well with the number of workers. For example, when $n = 500$, the estimation of JQ runs within 2.5s without pruning technique, while finishing within 1s facilitated by our proposed pruning methods.

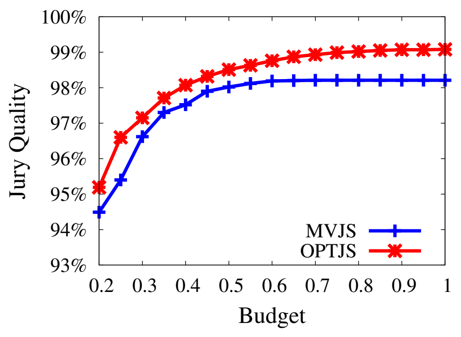
4.6.2 Real Dataset

Dataset Collection

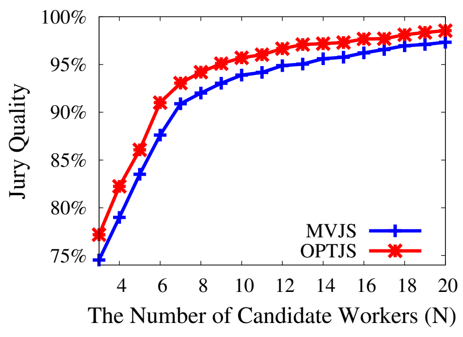
We collected the real world data from the Amazon Mechanical Turk (AMT) platform. AMT provides APIs and allows users to batch multiple questions in Human Intelligence Tasks (HIT). Each worker is rewarded with a certain amount of money upon completing a HIT. The API also allows to set the number of assignments (denoted m) to a HIT, guaranteeing it can be answered m times by different workers. To generate the HITs, we use the public sentiment analysis dataset[16], which contains 5,152 tweets related to various companies. We randomly select 600 tweets from them, and generate a HIT for each tweet, which asks whether the sentiment of a tweet is positive or not (decision making task). The ground truth of this question is provided by the dataset. The true answers for *yes* and *no* is approximately equal, so we set the prior as $\alpha = 0.5$.

To perform experiments on AMT, we randomly batch 20 questions in a HIT and set $m = 20$ for each HIT, where each HIT is rewarded \$0.02. After all HITs are finished, we collect a dataset which contains 600 decision-making tasks, and each task is answered by 20 different workers. We give several statistics on the worker answering information. There are 128 workers in total, and each of them has answered on average $\frac{600\times 20}{128} = 93.75$ questions. Only two workers have answered all questions and 67 workers have answered only 20 questions. We used these answers to compute every worker's quality, which is defined as the proportion of correctly answered questions by the worker in all her answered questions. The average quality for all workers is 0.71. There are 40 workers whose qualities are greater than 0.8, and about 10% whose quality is less than the value 0.6.

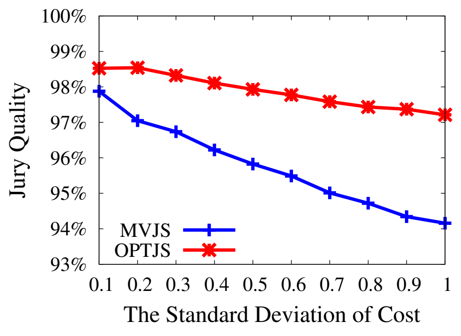
[16]http://www.sananalytics.com/lab/twitter-sentiment/



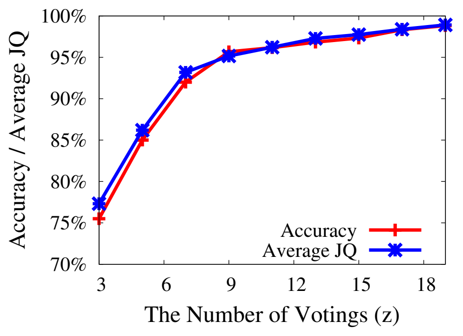
(a) Varying B



(b) Varying N



(c) Varying $\widehat{\sigma}$



(d) Is JQ a good prediction?

Figure 4.12: Real Dataset Evaluation.

JSP

To evaluate JSP, for each question, we form the candidate workers set W by collecting all 20 workers who answered the question, i.e., having $N = |W| = 20$. We follow the settings in experiments on synthetic data except that worker qualities are computed using the real-world data. We then solve JSP for each question by varying $B \in [0.1, 1.0]$, $N \in [3, 20]$ and $\widehat{\sigma} \in [0, 1]$. We compute the average returned JQ by solving JSP for all 600 questions, which is recorded as a point in Figures 4.12(a)-(c), respectively. It can be seen that Figure 4.12(a)-(c) has a similar results pattern as Figure 4.8(b)-(d), i.e., experimental results on the synthetic datasets. Especially, *OPTJS* always outperforms *MVJS* in real-world scenarios.

Is JQ is a good prediction?

Finally, we try to evaluate whether JQ, defined in Definition 4.3, is a good way to predict the quality for BV in reality. Notice that, after workers give their votes, we can adopt BV to get the voting result, and then compare it with the true answer of the question. And thus, the goodness of BV in reality can be measured by the "accuracy", which counts the proportion of correctly answered questions according to BV.

We now test whether JQ is a good prediction of accuracy in reality. For each question, we vary the number of votes (denoted as z). For a given $z \in [0, 20]$, based on the question's answering sequence, we collect its first z votes, then
(i) for each question, knowing the first z workers who answered the question, we can compute the JQ by considering these workers' qualities. Then we take the average of JQ among all 600 questions;
(ii) by considering the first z workers' qualities who answered the question and their votes, BV can decide the result of the question. After that, the accuracy can be computed by comparing voting result and the true answer for each question.

Now given a $z \in [3, 20]$, we compare the average JQ and accuracy in Figure 4.12(d), which shows that they are highly similar. Hence, it verifies that JQ for BV is really a good prediction of accuracy for BV in reality.

4.7 Extensions to Various Task Types and Worker Models

In Section 4.4 we have outlined the algorithm for estimating JQ only for decision making tasks in which only two answers are possible and the quality of a worker is modeled as a constant parameter, encoding the probability of a correct answer. But in real-world scenarios the task and worker model can be more complex.

In the case of the task itself, a more natural way is to model the answers as multiple choices, i.e., multiple labels. For example, sentiment analysis tasks [127] have the objective of detecting the correct sentiment for a piece of text (crawled from tweets) and the choices for each task are *positive*, *neutral*, and *negative*. Here we assume that each task has ℓ possible answers/labels, denoted as $\{0, 1, \ldots, \ell - 1\}$ and it has only one true label $\mathbf{t} \in [0, \ell - 1]$. Note that for the case that each task can have multiple true labels, we can follow [149], which decomposes each task into ℓ decision making tasks, and publish these ℓ tasks to workers.

In the case of worker quality model, it can be modeled by measuring the sensitivity and specificity of each worker [215], and it can be generalized to be modeled as a *confusion matrix* (CM) [92]. A confusion matrix C is a matrix of size $\ell \times \ell$ where each element C_{jk} encodes the probability that the worker gives label k as an answer when the true label is j. An example CM with $\ell = 2$ for a worker is $C = \begin{bmatrix} 0.7 & 0.3 \\ 0.2 & 0.8 \end{bmatrix}$. The CM quality model is a very general model of worker quality, and it can also solve the case in [127], which models each worker as a constant parameter $q \in [0, 1]$ for multiple label tasks. Many studies [19,92,215] have addressed how to derive CM from worker's past answering history.

In this section, we extend our strategies and JQ computation algorithms to the setting in which a task has a number of ℓ possible labels, where the prior is a vector $\vec{\alpha} = \{\alpha_0, \alpha_1, \ldots, \alpha_{\ell-1}\}$ s.t. $\sum_{j=0}^{\ell-1} \alpha_j = 1$, and the quality of each worker j_i is modeled as a confusion matrix $C^{(i)}$. We first prove the optimal strategy in Section 4.7.1, and then extend the JQ and JSP respectively in Section 4.7.2 and 4.7.3.

4.7.1 Optimal Strategy Extension

To derive the optimal strategy for multiple-choice tasks, note that here $S(\cdot)$ is a function which takes V $(\in \Omega = \{0, 1, \ldots, \ell - 1\}^n)$, jury set J (where each worker j_i is modeled as a CM $C^{(i)}$) and prior $\vec{\alpha}$ as input, the output is the estimated true answer $S(V, J, \vec{\alpha}) \in [0, \ell - 1]$. For simplicity we denote $S(V)$ as

$S(V, J, \vec{\alpha})$. Similar to Equation 4.4, here $\mathbb{E}[\mathbb{1}_{\{S(\mathbf{V})=\mathbf{t}\}}]$ can be expressed as:

$$\begin{aligned} &\textstyle\sum_{V\in\Omega}\ \sum_{t=0}^{\ell-1}\ \Pr(\ \mathbf{t}=t\ )\cdot\Pr(\ V\mid\mathbf{t}=t\ )\cdot\mathbb{E}[\ \mathbb{1}_{\{S(V)=t\}}\ ] \\ =&\textstyle\sum_{V\in\Omega}\ \sum_{t=0}^{\ell-1}\ \alpha_t\cdot(\ \prod_{i=1}^{n} C_{t,v_i}^{(i)}\ )\cdot\mathbb{E}[\ \mathbb{1}_{\{S(V)=t\}}\ ] \end{aligned}$$

Similarly we can derive that the optimal strategy $S^*(V)$ is

$$S^*(V) = \underset{t\in\{0,1,\ldots,\ell-1\}}{\operatorname{argmax}}\ \alpha_t\cdot\textstyle\prod_{i=1}^{n} C_{t,v_i}^{(i)}. \tag{4.17}$$

Note that if multiple labels can reach the highest probability, we set $S^*(V)$ as the label with the lowest index among those labels. We can see this is a optimal strategy by the following reasoning:
(1) if there exists a deterministic strategy S' such that, for a specific V, it satisfies $S'(V) = m \in [0, \ell-1]$ while

$$\begin{aligned} &\Pr(\ \mathbf{t}=m\ )\cdot\Pr(\ V\mid\mathbf{t}=m\ ) \\ <&\max_{t\in[0,\ell-1]}\Pr(\ \mathbf{t}=t\ )\cdot\Pr(\ V\mid\mathbf{t}=t\ ) \\ =&\Pr(\ \mathbf{t}=S^*(V)\ )\cdot\Pr(\ V\mid\mathbf{t}=S^*(V)\ ), \end{aligned}$$

then we can change the value of $S'(V)$ to $S'(V) = S^*(V)$, which increases the JQ for the strategy S';
(2) if there exists a randomized strategy S' such that, for a specific V, it satisfies $S'(V) = S^*(V)$ with probability $p < 1$, then we can set $S'(V) = S^*(V)$ with probability 1 and $S'(V) = m$ (where $m \neq S^*(V)$) with probability 0, then the change will increase the JQ for the strategy S'.

Thus we can prove the optimal strategy S^*, which is the same as Bayesian Voting strategy BV, or $S^* = BV$.

4.7.2 JQ Computation Extension

Compared with Algorithm 4, the case of multiple label tasks where each

No.	*V*	P(**V**=*V* | **t**=0)*P(**t**=0)	BV: [compare] (result) [√/×]
1	(0,0)	0.7*0.4*1/3=0.0933	[0.0933 , 0.0067 , 0.0067] (0) [√]
2	(0,1)	0.7*0.3*1/3=0.07	[0.07 , 0.02 , 0.0033] (0) [√]
3	(0,2)	0.7*0.3*1/3=0.07	[0.07 , 0.0067 , 0.0233] (0) [√]
4	(1,0)	0.2*0.4*1/3=0.0267	[0.0267 , 0.0533 , 0.0233] (1) [×]
5	(1,1)	0.2*0.3*1/3=0.02	[0.02 , 0.016 , 0.01] (0) [√]
6	(1,2)	0.2*0.3*1/3=0.02	[0.02 , 0.0533 , 0.07] (2) [×]
7	(2,0)	0.1*0.4*1/3=0.0133	[0.0133 , 0.0067 , 0.04] (2) [×]
8	(2,1)	0.1*0.3*1/3=0.01	[0.01 , 0.02 , 0.02] (1) [×]
9	(2,2)	0.1*0.3*1/3=0.01	[0.01 , 0.0067 , 0.14] (2) [×]

(a) $\mathbf{t} = 0$, $\alpha_0 = 1/3$

No.	*V*	P(**V**=*V* | **t**=1)*P(**t**=1)	BV: [compare] (result) [√/×]
1	(0,0)	0.1*0.2*1/3=0.0067	[0.0933 , 0.0067 , 0.0067] (0) [×]
2	(0,1)	0.1*0.6*1/3=0.02	[0.07 , 0.02 , 0.0033] (0) [×]
3	(0,2)	0.1*0.2*1/3=0.0067	[0.07 , 0.0067 , 0.0233] (0) [×]
4	(1,0)	0.8*0.2*1/3=0.0533	[0.0267 , 0.0533 , 0.0233] (1) [√]
5	(1,1)	0.8*0.6*1/3=0.016	[0.02 , 0.016 , 0.01] (0) [×]
6	(1,2)	0.8*0.2*1/3=0.0533	[0.02 , 0.0533 , 0.07] (2) [×]
7	(2,0)	0.1*0.2*1/3=0.0067	[0.0133 , 0.0067 , 0.04] (2) [×]
8	(2,1)	0.1*0.6*1/3=0.02	[0.01 , 0.02 , 0.02] (1) [√]
9	(2,2)	0.1*0.2*1/3=0.0067	[0.01 , 0.0067 , 0.14] (2) [×]

(b) $\mathbf{t} = 1$, $\alpha_1 = 1/3$

No.	*V*	P(**V**=*V* | **t**=2)*P(**t**=2)	BV: [compare] (result) [√/×]
1	(0,0)	0.1*0.2*1/3=0.0067	[0.0933 , 0.0067 , 0.0067] (0) [×]
2	(0,1)	0.1*0.1*1/3=0.0033	[0.07 , 0.02 , 0.0033] (0) [×]
3	(0,2)	0.1*0.7*1/3=0.0233	[0.07 , 0.0067 , 0.0233] (0) [×]
4	(1,0)	0.3*0.2*1/3=0.02	[0.0267 , 0.0533 , 0.0233] (1) [×]
5	(1,1)	0.3*0.1*1/3=0.01	[0.02 , 0.016 , 0.01] (0) [×]
6	(1,2)	0.3*0.7*1/3=0.07	[0.02 , 0.0533 , 0.07] (2) [√]
7	(2,0)	0.6*0.2*1/3=0.04	[0.0133 , 0.0067 , 0.04] (2) [√]
8	(2,1)	0.6*0.1*1/3=0.02	[0.01 , 0.02 , 0.02] (1) [×]
9	(2,2)	0.6*0.7*1/3=0.14	[0.01 , 0.0067 , 0.14] (2) [√]

(c) $\mathbf{t} = 2$, $\alpha_2 = 1/3$

Figure 4.13: Illustrating JQ Calculation for Different **t** and **V**.

Algorithm 9 GetBucketSize (Chapter 4).

```
Input:  $J = \{j_1, j_2, \ldots, j_n\}$, $numBuckets$, $n$, $\ell$, $\vec{\alpha}$
Output: $\delta$
 1: $upper = 0$;
 2: $tmin = 1$; $tmax = 0$;
 3: for $t = 0$ to $\ell - 1$ do
 4:   if $tmin > \alpha_t$ then $tmin = \alpha_t$;
 5:   if $tmax < \alpha_t$ then $tmax = \alpha_t$;
 6: end for
 7: if $upper < \ln \frac{tmax}{tmin}$ then $upper = \ln \frac{tmax}{tmin}$; // prior
 8: for $i = 1$ to $n$ do
 9:   $tmin = 1$; $tmax = 0$;
10:   for $t = 0$ to $\ell - 1$ do
11:     for $j = 0$ to $\ell - 1$ do
12:       if $tmin > C^{(i)}_{t,j}$ then $tmin = C^{(i)}_{t,j}$;
13:       if $tmax < C^{(i)}_{t,j}$ then $tmax = C^{(i)}_{t,j}$;
14:     end for
15:     if $upper < \ln \frac{tmax}{tmin}$ then $upper = \ln \frac{tmax}{tmin}$; // CM
16:   end for
17: end for
18: $\delta = \frac{upper}{numBuckets}$;
19: return $\delta$
```

worker is modeled as a confusion matrix requires several adaptations. First let us illustrate an example below.

Example 12. *Take the example in Figure 4.13, where $\ell = 3$, $\vec{\alpha} = \{1/3,\ 1/3,\ 1/3\}$, $J = \{j_1, j_2\}$, and*

$$C^{(1)} = \begin{bmatrix} 0.7 & 0.2 & 0.1 \\ 0.1 & 0.8 & 0.1 \\ 0.1 & 0.3 & 0.6 \end{bmatrix}, C^{(2)} = \begin{bmatrix} 0.4 & 0.3 & 0.3 \\ 0.2 & 0.6 & 0.2 \\ 0.2 & 0.1 & 0.7 \end{bmatrix}.$$

We have to enumerate $\ell^n = 3^2 = 9$ different configurations of the possible votings $\mathbf{V} \in \{0,1,2\}^2$ by varying different the true labels $\mathbf{t} \in \{0,1,2\}$ from Figure 4.13(a)-(c). Take the voting $V = \{0,1\}$ as an example in Figure 4.13(a), which assumes $\mathbf{t} = 0$. The probability is $\Pr(V \mid \mathbf{t} = 0) \cdot \Pr(\mathbf{t} = 0) = C^{(1)}_{00} \cdot C^{(2)}_{01} \cdot \alpha_0 = 0.07$. By following the *optimal strategy outlined above, we should compare the probability* $\alpha_0 \cdot \Pr(V \mid \mathbf{t} = 0)$, $\alpha_1 \cdot \Pr(V \mid \mathbf{t} = 1)$, *and* $\alpha_2 \cdot \Pr(V \mid \mathbf{t} = 2)$. *As shown in the fourth column of the label, it returns label* 0, *since it corresponds to the highest probability. As* $S^*(V) = 0 = \mathbf{t}$, *so the probability* 0.07 *will be added to JQ.*

Based on the example in Figure 4.13 illustrated above, dealing with multiple label task while modeling each worker as a CM is different from that in Figure 4.2, which deals with decision making task and it models each worker as a quality constant. We summarize two main challenges and our solutions in the following:
(1) As in Figure 4.2, it models each worker a constant quality value, assuming independence between different labels, i.e., $\Pr(v_i = 0 \mid \mathbf{t} = 0) = \Pr(v_i = 1 \mid \mathbf{t} = 1)$. Different from Figure 4.2, in Figure 4.13 the CM models the dependency between the various possible labels, and because of the fact that we can not find the correspondence between different tables in Figure 4.13, we should treat each table (different $\mathbf{t}$) respectively. So we perform ℓ iterations, where each iteration consider a specific $\mathbf{t} \in [0, \ell - 1]$;
(2) as for each voting V, the BV should select

$$BV(V) = \operatorname*{argmax}_{t \in [0, \ell-1]} \alpha_t \cdot \Pr(V \mid \mathbf{t} = t),$$

which is the comparison between ℓ probabilities, thus rather than a constant value, given a fixed $t' \in \{0, 1, \ldots, \ell - 1\}$, we keep an ℓ-tuple

$$\left(\ln \frac{\Pr(V \mid \mathbf{t} = t') \cdot \alpha_{t'}}{\Pr(V \mid \mathbf{t} = 0) \cdot \alpha_0}, \; \ldots \; , \; \ln \frac{\Pr(V \mid \mathbf{t} = t') \cdot \alpha_{t'}}{\Pr(V \mid \mathbf{t} = \ell - 1) \cdot \alpha_{\ell-1}} \right)$$

as the *key* in the map structure, and the corresponding *prob* is the aggregated probabilities of the same *key*. Then the *prob* can be added in $\widehat{JQ}$ if all components in the tuple are positive.

Algorithm 10 illustrates the JQ estimation for multiple-label tasks. As discussed above, we take ℓ iterations to consider each possible $\mathbf{t}$ respectively, where

Algorithm 10 EstimateGeneralJQ (Chapter 4).

```
Input:  $J = \{j_1, j_2, \ldots, j_n\}$, $numBuckets$, $n$, $\ell$, $\vec{\alpha}$
Output: $\widehat{JQ}$
 1: $\delta =$ GetBucketSize($Q, numBuckets, n, \ell, \vec{\alpha}$);
 2: $\widehat{JQ} = 0$;
 3: for $t = 0$ to $\ell - 1$ do
 4:   $SM = map()$;
 5:   $key = (\ 0, 0, \ldots, 0\ )$; // initialize an $\ell$-tuple with all 0
 6:   for $j = 0$ to $\ell - 1$ do
 7:     $key[\ j+1\ ] = \left\lceil \frac{\ln(\alpha_t/\alpha_j)}{\delta} - \frac{1}{2} \right\rceil$; // incorporate prior
 8:   end for
 9:   $SM[\ key\ ] = 1$; // the $key$ contains prior information
10:   for $i = 1$ to $n$ do
11:     $M = map()$; // initialize an empty map structure
12:     for $(key, prob) \in SM$ do
13:       for $v = 0$ to $\ell - 1$ do
14:         $newkey = key$; // copy from $key$ to $newkey$
15:         for $jj = 0$ to $\ell - 1$ do
16:           $newkey[\ jj+1\ ] += \left\lceil \frac{\ln(C^{(i)}_{t,v}/C^{(i)}_{jj,v})}{\delta} - \frac{1}{2} \right\rceil$;
17:         end for
18:         if $newkey \notin M$ then
19:           $M[\ newkey\ ] = 0$;
20:         end if
21:         $M[\ newkey\ ] = M[\ newkey\ ] + prob \cdot C^{(i)}_{t,v}$;
22:       end for
23:     end for
24:     $SM = M$;
25:   end for
26:   $\widehat{JQsub} = 0$;
27:   for $(key, prob) \in SM$ do
28:     $count = 0$;
29:     $flag =$ true;
30:     for $j = 0$ to $\ell - 1$ do
31:       if $key[\ j+1\ ] < 0$ then
32:         $flag =$ false;
33:         break;
34:       else if $key[\ j+1\ ] == 0$ then
35:         $count = count + 1$;
36:       end if
37:     end for
38:     if $flag =$ true then
39:       $\widehat{JQsub} += prob/count$;
40:     end if
41:   end for
42:   $\widehat{JQ} = \widehat{JQ} + \widehat{JQsub} \cdot \alpha_t$; // prior
43: end for
44: return $\widehat{JQ}$;
```

for each $t \in [0, \ell - 1]$, it first generates an initial tuple by incorporating the prior information (step 5-8), and then iterates over the n workers. For each worker j_i, it generates a new map structure M based on stored map structure SM for the previous iteration (dealing with worker j_{i-1}). The way to form M (step 10-19) is that for each $(key, prob)$ pair in SM, it generates ℓ tuples (each tuple is of size ℓ) by considering different votes from worker j_i, and for each possible vote, it updates ℓ components in the *newkey* and updates *prob* in the corresponding *newkey* (step 12-18). After all workers have been iterated, it starts to deal with the map structure for the last iteration (step 20-31). The corresponding *prob* value for the *key* is aggregated if all the elements in the tuple are positive. In the case when there are multiple components with value 0 in the *key*, as only the lowest label index will be returned (Section 4.7.1), so only $1/count$ of the *prob* should be added (step 22-31).

The method `GetBucketSize` is tasked with computing the value of δ, by considering the prior and the CM of each worker. An important parameter in the computation of $\delta = \frac{upper}{numBuckets}$, and the value $upper$ is computed in Algorithm 9, which is the same as follows:

$$\max \left\{ \ln \frac{\max_{t \in [0,\ell-1]} \ \alpha_t}{\min_{t \in [0,\ell-1]} \ \alpha_t}, \ \max_{\substack{i \in [1,n] \\ j \in [0,\ell-1]}} \ \ln \frac{\max_{t \in [0,\ell-1]} \ C_{t,j}^{(i)}}{\min_{tt \in [0,\ell-1]} \ C_{tt,j}^{(i)}} \right\}.$$

Note that as we can easily deal with the case where $\alpha_t = 0$ for some $t \in [0, \ell - 1]$ and $C_{t,j}^{(i)} = 0$ for some $i \in [1, n]$, $t, j \in [0, \ell - 1]$, so we do not have to include them in computing $upper$, which will result in overflow.

Algorithm complexity. Similar to the analysis in Algorithm 4, suppose $numBuckets = d \cdot n$. We know that each component in the *key* has at most $2dn^2 + 1$ possible values, and there are $\mathcal{O}(d^\ell \cdot n^{2\ell})$ possible tuples, making the time complexity of Algorithm 10 to be $\mathcal{O}(\ell \cdot d^\ell \cdot n^{2\ell+1})$ by considering ℓ iterations for different $\mathbf{t}$. Note that even though this is unfortunately exponential in the number of labels (ℓ), in real settings, however, the number of labels is a small constant (as otherwise it might confuse the workers), so the algorithm is still polynomial for all practical purposes. Moreover, solving JSP and computing JQ is usually and offline process, without stringent efficiency requirements.

4.7.3 JSP extension

In this section, we extend JSP to support for CM and multiple label tasks. Recall that in Section 4.5 we first study two Lemmas (Lemma 4.2 and 4.3), which provide some good properties of JQ, resulting in straightforward solution under specific cost models, then we adapt Simulated Annealing Heuristic to deal with more general cost model for JSP. In this section, we first similarly study whether the two lemmas satisfy or not for more general task and worker models, and then discuss how to solve the general JSP.

We first prove an extension for Lemma 4.3, that is, the monotonicity property on jury size holds even for more general task and worker model:

Lemma 4.4 (Extensions for Lemma 4.2). *Given $\vec{\alpha}$ and J, $JQ(J, BV, \vec{\alpha}) \leq JQ(J', BV, \vec{\alpha})$ where $J' = J \cup \{j_{n+1}\}$.*

Proof. Similar to the proof in Lemma 4.2, based on the prior $\vec{\alpha} = \{\alpha_0, \alpha_1 \cdots \alpha_{\ell-1}\}$, for a specific $V \in \Omega$, we denote

$$A_t(V) = \alpha_t \cdot \Pr(V \mid \mathbf{t} = t) \cdot \mathbb{E}[\mathbb{1}_{\{BV(V)=t\}}]$$

for $t = 0, 1, \ldots, \ell - 1$, and based on Equation 4.17, we can prove that

$$\textstyle\sum_{t=0}^{\ell-1} A_t(V) = \max_{t \in [0, \ell-1]} \{ \alpha_t \cdot P(V \mid \mathbf{t} = t) \}. \tag{4.18}$$

By adding a worker j_{n+1} with CM $C^{(n+1)}$ (size $\ell \times \ell$), the voting $V \in \Omega$ becomes ℓ votes, where the j-th one is denoted as $V^{(j)} = \{ v_0^{(j)}, v_1^{(j)}, \ldots, v_{n+1}^{(j)} \}$ s.t. $v_i^{(j)} = v_i$

for $i \in [1, n]$ and $v_{n+1}^{(j)} = j$. So Equation 4.18 becomes

$$\begin{aligned}
&\sum_{j=0}^{\ell-1}\sum_{t=0}^{\ell-1} A_t(V^{(j)}) \\
=&\sum_{j=0}^{\ell-1} \max_{t\in[0,\ell-1]} \{ \alpha_t \cdot P(V^{(j)} \mid \mathbf{t}=t) \} \\
=&\sum_{j=0}^{\ell-1} \max_{t\in[0,\ell-1]} \{ \alpha_t \cdot C_{t,j}^{(n+1)} \cdot P(V \mid \mathbf{t}=t) \}.
\end{aligned} \tag{4.19}$$

By denoting $b_t = \alpha_t \cdot P(V|\mathbf{t} = t)$ for $t \in [0, \ell - 1]$, if we can prove Equation 4.19 is not lower than Equation 4.18, i.e.,

$$\sum_{j=0}^{\ell-1} \max_{t\in[0,\ell-1]} \{ b_t \cdot C_{t,j}^{(n+1)} \} \geq \max_{t\in[0,\ell-1]} \{ b_t \}, \tag{4.20}$$

then we can prove the theorem by considering all $V \in \Omega$. The proof is straightforward, as for any $t' \in [0, \ell - 1]$, we can prove that

$$\sum_{j=0}^{\ell-1} \max_{t\in[0,\ell-1]} \{ b_t \cdot C_{t,j}^{(n+1)} \} \geq \sum_{j=0}^{\ell-1} b_{t'} \cdot C_{t',j}^{(n+1)} = b_{t'}.$$

By extending to any $t' \in [0, \ell - 1]$, we can finally prove Equation 4.20, thus proving the Lemma. □

Lemma 4.4 tells us that even for more general task and worker model, the principle *"the more workers, the better JQ for BV"* still holds. Hence, we can select all workers if each worker contributes voluntarily or the budget is enough to select all workers (i.e., $B \geq \sum_{i=1}^{N} c_i$).

Even though the extension for Lemma 4.2 holds, the extension for Lemma 4.3, i.e., detecting what kind of CM will contribute more to JQ remains an open question. Previous research [92, 160] has addressed how to rank workers (or to detect spammers in all workers) based on their associated Confusion Matrices. The basic idea of [92, 160] is to see whether a worker's vote is indicative on the true label or not.

To implement the idea, [92] transforms the "hard" label by a worker to a "soft" label distribution. To be precise, if a worker having CM C votes a label $j \in \{0, 1, \ldots, \ell-1\}$, then the soft label distribution (denoted as $p^{(j)}$), which encodes the probabilities of each label to be the true label can be derived by a direct application of Bayes' theorem [23], as follows:

$$p^{(j)} = \Big[\ \frac{\alpha_0 \cdot C_{0,j}}{\sum_{t=0}^{\ell-1} \alpha_t \cdot C_{t,j}}, \frac{\alpha_1 \cdot C_{1,j}}{\sum_{t=0}^{\ell-1} \alpha_t \cdot C_{t,j}}, \ldots, \frac{\alpha_{\ell-1} \cdot C_{\ell-1,j}}{\sum_{t=0}^{\ell-1} \alpha_t \cdot C_{t,j}}\ \Big].$$

Note that we use $p_k^{(j)}$ to visit the k-th component (i.e., $\frac{\alpha_{k-1} \cdot C_{k-1,j}}{\sum_{t=0}^{\ell-1} \alpha_t \cdot C_{t,j}}$) in the distribution. The more concentrated this distribution $p^{(j)}$ is, the more indicative of the true label based on the worker's vote for label j is, the "better" the worker's vote for the label j is. Based on this intuition, [92] define a penalty function of a distribution $p^{(j)}$, as

$$penalty(p^{(j)}) = \sum_{u=0}^{\ell-1} \sum_{v=0}^{\ell-1} p_{u+1}^{(j)} \cdot p_{v+1}^{(j)} \cdot \mathbb{1}_{\{u \neq v\}},$$

and formally define each worker's score by aggregating $penalty(p^{(j)})$ for $j \in [0, \ell-1]$ by considering the prior $\vec{\alpha}$:

$$score(\vec{\alpha}, C) = -\sum_{j=0}^{\ell-1} \alpha_j \cdot penalty(p^{(j)}). \tag{4.21}$$

Based on the known $\vec{\alpha}$, we can rank the workers in decreasing order of their respective $score(\vec{\alpha}, C)$, and for the case that each worker requires the same cost (i.e., $c_i = c_j = c$ for $i, j \in [1, n]$), we can heuristically select the first k workers with highest *score* where $k = \min\{\lfloor \frac{B}{c} \rfloor, N\}$.

The study in [160] does not consider the label priors and defines the penalty function for a worker's vote j by aggregating the square difference for pairwise combination of the probabilities in $C_{*,j}$ (which is the $(j-1)$-th column in C), i.e.,

$$diff(C, j) = \sum_{u=0}^{\ell-1} \sum_{v=u}^{\ell-1} (C_{u,j} - C_{v,j})^2,$$

and the score for a worker with CM C is defined by aggregating over all labels:

$$score(C) = \sum_{j=0}^{\ell-1} diff(C, j). \tag{4.22}$$

Then for this heuristic, we can similarly select the top k workers with highest score if each worker's cost is the same.

For more general cost models where each worker may require arbitrary cost, we can easily generalize Algorithm 7 for the JSP extension, since it calls the JQ estimation function as a black box, and we can simply replace `EstimateGeneralJQ` with `EstimateJQ` in Algorithm 7.

4.8 Related Works

Since we have reviewed most of the related works of crowdsourcing in Chapter 2, this section only highlights the part related to expert team formation.

In social network, several works [68, 113] studied the problem of expert team formation, that is, given the aggregated skill requirements for a task, how to find a team of experts with minimum cost (communication cost or individual financial requirement), such that the skill requirements are satisfied. Rather than the skill requirements in [68,113], we focus on the probability of drawing a correct answer, which requires to enumerate exponential number of possibilities and is indeed challenging. In fact we address the Jury Selection Problem, which is firstly proposed by [33]. But we find that the solution is sub-optimal in [33], which cannot leverage the known quality for workers. We formally address the optimal JSP problem in the chapter. Some other works [54, 162] also talk about how to wisely select sources for integration. The difference is that we assume the workers are given a multiple-label task and the worker model is known, while in their problem setting, the possible answers from different sources are not restricted, and the sources' exact real qualities are unknown in advance.

4.9 Chapter Summary

In this chapter, we have studied the task assignment problem in the worker-based setting, i.e., Jury Selection Problem (JSP). To be specific, we focus on decision-making tasks, whose objective is to choose a subset of workers, such that the probability of having a correct answer (or Jury Quality, JQ) is maximized. We approach this problem from an optimality perspective. As JQ is related to voting strategy, we prove that an existing strategy, called Bayesian Voting Strategy (BV) is optimal under the JQ. Although computing JQ under BV is NP-hard, we give an efficient algorithm with theoretical guarantees. Moreover, we incorporate the task provider prior information, and we show how to extend JQ computation for different worker models and task types. Finally we evaluate JSP under BV, we prove several properties which can be used for efficient JSP computations under some constraints, and provide an approximate solution to JSP by simulated annealing heuristics.

Having discussed the solutions to task assignment problem in the above two chapters, in next chapter, we will study another important component in crowdsourcing: truth inference. To be specific, we will study the truth inference problem, i.e., how to aggregate the answers collected from workers and derive the truth of each task.

Chapter 5

Analysis of Truth Inference

5.1 Introduction

Crowdsourcing solutions have been proposed to address tasks that are hard for machines, e.g., entity resolution [30] and sentiment analysis [124]. Due to the wide deployment of public crowdsourcing platforms, e.g., Amazon Mechanical Turk (AMT) [1], CrowdFlower [5], the access to crowd becomes much easier. The database community has shown great interests in crowdsourcing (see a survey [116]). Several crowdsourced databases (e.g., CrowdDB [70], Deco [151], Qurk [137]) are built to incorporate the crowd into query processing, and there are many studies on implementing crowdsourced operators, e.g., Join [35,136,192,199], Max [80,184], Top-*k* [46,214], Group-by [46], etc.

Due to the openness of crowdsourcing, the crowd (called *workers*) may yield low-quality or even noisy answers. Thus it is important to control the quality in crowdsourcing. To address this problem, most of existing crowdsourcing studies employ a redundancy-based strategy, which assigns each task to multiple workers and aggregates the answers given by different workers to infer the correct answer (called *truth*) of each task. A fundamental problem, called *Truth Inference*, is widely studied in existing crowdsourcing works [21,32,47,48,63,100,104,118,119,126,127,131,161,182,197,200,226], which decides how to effectively infer the truth for each task.

To address the problem, a straightforward approach is Majority Voting (MV), which takes the answer given by majority workers as the truth. However, the biggest limitation of MV is that it regards all workers as equal. In reality, workers may have different levels of qualities: a high-quality worker carefully answers tasks; a low-quality (or spammer) may randomly answer tasks in order to deceive money; a malicious worker may even intentionally give wrong answers. Thus it is important to capture each worker's quality, which can better infer the truth of each task by trusting more on the answers given by workers with higher qualities.

However, the ground truth of each task is unknown and it is hard to estimate a worker's quality. To address this problem, one can label the ground truth for a small portion of tasks (called *golden tasks*) and use them to estimate workers' quality. There are two types of methods to utilize golden tasks. The first is **qualification test**. Each worker requires to perform a set of golden tasks before she can really answer tasks, and her quality is computed based on her answering performance for these golden tasks. The second is **hidden test**. The golden tasks are mixed into the tasks and the workers do not know which are golden tasks. A worker's quality is computed based on her answering performance on these golden tasks. However, the two approaches have some limitations. (1) For qualification test, workers require to answer these "extra" tasks without pay, and many workers do not want to answer such tasks. (2) For hidden test, it is a waste to pay the "extra" tasks. (3) The two techniques may not improve the quality (see Section 5.6).

Considering these limitations, the database community [63,89,118,119,127, 131,219] and data mining community [21,47,48,100,104,126,161,182,197,200, 226] independently study this problem and propose various algorithms. However, these algorithms are not compared under the same experimental framework and it is hard for practitioners to select appropriate algorithms. To alleviate this problem, we provide a comprehensive survey on existing truth inference algorithms. We summarize them in terms of *task types*, *task modeling*, *worker modeling*, and *inference techniques*. We conduct a comprehensive comparison of 17 existing representative methods [21,47,48,100,104,118,119,126,161,182,197, 200, 226], experimentally compare them on 5 real datasets with varying sizes and task types in real crowdsourcing platforms, make a deep analysis on the experimental results, and provide extensive experimental findings.

To summarize, we make the following contributions:

• We survey 17 existing algorithms, summarize a framework (Section 5.3), and provide an in-depth analysis and summary on the 17 algorithms in different perspectives (Sections 5.4-5.5), which can help practitioners to easily grasp existing truth inference algorithms.

• We experimentally conduct a thorough comparison of these methods on 5 datasets with varying sizes, publicize our codes and datasets [157], and provide experimental findings, which give guidance for selecting appropriate methods under various scenarios (Section 5.6).

• We find that the truth inference problem is not fully solved, identify the limitations of existing algorithms, and point out several promising research directions (Section 5.7).

5.2 Problem Definition

Definition 5.1 (Task). *A task set $\mathcal{T}$ contains n tasks, i.e., $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$. For each task, workers are asked to answer it.*

Existing studies mainly focus on three types of tasks.

Decision-Making Tasks. A decision-making task has a claim and asks workers to make a decision on whether the claim is *true* (denoted as 'T') or *false* (denoted

Table 5.1: An Example Product Dataset.

ID	Product Name
r_1	*iPad Two 16GB WiFi White*
r_2	*iPad 2nd generation 16GB WiFi White*
r_3	*Apple iPhone 4 16GB White*
r_4	*iPhone 4th generation White 16GB*

Table 5.2: The Collected Workers' Answers for All Tasks.

	t_1:	t_2:	t_3:	t_4:	t_5:	t_6:
	$(r_1{=}r_2)$	$(r_1{=}r_3)$	$(r_1{=}r_4)$	$(r_2{=}r_3)$	$(r_2{=}r_4)$	$(r_3{=}r_4)$
w_1	F	T	T	F	F	F
w_2		F	F	T	T	F
w_3	T	F	F	F	F	T

as 'F'). Decision-making tasks are widely used and studied in existing crowdsourcing works [47,48,100,126,200] because of its conceptual simplicity.

Next we take entity resolution as an example, which tries to find pairs of products in Table 5.1 that refer to the same real-world entity. A straightforward way is to generate a task set $\mathcal{T} = \{(r_1{=}r_2), (r_1{=}r_3), (r_1{=}r_4), (r_2{=}r_3), (r_2{=}r_4), (r_3{=}r_4)\}$ with $n = 6$ decision-making tasks, where each task has two choices: (*true*, *false*), and asks workers to select a choice for the task. For example, t_2 (or $r_1{=}r_3$) asks whether the claim '*iPad Two 16GB WiFi White = Apple iPhone 4 16GB White*' is *true* ('T') or *false* ('F'). Tasks are then published to crowdsourcing platforms (e.g., AMT [1]) and workers' answers are collected.

Single-Choice (and Multiple-Choice) Tasks. A single-choice task contains a question and a set of candidate choices, and asks workers to select a single choice out of the candidate choices. For example, in sentiment analysis, a task asks workers to select the sentiment ('*positive*', '*neutral*', '*negative*') of a given tweet. Decision-making task is a special case of single-choice task, with two special choices ('T' and 'F'). The single-choice tasks are especially studied in [21,47,48,104,118,127,131,161,182,200,226]. A direct extension of single-choice task is multiple-choice task, where workers can select multiple choices (not only a single choice) out of a set of candidate choices. For example, in image tagging, given a set of candidate tags for an image, it asks workers to select the tags that the image contains. However, as addressed in [149, 222], a multi-label task can be easily transformed to a set of decision-making tasks, e.g., for an image tagging task (multi-label task), each transformed decision-making task asks whether or not a tag is contained in an image. Thus the methods in decision-making tasks can be directly extended to handle multiple-choice tasks.

Numeric Tasks. The numeric task asks workers to provide a value. For example, a task asks about the height of Mount Everest. Different from the tasks above, workers' inputs are numeric values, which have inherent orderings (e.g., compared with 8800m, 8845m is closer to 8848m). Existing works [118, 161] especially study such tasks by considering the inherent orderings between values.

Others. Besides the above tasks, there are other types of tasks, e.g., translate a language to another [32], or ask workers to collect data (e.g., the name of a celebrity) [70, 190]. However, it is hard to control the quality for such "open" tasks. Thus they are rarely studied in existing works [32,70,190]. In this chapter, we focus only on the above three tasks and leave other tasks for future work.

Definition 5.2 (Worker)**.** *A worker set $\mathcal{W}$ contains a set of workers, i.e., $\mathcal{W} = \{w\}$. Let $\mathcal{W}^i$ denote the set of workers that have answered task t_i and $\mathcal{T}^w$ denote the set of tasks that have been answered by worker w.*

Definition 5.3 (Answer)**.** *Each task t_i can be answered with a subset of workers in $\mathcal{W}$. Let v_i^w denote the worker w's answer for task t_i, and the set of answers $V = \{v_i^w\}$ contains the collected workers' answers for all tasks.*

Table 5.2 shows an example, with answers to $\mathcal{T}$ given by three workers $\mathcal{W} = \{w_1, w_2, w_3\}$. (The empty cell means that the worker does not answer the task.) For example, $v_4^{w_1} =$ F means worker w_1 answers t_4 (i.e., $r_2 = r_3$) with 'F', i.e., w_1 thinks that $r_2 \neq r_3$. The set of workers that answer t_1 is $\mathcal{W}^1 = \{w_1, w_3\}$, and the set of tasks answered by worker w_2 is $\mathcal{T}^{w_2} = \{t_2, t_3, t_4, t_5, t_6\}$.

Definition 5.4 (Truth)**.** *Each task t_i has a true answer, called the* ground truth *(or* truth*), denoted as v_i^*.*

Table 5.3: Notations Used in Chapter 5.

Notation	Description
t_i	the i-th task ($1 \leq i \leq n$) and $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$
w	the worker w and $\mathcal{W} = \{w\}$ is the set of workers
$\mathcal{W}^i$	the set of workers that have answered task t_i
$\mathcal{T}^w$	the set of tasks that have been answered by worker w
v_i^w	the answer given by worker w for task t_i
V	the set of workers' answers for all tasks, i.e., $V = \{v_i^w\}$
v_i^*	the (ground) truth for task t_i ($1 \leq i \leq n$)

For the example task set $\mathcal{T}$ in Table 5.1, only pairs (r_1= r_2) and (r_3= r_4) are *true*, and thus $v_1^* = v_6^* = \text{T}$, and others' truth are F.

Based on the above notations, the truth inference problem is to infer the (unknown) truth v_i^* for each task t_i based on V.

Definition 5.5 (Truth Inference in Crowdsourcing). *Given workers' answers V, infer the truth v_i^* of each task $t_i \in \mathcal{T}$.*

Table 5.3 summarizes the notations used in the chapter.

5.3 Solution Framework

A naive solution is Majority Voting (MV) [70, 137, 151], which regards the choice answered by majority workers as the truth. Based on Table 5.2, the truth derived by MV is v_i^* = F for $2 \leq i \leq 6$ and it randomly infers v_1^* to break the tie. The MV incorrectly infers v_6^*, and has 50% chance to infer v_1^* wrongly. The reason is that MV assumes that each worker has the same quality, and in reality, workers have different qualities: some are experts or ordinary workers, while others are spammers (who randomly answer tasks in order to deceive money) or even malicious workers (who intentionally give wrong answers). Take a closer look at Table 5.2, we can observe that w_3 has a higher quality, and the reason is that if we do not consider t_1 (which receives 1 'T' and 1 'F'), then w_3

gives 4 out of 5 answers that are reported by majority workers, while w_1 and w_2 give both 3 out of 5, thus we should give higher trust to w_3's answer and in this way can infer all tasks' truth correctly.

Based on the above discussions, existing works [21,47,48,63,100,104,118, 126,127,131,161,182,197,200,225,226] propose various ways to model a worker's quality. Although qualification test and hidden test can help to estimate a worker's quality, they require to label tasks with truth beforehand, and a worker also requires to answer these "extra" tasks. To address this problem, existing works [21,47,48,63,100,104,118,126,127,131,161,182,197,200,225,226] estimate each worker's quality purely based on workers' answers V. Intuitively, they capture the inherent relations between workers' qualities and tasks' truth: for a task, the answer given by a high-quality worker is highly likely to be the truth; conversely, for a worker, if the worker often correctly answers tasks, then the worker will be assigned with a high quality. By capturing such relations, they adopt an iterative approach, which jointly infers both the workers' qualities and tasks' truth.

By capturing the above relations, the general approach adopted by most of existing works [21,47,48,63,100,104,118,126,127,131,161,182,197,200,225,226] is shown in Algorithm 11. The quality of each worker $w \in \mathcal{W}$ is denoted as q^w. In Algorithm 11, it first initializes workers' qualities randomly or using qualification test (line 1), and then adopts an iterative approach with two steps (lines 4-16):

Step 1: Inferring the Truth (lines 4-5): it infers each task's truth based on workers' answers and qualities. In this step, different task types are handled differently. Furthermore, some existing works [197,200] explicitly model each task, e.g., [200] regards that different tasks may have different difficulties. We discuss how existing works model a task in Section 5.4.1.

Step 2: Estimating Worker Quality (lines 11-12): based on workers' answers and each task's truth (derived from step 1), it estimates each worker's quality.

In this step, existing works model each worker w's quality q^w differently. For example, [21,48,100,126] model q^w as a single value, while [47,104,126,161,182] model q^w as a matrix. We discuss worker's models in Section 5.4.2.

Convergence (lines 15-16): the two iterations will run until convergence. Typically to identify convergence, existing works will check whether the change of two sets of parameters (i.e., workers' qualities and tasks' truth) is below some defined threshold (e.g., 10^{-3}). Finally the inferred truth and workers' qualities are returned.

Running Example. Let us show how the method PM [21, 119] works for Table 5.2. PM models each worker w as a single value $q^w \in [0, +\infty)$ and a higher value implies a higher quality. Initially, each worker $w \in \mathcal{W}$ is assigned with the same quality $q^w = 1$. Then the two steps devised in PM are as follows:

Step 1 (line 5): $v_i^* = \text{argmax}_v \sum_{w \in \mathcal{W}^i} q^w \cdot \mathbb{1}_{\{v = v_i^w\}}$;

Step 2 (line 12): $q^w = -\log\big(\frac{\sum_{t_i \in \mathcal{T}^w} \mathbb{1}_{\{v_i^* \neq v_i^w\}}}{\max_{w \in \mathcal{W}}\{\sum_{t_i \in \mathcal{T}^w} \mathbb{1}_{\{v_i^* \neq v_i^w\}}\}}\big)$.

The indicator function $\mathbb{1}_{\{\cdot\}}$ returns 1 if the statement is true; 0, otherwise. For example, $\mathbb{1}_{\{5=3\}} = 0$ and $\mathbb{1}_{\{5=5\}} = 1$. For the 1st iteration, in step 1, it computes each task's truth from workers' answers by considering which choice receives the highest aggregated workers' qualities. Intuitively, the answer given by many high quality workers are likely to be the truth. For example, for task t_2, as it receives one T and two F's from workers and each worker is of the same quality, then v_2^* = F. Similarly we get v_1^* = T and v_i^* = F for $2 \leq i \leq 6$. In step 2, based on the computed truth in step 1, it gives a high (low) quality to a worker if the worker makes few (a lot of) mistakes. For example, as the number of mistakes (i.e., $\sum_{t_i \in \mathcal{T}^w} \mathbb{1}_{\{v_i^* \neq v_i^w\}}$) for workers w_1, w_2, w_3 are 3, 2, 1, respectively, thus the computed qualities are $q^{w_1} = -\log(3/3) = 0$, $q^{w_2} = -\log(2/3) = 0.41$ and $q^{w_3} = -\log(1/3) = 1.10$. Following these two steps, the process will then iterate until convergence. In the converged results, the truth are $v_1^* = v_6^*$ = T, and v_i^* = F $(2 \leq i \leq 5)$; the qualities are $q^{w_1} = 4.9 \times 10^{-15}$, $q^{w_2} = 0.29$ and $q^{w_3} = 16.09$.

Algorithm 11 Solution Framework (Chapter 5).

```
Input: workers' answers $V$
Output: inferred truth $v_i^*$ ($1 \le i \le n$), worker quality $q^w$ ($w \in \mathcal{W}$)
 1: Initialize all workers' qualities ($q^w$ for $w \in \mathcal{W}$);
 2: while true do
 3:     // Step 1: Inferring the Truth
 4:     for $1 \le i \le n$ do
 5:         Inferring the truth $v_i^*$ based on $V$ and $\{q^w \mid w \in \mathcal{W}\}$;
 6:     end for
 7:     // Step 2: Estimating Worker Quality
 8:     for $w \in \mathcal{W}$ do
 9:         Estimating the quality $q^w$ based on $V$ and $\{v_i^* \mid 1 \le i \le n\}$;
10:     end for
11:     // Check for Convergence
12:     if Converged then
13:         break;
14:     end if
15: end while
16: return $v_i^*$ for $1 \le i \le n$ and $q^w$ for $w \in \mathcal{W}$;
```

We can observe that PM can derive the truth correctly, and w_3 has a higher quality compared with w_1 and w_2.

5.4 Important Factors

In this section, we categorize existing works [21,47,48,63,100,104,118,126,127,131,161,182,197,200,225,226] following two factors:

Task Modeling (Section 5.4.1): how existing works model a task (e.g., task's difficulty, latent topics).

Worker Modeling (Section 5.4.2): how existing works model a worker's quality (e.g., worker probability, diverse skills).

We summarize how existing works [21,47,48,63,100,104,118,126,127,131,161,182,197,200,225,226] can be categorized based on the above factors in Table 5.4. Next we analyze each factor, respectively.

Table 5.4: Comparisons of Different Truth Inference Methods.

Method	Task Types	Task Modeling	Worker Modeling	Techniques
MV	DM, SC	No Model	No Model	Direct
Mean	Numeric	No Model	No Model	Direct
Median	Numeric	No Model	No Model	Direct
ZC [48]	DM, SC	No Model	Worker Probability	PGM
GLAD [200]	DM, SC	Task Difficulty	Worker Probability	PGM
D&S [47]	DM, SC	No Model	Confusion Matrix	PGM
Minimax [226]	DM, SC	No Model	Diverse Skills	Optimization
BCC [104]	DM, SC	No Model	Confusion Matrix	PGM
CBCC [182]	DM, SC	No Model	Confusion Matrix	PGM
LFC [161]	DM, SC	No Model	Confusion Matrix	PGM
CATD [118]	DM, SC, Numeric	No Model	Worker Probability Confidence	Optimization
PM [21,119]	DM, SC, Numeric	No Model	Worker Probability	Optimization
Multi [197]	DM	Latent Topics	Diverse Skills Worker Bias Worker Variance	PGM
KOS [100]	DM	No Model	Worker Probability	PGM
VI-BP [126]	DM	No Model	Confusion Matrix	PGM
VI-MF [126]	DM	No Model	Confusion Matrix	PGM
LFC_N [161]	Numeric	No Model	Worker Variance	PGM

Note: In the table, "DM" means "Decision-Making", "SC" means "Single-Choice".

5.4.1 Task Modeling

Task Difficulty

Different from most existing works which assume that a worker has the same quality for answering different tasks, some recent works [131,200] model the difficulty in each task. They assume that each task has its difficulty level, and the more difficult a task is, the harder a worker can correctly answer the task. For example, in [200], it models the probability that worker w correctly answers task t_i as follows: $\Pr(v_i^w = v_i^* \mid d_i, q^w) = 1/(1 + e^{-d_i \cdot q^w})$, where $d_i \in (0, +\infty)$ represents the difficulty for task t_i, and the higher d_i is, the easier task t_i is. Intuitively, for a fixed worker quality $q^w > 0$, an easier task (high value of d_i) leads to a higher probability that the worker correctly answers the task.

Latent Topics

Different from modeling each task as a value (e.g., difficulty), some recent works [63,131,197,217] model each task as a vector with K values. The basic idea is to exploit the diverse topics in a task, where the topic number (i.e., K) is pre-defined. For example, existing studies [63,131] make use of the text description in each task and adopt topic model techniques [25,216] to generate a vector of size K for the task; while Multi [197] learns a K-size vector without referring to external information (e.g., text descriptions). Based on the task models, a worker is probable to answer a task correctly if the worker has high qualities on the task's related topics.

5.4.2 Worker Modeling

Worker Probability

Worker probability uses a single real number (between 0 and 1) to model a worker w's quality $q^w \in [0, 1]$, which represents the ability that worker w correctly answers a task. The higher q^w is, the worker w has higher ability to correctly answer tasks. The model has been widely used in existing works [21,48,100,126]. Some recent works [119,200] extend the worker probability to model a worker's quality in a wider range, e.g., $q^w \in (-\infty, +\infty)$, and a higher q^w means the worker w's higher quality in answering tasks.

Confusion Matrix

Confusion matrix [47,104,126,161,182] is used to model a worker's quality for answering single-choice tasks. Suppose each task in $\mathcal{T}$ has ℓ fixed choices, then the confusion matrix q^w is an $\ell \times \ell$ matrix, where the j-th ($1 \leq j \leq \ell$) row, i.e., $q^w_{j,\cdot} = [\ q^w_{j,1}, q^w_{j,2}, \ldots, q^w_{j,\ell}\ ]$, represents the probability distribution of worker w's possible answers for a task if the truth of the task is the j-th choice. Each element $q^w_{j,k}$ $(1 \le j \le \ell, 1 \le k \le \ell)$ means that "given the truth of a task is the j-th choice, the probability that worker w selects the k-th choice", i.e., $q^w_{j,k} = \Pr(v^w_i = k \mid v^*_i = j)$ for any $t_i \in \mathcal{T}$. For example, decision-making tasks ask workers to select 'T' (1st choice) or 'F' (2nd choice) for each claim ($\ell = 2$), then an example confusion matrix for w is $q^w = \begin{bmatrix} 0.8 & 0.2 \\ 0.3 & 0.7 \end{bmatrix}$, where $q^w_{1,2} = 0.2$ means that if the truth of a task is 'T', the probability that the worker answers the task as 'F' is 0.2.

Worker Bias and Worker Variance

Worker bias and variance [161,197] are proposed to handle numeric tasks, where worker bias captures the effect that a worker may underestimate (or overestimate) the truth of a task, and worker variance captures the variation of errors around the bias. For example, given a set of photos with humans, each numeric task asks workers to estimate the height of the human on it. Suppose a worker w is modeled with bias τ_w and variance σ_w, then the answer v^w_i given by worker w is modeled to draw from the Gaussian distribution: $v^w_i \sim \mathcal{N}(v^*_i + \tau_w, \sigma_w)$, that is, (1) a worker with bias $\tau_w \gg 0$ ($\tau_w \ll 0$) will overestimate (underestimate) the height, while $\tau_w \to 0$ leads to more accurate estimation; (2) a worker with variance $\sigma_w \gg 0$ means a large variation of error, while $\sigma_w \to 0$ leads to a small variation of error.

Confidence

Existing works [98, 118] observe that if a worker answers plenty of tasks, then the estimated quality for the worker is confident; otherwise, if a worker answers only a few tasks, then the estimated quality is not confident. Inspired by this observation, [131] assigns higher qualities to the workers who answer plenty of tasks, than the workers who answer a few tasks. To be specific, for a worker w, it uses the Chi-Square distribution [2] with 95% confidence interval, i.e., $\mathcal{X}^2_{(0.975,|\mathcal{T}^w|)}$ as a coefficient to scale up the worker's quality, where $|\mathcal{T}^w|$ is the number of tasks that worker w has answered. $\mathcal{X}^2_{(0.975,|\mathcal{T}^w|)}$ increases with $|\mathcal{T}^w|$, i.e., the more tasks w has answered, the higher worker w's quality is scaled to.

Diverse Skills

A worker may have various levels of expertise for different topics. For example, a sports fan that rarely pays attention to entertainment may answer tasks related to *sports* more correctly than tasks related to *entertainment*. Different from most of the above models which have an assumption that a worker has the *same* quality to answer different tasks, existing works [63,131,197,217,220,226] model the diverse skills in a worker and capture a worker's diverse qualities for different tasks. The basic ideas of [63,226] are that they model a worker w's quality as a vector of size n, i.e., $q^w = [\, q^w_1, q^w_2, \ldots, q^w_n \,]$, where q^w_i indicates worker w's quality for task t_i. Different from [63,226], some recent works [131,197,217,220] model a worker's quality for different latent topics, i.e., $q^w = [\, q^w_1, q^w_2, \ldots, q^w_K \,]$, where the number K is pre-defined, indicating the number of latent topics. They [131, 197, 217, 220] assume that each task is related to one or more topics in these K latent topics, and a worker is highly probable to correctly answer a task if the worker has a high quality in the task's related topics.

5.5 Truth Inference Algorithms

Existing works [21,47,48,63,100,104,118,126,127,131,161,182,197,200,226] usually adopt the framework in Algorithm 11. Based on the used techniques, they can be classified into the following three categories: direct computation [70, 151], optimization methods [21, 63, 118, 226] and probabilistic graphical model methods [47,48,100,104,126,127,131,161,182,197,200]. Next we talk about them, respectively.

5.5.1 Direct Computation

Some baseline methods directly estimate v_i^* ($1 \leq i \leq n$) based on V, without modeling each worker or task. For decision-making and single-label tasks, Majority Voting (MV) regards the truth of each task as the answer given by most workers; while for numeric tasks, Mean and Median are two baseline methods that regard the mean and median of workers' answers as the truth for each task.

5.5.2 Optimization

The basic idea of optimization methods is to set a self-defined optimization function that captures the relations between workers' qualities and tasks' truth, and then derive an iterative method to compute these two sets of parameters collectively. The differences among existing works [21,118,119,226] are that they model workers' qualities differently and apply different optimization functions to capture the relations between the two sets of parameters.

(1) Worker Probability. PM [21, 119] models each worker's quality as a single value, and the optimization function is defined as:

$$\min_{\{q^w\},\{v_i^*\}} f(\{q^w\},\{v_i^*\}) = \sum_{w \in \mathcal{W}} q^w \cdot \sum_{t_i \in \mathcal{T}^w} d(v_i^w, v_i^*),$$

where $\{q^w\}$ represents the set of all workers' qualities, and similarly $\{v_i^*\}$ represents the set of all truth. It models a worker w's quality as $q^w \geq 0$, and $d(v_i^w, v_i^*) \geq 0$ defines the distance between worker's answer v_i^w and the truth v_i^*: the similar v_i^w is to v_i^*, the lower the value of $d(v_i^w, v_i^*)$ is. Intuitively, to minimize $f(\{q^w\},\{v_i^*\})$, a worker w's high quality q^w corresponds to a low value in $d(v_i^*, v_i^w)$, i.e., worker w's answer should be close to the truth. By capturing the intuitions, similar to Algorithm 11, PM [21,119] develops an iterative approach, and in each iteration, it adopts the two steps as illustrated in Section 5.3.

(2) Worker Probability and Confidence. Different from above, CATD [118] considers both worker probability and confidence in modeling a worker's quality. As discussed in Section 5.4.2, each worker w's quality is scaled up to a coefficient of $\mathcal{X}^2_{(0.975,|\mathcal{T}^w|)}$, i.e., the more tasks w has answered, the higher worker w's quality is scaled to. It develops an objective function, with the intuitions that a worker w who gives answers close to the truth and answers a plenty of tasks should have a high quality q^w. Similarly it adopts an iterative approach, and iterates the two steps until convergence.

(3) Diverse Skills. Minimax [226] leverages the idea of minimax entropy [227]. To be specific, it models the diverse skills of a worker w across different tasks and focuses on single-label tasks (with ℓ choices). It assumes that for a task t_i, the answers given by w are generated by a probability distribution $\pi^w_{i,\cdot} = [\ \pi^w_{i,1}, \pi^w_{i,2}, \ldots, \pi^w_{i,\ell}\ ]$, where each $\pi^w_{i,j}$ is the probability that worker w answers task t_i with the j-th choice. Following this, an objective function is defined by considering two constraints for tasks and workers: for a task t_i, the number of answers collected for a choice equals the sum of corresponding generated probabilities; for a worker w, among all tasks answered by w, given the truth is the j-th choice, the number of answers collected for the k-th choice equals the sum of corresponding generated probabilities. Finally [226] devises an iterative approach to infer the two sets of parameters $\{v^*_i\}$ and $\{\pi^w\}$.

5.5.3 Probabilistic Graphical Model (PGM)

A probabilistic graphical model [107] is a graph which expresses the conditional dependency structure (represented by edges) between random variables (represented by nodes). Figure 5.1 shows the general PGM adopted in existing works. Each node represents a variable. There are two plates, respectively for workers and tasks, where each one represents the repeating variables. For example, the plate for workers represents $|\mathcal{W}|$ repeating variables, where each variable corresponds to a worker $w \in \mathcal{W}$. For the variables, α, β, and v^w_i are known (α and β are priors for q^w and v_i^*, which can be set based on the prior knowledge); q^w and v_i^* are latent or unknown variables, which are two desired variables to compute. The directed edges model the conditional dependence between a child node and its associated parent node(s) in the sense that the child node follows a probabilistic distribution conditioned on the values taken by the parent node(s). For example, three conditional distributions in Figure 5.1 are $\Pr(q^w \mid \alpha)$, $\Pr(v_i^* \mid \beta)$ and $\Pr(v_i^w \mid q^w, v_i^*)$.

Next we illustrate the details (optimization goal and the two steps) of each method using PGM. In general the methods differ in the used worker model. It can be classified into three categories: **worker probability** [48, 100, 126, 200], **confusion matrix** [47, 104, 161, 182] and **diverse skills** [63, 131, 197]. For each category, we first introduce its basic method, e.g., ZC [48], and then summarize how other methods [100, 126, 200] extend the basic method ZC [48].

(1) Worker Probability: ZC [48] and its extensions [100, 126, 200].

ZC [48] adopts a PGM similar to Figure 5.1, with the simplification that it does not consider the priors (i.e., α, β). Suppose all tasks are decision-making tasks ($v_i^* \in \{\text{T, F}\}$) and each worker's quality is modeled as worker probability $q^w \in [0, 1]$. Then

$$\Pr(v_i^w \mid q^w, v_i^*) = (q^w)^{\mathbb{1}_{\{v_i^w = v_i^*\}}} \cdot (1 - q^w)^{\mathbb{1}_{\{v_i^w \neq v_i^*\}}},$$

which means that the probability worker w correctly (incorrectly) answers a task is q^w ($1 - q^w$). For decision-making tasks, ZC [48] tries to maximize the probability of the occurrence of workers' answers, called *likelihood*, i.e., $\max_{\{q^w\}} \Pr(V \mid \{q^w\})$, which regards $\{v_i^*\}$ as latent variables:

$$\Pr(V| \{q^w\}) = \frac{1}{2} \cdot \prod\nolimits_{i=1}^{n} \sum\nolimits_{z \in \{\text{T, F}\}} \prod\nolimits_{w \in \mathcal{W}^i} \Pr(v_i^w | \, q^w, v_i^* = z). \tag{5.1}$$

However, it is hard to optimize due to the non-convexity. Thus ZC [48] applies the EM (Expectation-Maximization) framework [49] and iteratively updates q^w

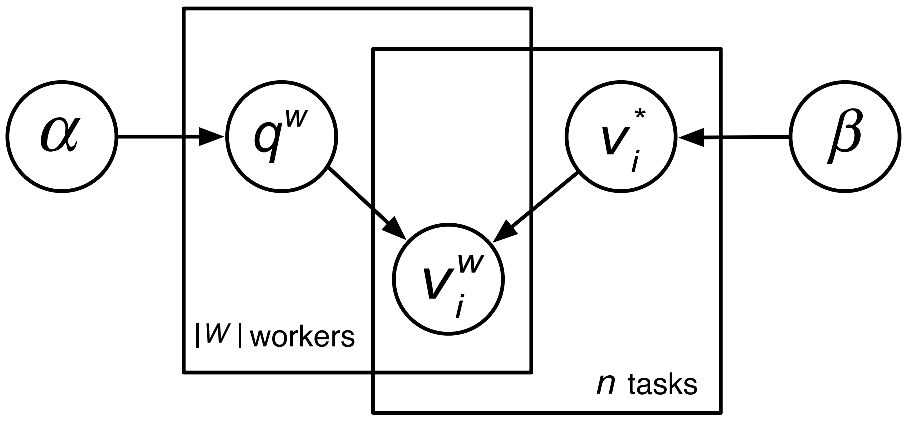


Figure 5.1: The Probabilistic Graphical Model Framework.

and v_i^* to approximate its optimal value. Note ZC [48] develops a system to address entity linking for online pages. In this chapter we focus on the part of leveraging the crowd's answers to infer the truth (i.e., Section 4.3 in [48]), and we omit other parts (e.g., constraints on its probabilistic model).

There are several extensions of ZC, e.g., GLAD [200], KOS [100], VI-BP [126], VI-MF [126], and they focus on different perspectives:

Task Model. GLAD [200] extends ZC [48] in task model. Rather than assuming that each task is the same, it [200] models each task t_i's difficulty $d_i \in (0, +\infty)$ (the higher, the easier). Then it models the worker's answer as $\Pr(v_i^w = v_i^* \mid d_i, q^w) = 1/(1 + e^{-d_i \cdot q^w})$, and integrates it into Equation 5.1 to approximate the optimal value using Gradient Descent [107] (an iterative method).

Optimization Function. KOS [100], VI-BP [126], and VI-MF [126] extend ZC [48] in an optimization goal. Recall that ZC tries to compute the optimal $\{q^w\}$ that maximizes $\Pr(V \mid \{q^w\})$, which is the *Point Estimate*. Instead, [100,126] leverage the *Bayesian Estimators* to calculate the integral of all possible q^w, and the target is to estimate the truth $v_i^* = \operatorname{argmax}_{z \in \{\mathrm{T},\,\mathrm{F}\}} \Pr(v_i^* = z \mid V)$, where

$$\Pr(v_i^* = z \mid V) = \int_{\{q^w\}} \Pr(v_i^* = z, \{q^w\} \mid V) \ \mathrm{d}\{q^w\}. \tag{5.2}$$

It is hard to directly compute Equation 5.2, and existing works [100,126] seek for *Variational Inference* (*VI*) techniques [191] to approximate the value: KOS [100] first leverages *Belief Propagation* (one typical *VI* technique) to iteratively approximate the value in Equation 5.2, then [126] proposes a more general model based on KOS, called VI-BP. Moreover, it [126] also applies *Mean Field* (anther *VI* technique) in VI-MF to iteratively approach Equation 5.2.

(2) Confusion Matrix: D&S [47] and its extensions [104,161,182].

D&S [47] focuses on single-label tasks (with fixed ℓ choices) and models each worker as a confusion matrix q^w with size $\ell \times \ell$ (Section 5.4.2). The worker w's answer follows the probability $\Pr(v_i^w \mid q^w, v_i^*) = q^w_{v_i^*, v_i^w}$. Similar to Equation 5.1, D&S [47] tries to optimize the function $\operatorname{argmax}_{\{q^w\}} \Pr(V \mid \{q^w\})$, where

$$\Pr(V \mid \{q^w\}) = \textstyle\prod_{i=1}^{n} \sum_{1 \le z \le \ell} \Pr(v_i^* = z) \cdot \prod_{w \in \mathcal{W}^i} q^w_{z, v_i^w},$$

and it applies the EM framework [49] to devise two iterative steps.

The above method D&S [47], which models a worker as a confusion matrix, is also a widely used model. There are some extensions, e.g., LFC [161], LFC_N [161], BCC [104] and CBCC [182].

Priors. LFC [161] extends D&S [47] to incorporate the priors into worker's model, by assuming that the priors, denoted as $\alpha^w_{j,k}$ for $1 \le j, k \le \ell$ are known in advance, and the worker's quality $q^w_{j,k}$ is generated following $\text{Beta}(\alpha^w_{j,k}, \sum_{k=1}^{\ell} \alpha^w_{j,k})$ distribution.

Task Type. LFC_N [161] also handles numeric tasks. Different from decision-making and single-choice tasks, it assumes that worker w's answer follows $v_i^w \sim \mathcal{N}(v_i^*, \sigma_w^2)$, where σ_w is the variance, and a small σ_w implies that v_i^w is close to truth v_i^*.

Optimization Function. BCC [104] has a different optimization goal compared with D&S [47] and it aims at maximizing the posterior joint probability. For example, in Figure 5.1, it optimizes the posterior joint probability of all unknown variables, i.e.,

$$\prod_{i=1}^{n} \Pr(v_i^* \mid \beta) \prod_{w \in \mathcal{W}} \Pr(q^w \mid \alpha) \prod_{i=1}^{n} \prod_{w \in \mathcal{W}^i} \Pr(v_i^w \mid q^w, v_i^*).$$

To optimize the above formula, the technique of Gibbs Sampling [107] is used to iteratively infer the two sets of parameters $\{q^w\}, \{v_i^*\}$ until convergence, where q^w is modeled as a confusion matrix. Then CBCC [182] extends BCC [104] to support community. The basic idea is that each worker belongs to one community, where each community has a representative confusion matrix, and workers in the same community share very similar confusion matrices.

(3) Diverse Skills: Multi [197] and others [63, 131, 220].

Recently, there are some works (e.g., [63, 131, 197, 220]) that model a worker's diverse skills. Basically, they model a worker w's quality q^w as a vector of size K (Section 5.4.2), which captures a worker's diverse skills over K latent topics. For example, [131] combines the process of topic model (i.e., TwitterLDA [216]) and truth inference together, and [220] leverages entity linking and knowledge base to exploit a worker's diverse skills.

5.6 Experiments

In this section, we evaluate 17 existing methods (Table 5.4) on real datasets. We first introduce the experimental setup (Section 5.6.1), and then analyze the quality of collected crowdsourced data (Section 5.6.2). Finally we compare with existing methods (Section 5.6.3). We have made all our used datasets and codes available [157] for reproducibility and future research. We implement the experiments in Python on a server with CPU 2.40GHz and 60GB memory.

Table 5.5: The Statistics of Each Dataset.

Dataset	**#tasks (n)**	**#truth**	$\lvert V \rvert$	$\lvert V \rvert / n$	$\lvert \mathcal{W} \rvert$
Datasets for Decision-Making Tasks					
`D_Product` [192]	8,315	8,315	24,945	3	176
`D_PosSent`	1,000	1,000	20,000	20	85
Datasets for Single-Label Tasks					
`S_Rel` [31]	20,232	4,460	98,453	4.9	766
`S_Adult` [11]	11,040	1,517	92,721	8.4	825
Datasets for Numeric Tasks					
`N_Emotion` [173]	700	700	7,000	10	38

5.6.1 Experimental Setup

Datasets

There are many public crowdsourcing datasets [43]. Among them, we select 5 representative datasets based on three criteria: (1) the dataset is large in task size; (2) each task received multiple answers; (3) all datasets cover different task types. In Table 5.5, for each selected dataset, we list four statistics: the number of tasks, or #tasks (n), #collected answers ($\lvert V \rvert$), the average number of answers for each task ($\lvert V \rvert / n$), #truth (some large datasets only provide a subset as ground truth) and #workers ($\lvert \mathcal{W} \rvert$). For example, for dataset `D_Product`, it contains 8,315 tasks, with 24,945 answers collected from 176 workers, and each task is answered with 3 times on average. Next, we introduce the details of each dataset (with different task types). We manually collect answers for `D_PosSent` [180] from AMT [1]; while for other datasets, we use the public datasets collected by other researchers [11,31,173,192].

Decision-Making Tasks (start with prefix '`D_`'):

• `D_Product` [192]. Each task in the dataset contains two products (with descriptions) and two choices (T, F), and it asks workers to identify whether the claim "*the two products are the same*" is true ('T') or false ('F'). An example task is "*Sony Camera Carrying-LCSMX100* and *Sony LCS-MX100 Camcorder* are the same?".

There are 8135 tasks, and 1101 (7034) tasks' truth are T (F).

• `D_PosSent`. Each task in the dataset contains a tweet related to a company (e.g., "*The recent products of Apple is amazing!*"), and asks workers to identify whether the tweet has positive sentiment to that company. The workers give '*yes*' or '*no*' to each task. Based on the dataset [180], we create 1000 tasks. Among them, 528 (472) tasks' truth are *yes* (*no*). In AMT [1], we batch 20 tasks in a Human Intelligence Task (HIT) and assign each HIT to 20 workers. We pay each worker \$0.03 upon answering a HIT. We manually create qualification test by selecting 20 tasks, and each worker should answer the qualification test before she can answer our tasks.

Single-Choice Tasks (start with prefix '`S_`'):

• `S_Rel` [31]. Each task contains a topic and a document, and it asks workers to choose the relevance of the topic w.r.t. the document by selecting one out of four choices: '*highly relevant*', '*relevant*', '*non-relevant*', and '*broken link*'.

• `S_Adult` [11]. Each task contains a website, and it asks workers to identify the adult level of the website by selecting one out of four choices: '*G*' (General Audience), '*PG*' (Parental Guidance), '*R*' (Restricted), and '*X*' (Porn).

Numeric Tasks (start with prefix '`N_`'):

• `N_Emotion` [173]. Each task in the dataset contains a text and a range $[-100, 100]$, and it asks each worker to select a score in the range, indicating the degree of emotion (e.g., anger) of the text. A higher score means a higher degree for the emotion.

Metrics

We use different metrics for different task types.

Decision-Making Tasks. We use *Accuracy* as the metric, which is defined as the fraction of tasks whose truth are inferred correctly. Given a method, let $\widehat{v}_i^*$ denote the inferred truth of task t_i, then

$$Accuracy = \sum_{i=1}^{n} \mathbb{1}_{\{\widehat{v}_i^* = v_i^*\}} / n. \tag{5.3}$$

However, for applications such as entity resolution (e.g., dataset D_Product), where the number of F is much larger than the number of T as truth (the proportion of tasks with T and F as truth is 0.12:0.88 in D_Product). In this case, even a naive method that returns all tasks as F achieves very high *Accuracy* (88%), which is not expected, as we care more for the *same entities* (i.e., choice T) in entity resolution. Thus a typical metric *F1-score* is often used, which is defined as the harmonic mean of *Precision* and *Recall*:

$$F1\text{-}score = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} = \frac{2 \cdot \sum_{i=1}^{n} \mathbb{1}_{\{v_i^* = \mathrm{T}\}} \cdot \mathbb{1}_{\{\widehat{v}_i^* = \mathrm{T}\}}}{\sum_{i=1}^{n} (\mathbb{1}_{\{v_i^* = \mathrm{T}\}} + \mathbb{1}_{\{\widehat{v}_i^* = \mathrm{T}\}})}. \tag{5.4}$$

Single-Choice Tasks. We use the metric *Accuracy* (Equation 5.3).

Numeric Tasks. We use two metrics, *MAE* (Mean Absolute Error) and *RMSE* (Root Mean Square Error), defined as below:

$$MAE = \frac{\sum_{i=1}^{n} |v_i^* - \widehat{v}_i^*|}{n}, \quad RMSE = \sqrt{\frac{\sum_{i=1}^{n} (v_i^* - \widehat{v}_i^*)^2}{n}}, \tag{5.5}$$

where *RMSE* gives a higher penalty for large errors.

Note that for the metrics *Accuracy* and *F1-score*, they are in $[0, 1]$ and the higher, the better; however, for *MAE* and *RMSE* (defined on errors), they are in $[0, +\infty]$ and the lower, the better.

5.6.2 Crowdsourced Data Quality

In this section we first ask the following three questions related to the quality of crowdsourced data, and then answer them.

1. Are the crowdsourced data consistent? In other words, are the answers from different workers the same for a task? (Section 5.6.2)

2. Are there a lot of redundant workers? In other words, does each worker answer plenty of tasks? (Section 5.6.2)

3. Do workers provide high-quality data? In other words, are each worker's answers consistent with the truth? (Section 5.6.2)

Data Consistency

Decision-Making & Single-Label Tasks. Note that each task contains a fixed number (denoted as ℓ) of choices. For a task t_i, let $n_{i,j}$ denote the number of answers given to the j-th choice, e.g., in Table 5.2, for t_2, $n_{2,1} = 1$ and $n_{2,2} = 2$. In order to capture how concentrated the workers' answers are, we first compute the entropy [167] over the distribution of each task's collected answers, and then define data consistency ($\mathcal{C}$) as the average entropy, i.e., $\mathcal{C} = -\frac{1}{n} \cdot \sum_{i=1}^{n} \sum_{j=1}^{\ell} \frac{n_{i,j}}{\sum_{j=1}^{\ell} n_{i,j}} \cdot \log_{\ell} \frac{n_{i,j}}{\sum_{j=1}^{\ell} n_{i,j}}$. Note that we use "$\log_{\ell}$" other than "ln" to ensure that the value $\mathcal{C} \in [0,1]$, and the lower $\mathcal{C}$ is, the more consistent workers' answers are.

Based on V, we compute $\mathcal{C}$ for each dataset. The computed $\mathcal{C}$ of the four datasets are 0.38, 0.85, 0.82, and 0.39, respectively. It can be seen that the crowdsourced data is not consistent. To be specific, for decision-making and single-label datasets, $\mathcal{C} \geq 0.38$, and there exists highly inconsistent dataset `D_PosSent` with $\mathcal{C} = 0.85$.

Numeric Tasks. As the answers obtained for each task has inherent orderings, in order to capture the consistency of workers' answers, for a task t_i, we first compute the *median* v_i (a robust metric in statistics and it is not sensitive to outliers) over all its collected answers; then the consistency ($\mathcal{C}$) is defined as the average deviation compared with the median, i.e., $\mathcal{C} = \frac{1}{n} \cdot \sum_{i=1}^{n} \sqrt{\frac{\sum_{w \in \mathcal{W}^i} (v_i^w - v_i)^2}{|\mathcal{W}^i|}}$, where $\mathcal{W}^i$ is the set of workers that have answered t_i. We have $\mathcal{C} \in [0, +\infty]$, and a lower $\mathcal{C}$ leads to more consistent answers.

For numeric dataset `N_Emotion`, the computed $\mathcal{C}$ is 20.44.

Summary. The crowdsourced data is inconsistent, which motivates to develop methods that can solve truth inference in crowdsourcing.

Worker Redundancy

For each worker, we define her redundancy as the number of tasks answered by the worker. We record the redundancy of each worker in each dataset, and then draw the histograms of worker redundancies in Figure 5.2. Specifically, in each dataset, we vary the number of tasks (k), and record how many workers that answer k tasks. We can see in Figure 5.2 that the worker redundancy conforms to the long-tail phenomenon, i.e., most workers answer a few tasks and only a few workers answer plenty of tasks.

Summary. The worker redundancy of crowdsourced data in real crowdsourcing platforms conforms to long-tail phenomenon.

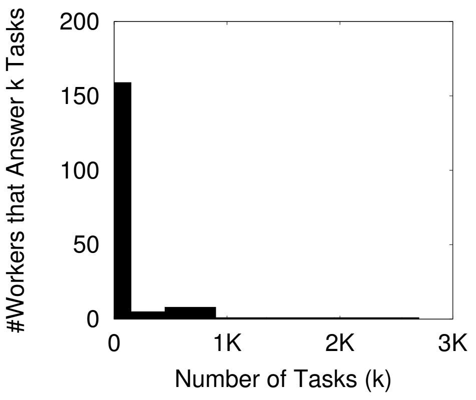
Worker Quality

In Figure 5.3, for each dataset, we show each worker's quality, computed based on comparing worker's answers with tasks' truth.

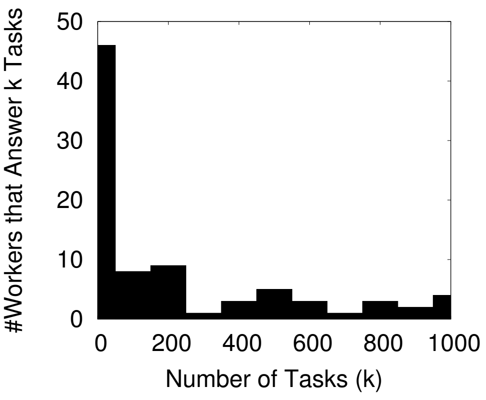
Decision-Making & Single-Label Tasks. We compute each worker w's *Accuracy*, i.e., the proportion of tasks that are correctly answered by w, i.e., $\frac{\sum_{t_i \in \mathcal{T}^w} \mathbb{1}_{\{v_i^w = v_i^*\}}}{|\mathcal{T}^w|} \in [0, 1]$ and a higher value means a higher quality. For each dataset, we compute the corresponding *Accuracy* for each worker and draw the histograms of each worker. It can be seen from Figures 5.3(a)-(d) that histograms of workers' *Accuracy* are in different shapes for different datasets. To be specific, workers for `D_Product` and `D_PosSent` are of high *Accuracy*, while workers have mediate *Accuracy* for `S_Adult`, and low *Accuracy* for `S_Rel`. The average *Accuracy* for all workers in each dataset are 0.79, 0.79, 0.53 and 0.65, respectively.

Numeric Tasks. It can be seen from Figure 5.3(e) that workers' *RMSE* vary in $[20, 45]$, and the average *RMSE* is 28.9.

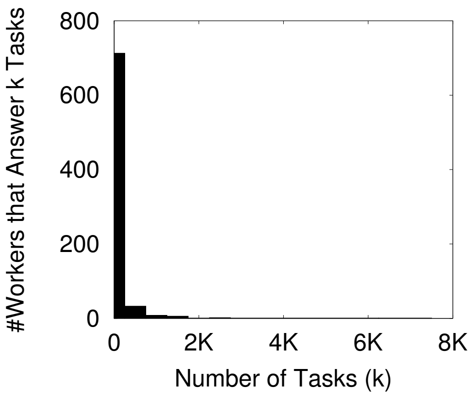
Summary. The workers' qualities vary in the same dataset, which makes it necessary to identify the trustworthy workers.



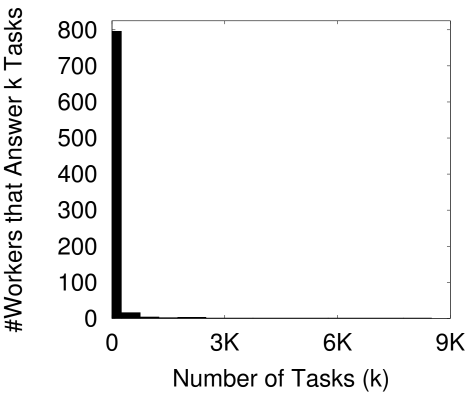
(a) D_Product (176 workers)



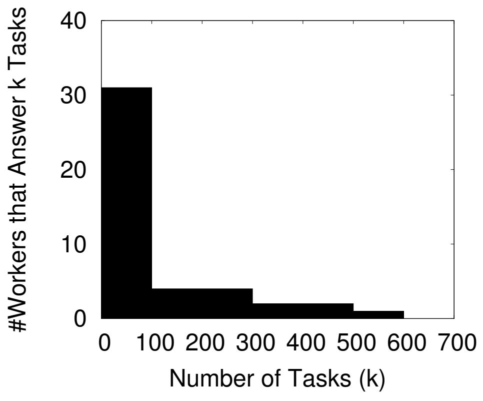
(b) D_PosSent (85 workers)



(c) S_Rel (766 workers)

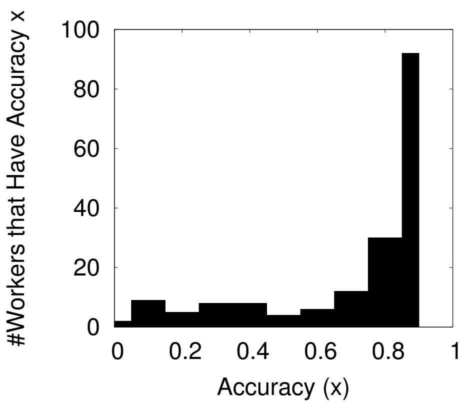


(d) S_Adult (825 workers)

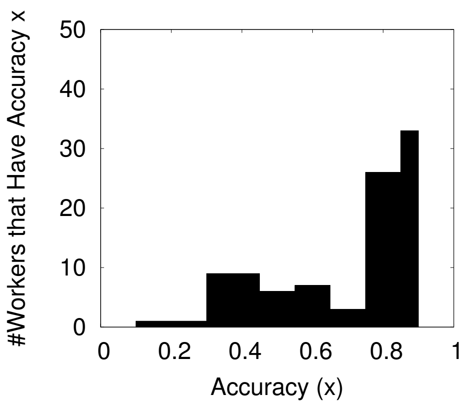


(e) N_Emotion (38 workers)

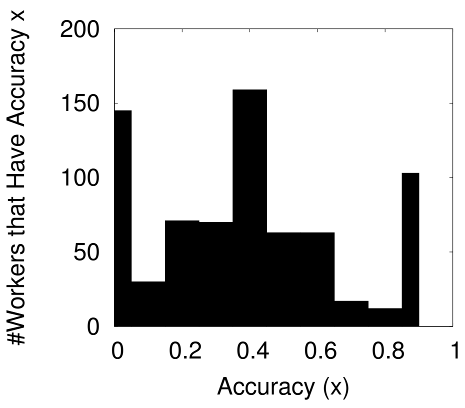
Figure 5.2: The Statistics of Worker Redundancy for Each Dataset.



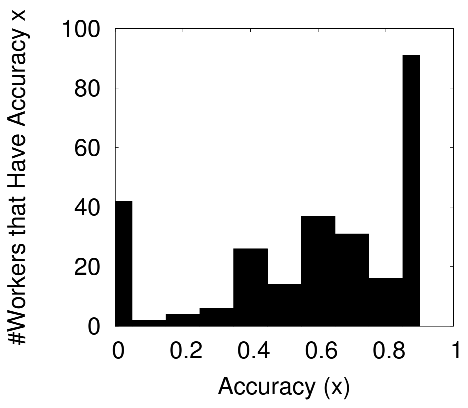
(a) D_Product (176 workers)



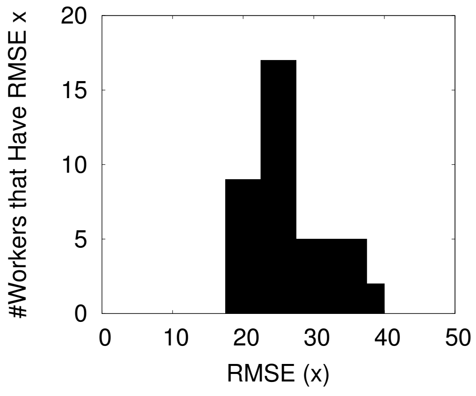
(b) D_PosSent (85 workers)



(c) S_Rel (766 workers)



(d) S_Adult (825 workers)



(e) N_Emotion (38 workers)

Figure 5.3: The Statistics of Worker Quality for Each Dataset.

5.6.3 Crowdsourced Truth Inference

In this section we compare the performance of existing methods [21,47,48, 100,104,118,119,126,127,161,182,197,200,226]. Our comparisons are performed based on the following perspectives:

1. What is the performance of different methods? In other words, if we only know the workers' answers (i.e., V), which method performs the best? Furthermore, for a method, how does the truth inference quality change with more workers' answers? (Section 5.6.3)

2. What is the effect of qualification test? In other words, if we assume a worker has performed some golden tasks before answering real tasks, and initialize the worker's quality (line 1 in Algorithm 11) based on the worker's answering performance for golden tasks, will this increase the quality of each method? (Section 5.6.3)

3. What is the effect of hidden test? In other words, if we mix a set of golden tasks in real tasks, then how much gain in truth inference can be benefited for each method? (Section 5.6.3)

4. What are the effects of different task types, task models, worker models, and inference techniques? In other words, what factors are beneficial to inferring the truth? (Section 5.6.3)

Varying Data Redundancy

For data redundancy, we define it as the number of answers collected for each task. In our 5 used datasets (Table 5.5), the data redundancy for each dataset is $|V|/n$. In Figures 5.4, 5.5, and 5.6, we observe the quality of each method in each dataset with varying data redundancy. For example, in Figure 5.4(a), on dataset `D_PosSent` (with $|V|/n = 3$), we compare with 14 methods that can be used in decision-making tasks (Table 5.4), i.e., MV, ZC, GLAD, D&S, Minimax, BCC, CBCC, LFC, CATD, PM, Multi, KOS, VI-BP and VI-MF. We vary the

Table 5.6: The Comparison of Different Methods on Decision-Making Tasks.

Method	D_Product			D_PosSent		
	Accuracy	*F1-score*	*Time*	*Accuracy*	*F1-score*	*Time*
MV	89.66%	59.05%	0.13s	93.31%	92.85%	0.08s
ZC [48]	92.80%	63.59%	1.04s	95.10%	94.60%	0.55s
GLAD [200]	92.20%	60.17%	907.11s	95.20%	94.71%	407.66s
D&S [47]	93.66%	**71.59%**	1.46s	**96.00%**	**95.66%**	0.80s
Minimax [226]	84.09%	55.26%	272.05s	95.80%	95.43%	35.71s
BCC [104]	**93.78%**	70.10%	9.82s	**96.00%**	**95.66%**	6.06s
CBCC [182]	93.72%	70.87%	5.53s	**96.00%**	**95.66%**	4.12s
LFC [161]	93.73%	71.48%	1.42s	**96.00%**	**95.66%**	0.83s
CATD [118]	92.66%	65.92%	2.97s	95.50%	95.07%	1.32s
PM [21,119]	89.81%	59.34%	0.56s	95.04%	94.53%	0.33s
Multi [197]	88.67%	58.32%	15.48s	95.70%	95.44%	4.98s
KOS [100]	89.55%	50.31%	24.06s	93.80%	93.06%	10.14s
VI-BP [126]	64.64%	37.43%	306.23s	**96.00%**	**95.66%**	58.52s
VI-MF [126]	83.91%	55.31%	38.96s	**96.00%**	**95.66%**	6.71s

data redundancy $r \in [1, 3]$, where for each specific r, we randomly select r out of 3 answers collected for each task, and construct a dataset with the selected answers (i.e., a dataset with the number of answers $r \cdot n$ for all n tasks). Then we run each method on the constructed dataset and record the *Accuracy* based on comparing each method's inferred truth with the ground truth. We repeat each experiment for 30 times and the average quality is reported. As discussed in Section 5.6.1, we use metrics *Accuracy*, *F1-score* on decision-making tasks (`D_Product`, `D_PosSent`), metric *Accuracy* on single-label tasks (`S_Rel`, `S_Adult`) and metrics *MAE*, *RMSE* on numeric tasks (`N_Emotion`). To have a clear comparison, we record the quality and efficiency in the complete dataset (i.e., with redundancy $|V|/n$) for all methods in Tables 5.6, 5.7 and 5.8. Based on the results in Figures 5.4-5.6, and Tables 5.6-5.8, we analyze the quality and efficiency of different methods.

(1) The Quality of Different Methods in Different Datasets.

Decision-Making Tasks. For dataset `D_Product`, i.e., Figures 5.4(a), (b), we can observe that (1) as the data redundancy r is varied in $[1, 3]$, the quality increases

Table 5.7: The Comparison of Different Methods on Single-Choice Tasks.

Method	S_Rel		S_Adult	
	Accuracy	*Time*	*Accuracy*	*Time*
MV	54.19%	0.49s	36.04%	0.40s
ZC [48]	48.21%	7.39s	35.34%	6.42s
GLAD [200]	53.59%	5850.39s	36.47%	4194.50s
D&S [47]	61.30%	10.67s	36.05%	9.18s
Minimax [226]	57.59%	1728.09s	36.03%	1223.75s
BCC [104]	60.72%	153.50s	36.34%	137.92s
CBCC [182]	56.05%	44.69s	36.28%	42.52s
LFC [161]	**61.64%**	10.75s	36.29%	9.26s
CATD [118]	45.32%	16.13s	36.23%	12.96s
PM [21,119]	59.02%	2.60s	**36.50%**	2.09s

Table 5.8: The Comparison of Different Methods on Numeric Tasks.

Method	N_Emotion		
	MAE	*RMSE*	*Time*
Mean	**12.02**	**17.84**	0.09s
Median	13.53	21.26	0.11s
CATD [118]	16.36	25.94	2.15s
PM [21,119]	13.91	21.96	0.36s
LFC_N [161]	12.20	18.97	0.23s

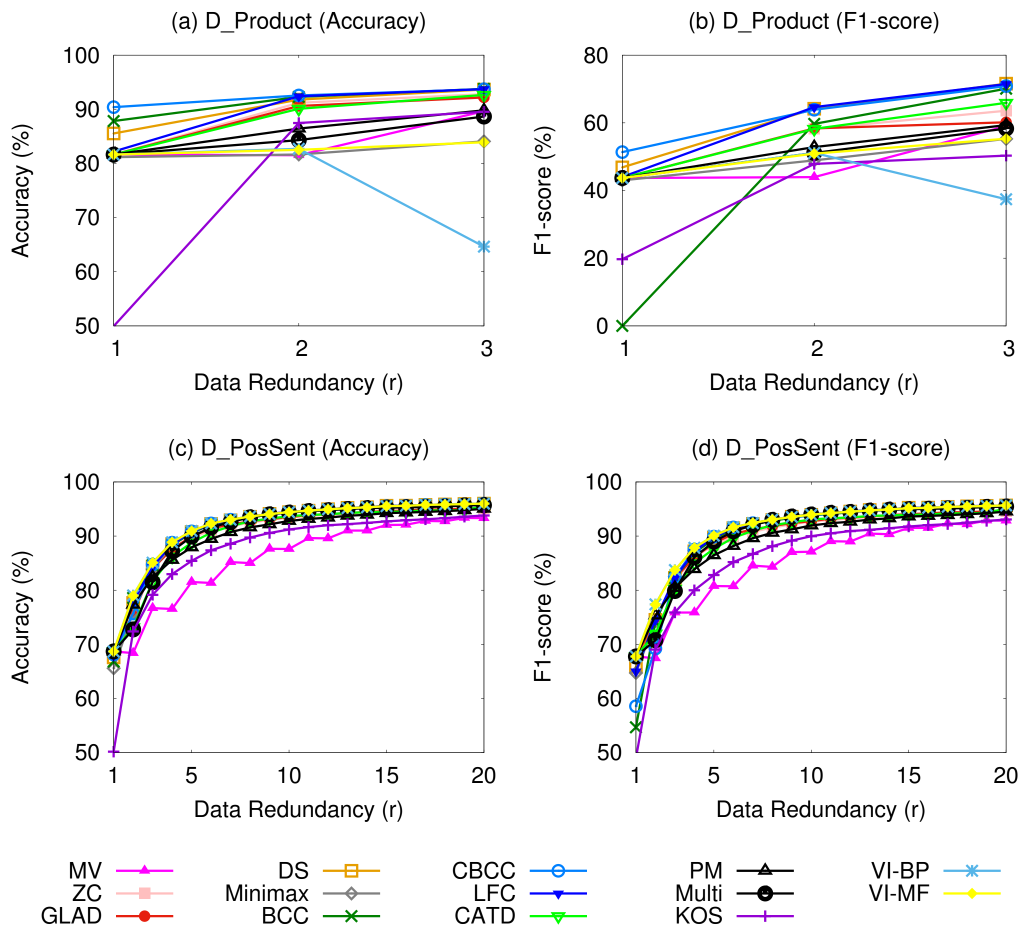


Figure 5.4: Quality Comparisons on Decision-Making Tasks.

with *r* for different methods. (2) In Table 5.6, it can be observed that for *Accuracy*, the quality does not make significant differences between methods (most methods' quality are around 90%); while for *F1-score*, it makes differences, and only 4 methods' quality (D&S, BCC, CBCC, LFC) are above 70%, leading more than 4% compared with other methods. We have analyzed in Section 5.6.1 that *F1-score* is more meaningful to `D_Product` compared with *Accuracy*, as we are more interested in finding out the "*same*" products. (3) In terms of task models, incorporating task difficulty (GLAD) or latent topics (Minimax) do not bring significant benefits in quality. (4) In terms of worker models, we can observe that the four

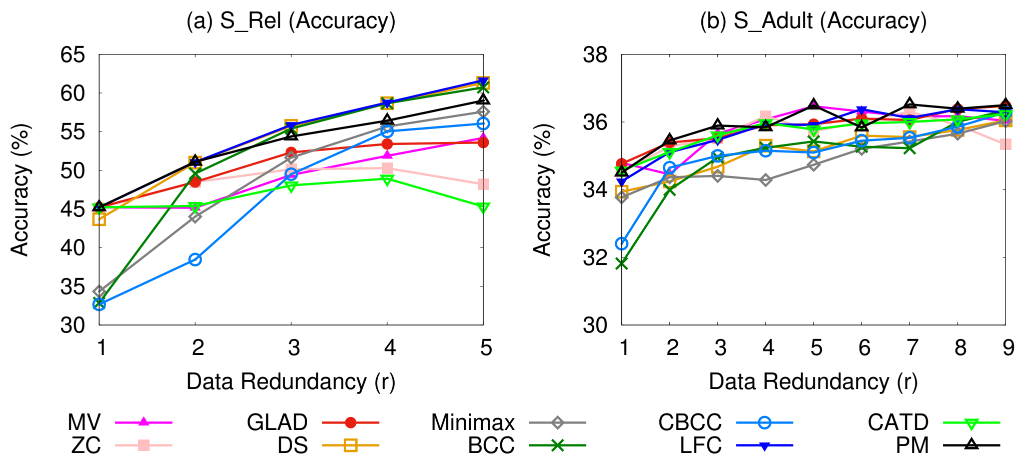


Figure 5.5: Quality Comparisons on Single-Label Tasks.

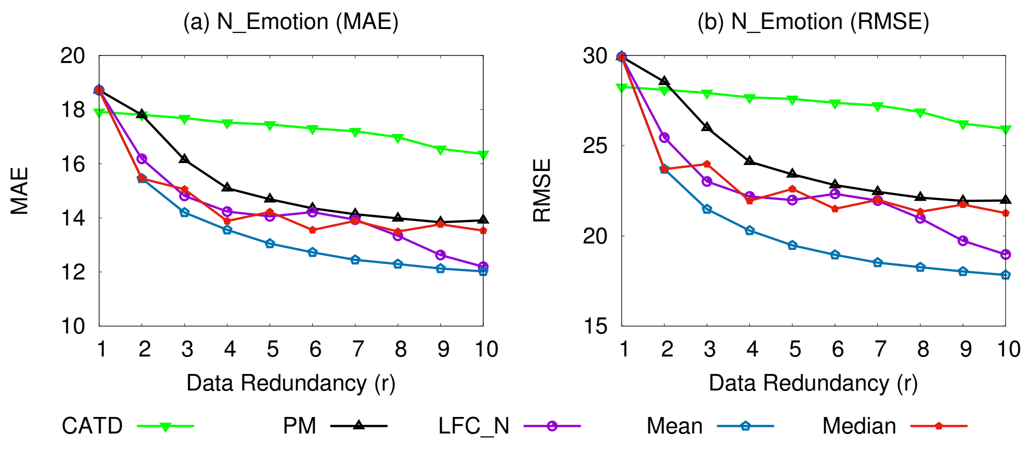


Figure 5.6: Quality Comparisons on Numeric Tasks.

methods with confusion matrices (i.e., D&S, BCC, CBCC, LFC) perform significantly better than other methods with worker probability. The reason is that confusion matrix models each worker as a 2×2 matrix q^w in decision-making tasks, which captures both $q^w_{1,1} = \Pr(v^w_i = \text{T} \mid v^*_i = \text{T})$, i.e., the probability that a worker w answers correctly if the truth is T and $q^w_{2,2} = \Pr(v^w_i = \text{F} \mid v^*_i = \text{F})$, i.e., the probability that w answers correctly if the truth is F. However, the worker probability models a worker as a single value, which substantially assumes that

$q^w_{1,1} = q^w_{2,2}$ in confusion matrix. This cannot fully capture a worker's answering performance. Note that in D_Product, typically workers have high values for $q^w_{2,2}$ and low values for $q^w_{1,1}$. Since for a pair of different products, if one difference is spotted between them, then it will be answered correctly, which is easy ($q^w_{2,2}$ is high); while for a pair of same products, it will be answered correctly only if all the features in the products are spotted the same, which is hard ($q^w_{1,1}$ is low). Although VI-BP and VI-MF also use confusion matrix, they perform bad, probably because they leverage *Variational Inference* to infer the parameters, which may derive workers' qualities wrongly. For other worker models such as worker bias (Multi), worker variance (Multi, LFC_N), diverse skills (Multi, Minimax) and confidence (CATD), they do not outperform the confusion matrix methods in quality, probably due to the fact that the methods cannot infer those parameters correctly. (5) For BCC, the *F1-score* is 0 as r = 1, since BCC returns all tasks as F. which gives no information to T. However, in Table 5.6 (with completed data), the method BCC performs the best in *Accuracy*, while the method D&S performs the best in *F1-score*.

For dataset D_PosSent, i.e., Figures 5.4(c),(d), it can be observed that (1) as r is varied in $[1, 10]$, the quality increases significantly with r for different methods (improving around 20%), and then have a minor increase ever since ($r \in [11, 20]$). (2) Similar to D_Product, the six methods with confusion matrix as worker models (i.e., D&S, BCC, CBCC, LFC, VI-BP and VI-MF) perform equally the best, since confusion matrix captures more information than worker probability. However, other methods with more complicated task models and worker models do not express their benefits in quality. (3) *Accuracy* and *F1-score* in D_PosSent do not have significant differences, since unlike D_Product, the #tasks with T and F as truth in D_PosSent (i.e., 528 and 472) is balanced.

Single-Label Tasks. In Figure 5.5 and Table 5.7, on datasets S_Rel and S_Adult, we compare with 10 methods that specifically address single-label tasks (Table 5.4), i.e., MV, ZC, GLAD, D&S, Minimax, BCC, CBCC, LFC, CATD, and PM. We have the following observations: (1) on S_Rel, in general, the quality of methods increase with r; while on S_Adult, the quality of methods increase with $r \in [1, 5]$, and keep stable for $r \geq 5$. (2) In terms of quality, on S_Rel, the three methods D&S, BCC and LFC with quality $\geq 60\%$ outperform the other methods; while on S_Adult, the performance of different methods are similar. (3) On S_Rel, the quality of methods CATD and ZC decrease as $r \geq 4$, probably because they are sensitive to low quality workers' answers. (4) The quality of methods for single-label tasks are lower than that for decision-making tasks, since workers are not good at answering tasks with multiple choices, and the methods for single-label tasks are sensitive to low quality workers.

Numeric Tasks. In Figure 5.6 and Table 5.8, we compare with 5 methods that specifically address numeric tasks (Table 5.4): CATD, PM, LFC_N, Mean and Median. Note that *MAE* and *RMSE* are defined as errors, and the lower, the better. We have the following observations: (1) generally the errors of almost all methods decrease with the increasing r. (2) Among all methods, the baseline method Mean performs best, which regards each worker as equal. This means that workers' qualities may not be accurately inferred in CATD, PM and LFC_N. (3) It can be seen that the methods for numeric tasks are not well-addressed, as only 3 methods (i.e., CATD, PM, LFC_N) are specifically devised for numeric tasks.

(2) The Efficiency of Different Methods.

In terms of efficiency, some methods (MV, Mean and Median) can infer the truth directly, while other existing works (ZC, GLAD, D&S, Minimax, BCC, CBCC, LFC, CATD, PM, Multi, KOS, VI-BP, VI-MF, LFC_N) follow the iterative framework (Algorithm 11) until convergence is attained. For the iterative methods, the time complexity can be expressed as $\mathcal{O}(c \cdot t)$, where c is the #iterations to converge, and t is the time in each iteration. Thus a method is inefficient if it takes many iterations to converge, or the time is slow in each iteration. We record each method's efficiency in Tables 5.6-5.8. We can observe that non-iterative methods (MV, Mean and Median) are finished within 1s. For iterative methods, (1) ZC, D&S, LFC,CATD, PM, LFC_N can finish within 15s, which is efficient. The reason is that they have a small c and t. (2) The methods BCC, CBCC, Multi, KOS and VI-MF take more than 15s, but less than 3min to finish the process. The reason is that for BCC, CBCC and VI-MF, they take many iterations c to converge; while Multi and KOS take a long time t in each iteration. (3) There are methods GLAD, Minimax and VI-BP that take up to 100min to finish, which is slow. The reason is that they solve an optimization function in each iteration, e.g., GLAD uses gradient descent [107], which will iteratively update parameters in each iteration, and it will take a lot time (i.e., t).

Summary. We summarize based on the above results. (1) The quality increases significantly with small data redundancy r, and keeps stable after a certain redundancy $\widehat{r} > r$. Note that $\widehat{r}$ varies in different methods on different datasets. (2) There is no method that performs consistently the best on all tested datasets. (3) In decision-making and single-label tasks, in general the three methods (D&S, BCC, LFC) perform better than others with complete data (Tables 5.6-5.11). Note D&S [47] is the most classical approach proposed in 1979, and all other methods (BCC and LFC) extend D&S in different perspectives. (4) In numeric tasks, they are not well-addressed in existing works, where the baseline method Mean performs best in `N_Emotion`. (5) In terms of task models, the methods that model task difficulty (GLAD) or latent topics (Multi) in tasks do not perform significantly better in quality; moreover, they often take more time to converge. (6) In terms of worker models, generally speaking, confusion matrix performs better in quality compared with worker probability; while other worker models (e.g., diverse skills, worker bias, variance and confidence) do not bring significant benefits. Not surprisingly, methods with complicated worker models often lead to inefficiency. (7) In terms of inference techniques, for effectiveness, the methods with Optimization and PGM are more effective than Direct Computation. For efficiency, Direct Computation is more efficient than Optimization and PGM since it can compute the truth directly.

Table 5.9: The Benefit (Δ) with Qualification Test on Decision-Making Tasks.

Method	D_Product (Simulation)		D_PosSent (Real)	
	Accuracy (Δ)	*F1-score* (Δ)	*Accuracy* (Δ)	*F1-score* (Δ)
ZC	92.95% (+0.15%)	64.4% (+0.81%)	95.10% (0.00%)	94.60% (0.00%)
GLAD	92.18% (–0.02%)	60.04% (–0.03%)	95.20% (0.00%)	94.71% (0.00%)
D&S	93.98% (+0.32%)	72.43% (+0.84%)	95.90% (–0.10%)	95.55% (–0.11%)
LFC	93.98% (+0.25%)	72.43% (+0.95%)	95.90% (–0.10%)	95.55% (–0.11%)
CATD	93.11% (+0.45%)	67.48% (+1.56%)	95.50% (+0.01%)	95.07% (+0.01%)
PM	90.55% (+0.74%)	61.26% (+1.92%)	95.10% (+0.05%)	94.60% (+0.06%)
VI-MF	85.26% (+1.35%)	57.31% (+2.00%)	95.90% (–0.10%)	95.54% (–0.12%)

Table 5.10: The Benefit (Δ) with Qualification Test on Single-Label Tasks.

Method	S_Rel (Simulation)	S_Adult (Simulation)
	Accuracy (Δ)	*Accuracy* (Δ)
ZC	55.24% (+7.03%)	35.54% (+0.20%)
GLAD	53.48% (–0.19%)	36.30% (+0.43%)
D&S	61.42% (+0.12%)	36.93% (+1.16%)
LFC	61.42% (+0.12%)	36.93% (+1.16%)
CATD	44.09% (–1.26%)	35.68% (–0.50%)
PM	59.41% (+0.39%)	36.70% (+0.41%)

Table 5.11: The Benefit (Δ) with Qualification Test on Numeric Tasks.

Method	N_Emotion (Simulation)	
	MAE (Δ)	*RMSE* (Δ)
CATD	17.97 (+1.61)	28.56 (+2.62)
PM	17.27 (+3.36)	27.42 (+5.46)
LFC_N	12.20 (0.00)	18.97 (0.00)

The Effect of Qualification Test

We next study how each method can be affected by qualification test. In real crowdsourcing platforms (e.g., AMT [1]), a fixed set of golden tasks can be set for each worker to answer when the worker first comes to answer tasks. We collect dataset `D_PosSent` from AMT [1], where each worker is required to answer 20 tasks with known ground truth (qualification test) when she first comes. However, in the other four public datasets, the data for qualification test are not made public (or not used in most cases). Thus, (1) we first *simulate* each worker's answers for qualification test; (2) then use each worker's answering performance for them to initialize the worker's quality (line 1 in Algorithm 11); (3) finally we run each method with the initialized worker's quality. For example, in `D_Product`, for each worker w, in her answers for all tasks ($\mathcal{T}^w$), we use bootstrap sampling [60], i.e., *sample with replacement* to sample 20 times, where each time we randomly sample her answer for one task (as there may be limited answers for a worker, thus bootstrap sampling is used, which can uncover the real distribution, i.e., worker's quality). Then we assume the 20 tasks' truth are known, and use her answering performance to initialize her quality.

To leverage a worker's answers for qualification test, for example, in ZC [48], as each worker is modeled as worker probability, then her quality is initialized as the fraction of correctly answered tasks in these 20 sampled ones. Finally the two steps in ZC are iteratively run until convergence, and the quality w.r.t. ground truth is computed. We find that there are only 8 methods (i.e., ZC, GLAD, D&S, LFC, CATD, PM, VI-MF and LFC_N) that can initialize workers' qualities using qualification test. For these methods, we repeat each experiment for 100 times. We denote $\widetilde{c}$ as the average quality with qualification test; c as the quality without qualification test (i.e., in Tables 5.6-5.8); and $\Delta = \widetilde{c} - c$ as the improvement of quality. Tables 5.9-5.11 show $\widetilde{c}$ and Δ of each method in each dataset. Note that the only difference between c and $\widetilde{c}$ is that they use different initializations of workers' qualities (line 1 in Algorithm 11).

Decision-Making & Single-Label Tasks. It can be observed that no matter in real qualification test (D_PosSent), or simulations (D_Product, S_Rel, S_Adult), not all methods can benefit from qualification test and the benefits vary in different datasets. For example, in D_Product and S_Adult, almost all methods can benefit; while in D_PosSent, only PM and CATD can benefit. The reason is that in D_Product, each task is only answered by 3 workers, while in D_PosSent, each task is answered by 20 workers. The dataset with small data redundancy (e.g., D_Product) requires qualification test for a good initialization of workers' qualities, while other datasets can correctly detect each worker's quality in an unsupervised manner. We can also observe that the benefit (Δ) is often small and sometimes $\Delta < 0$, since almost all methods adopt an iterative approach and approximate the objective value, thus an inadequate initialization may lead to a bad local optimum.

Numeric Tasks. For dataset N_Emotion, even no methods in CATD, PM and LFC_N can benefit, where both the errors *MAE* and *RMSE* increase for all methods. As mentioned before, this is probably because the methods are not studied properly in the numeric data, and the qualities modeled for workers are not accurate enough.

Summary. (1) Some methods can benefit from the qualification test with marginal benefit. (2) In numeric tasks, most of methods cannot benefit, and there is still room for improvement. (3) There are some methods that are hard to incorporate qualification test.

The Effect of Hidden Test

We evaluate how hidden test affects each method's quality. Given V, suppose we also know the set of golden tasks $\mathcal{T}' \subseteq \mathcal{T}$, then how much quality can be improved using existing methods? To implement this idea, we take a look at existing methods (Table 5.4) and observe that there are 9 methods (ZC, GLAD, D&S, Minimax, LFC, CATD, PM, VI-MF, and LFC_N) that can be easily extended

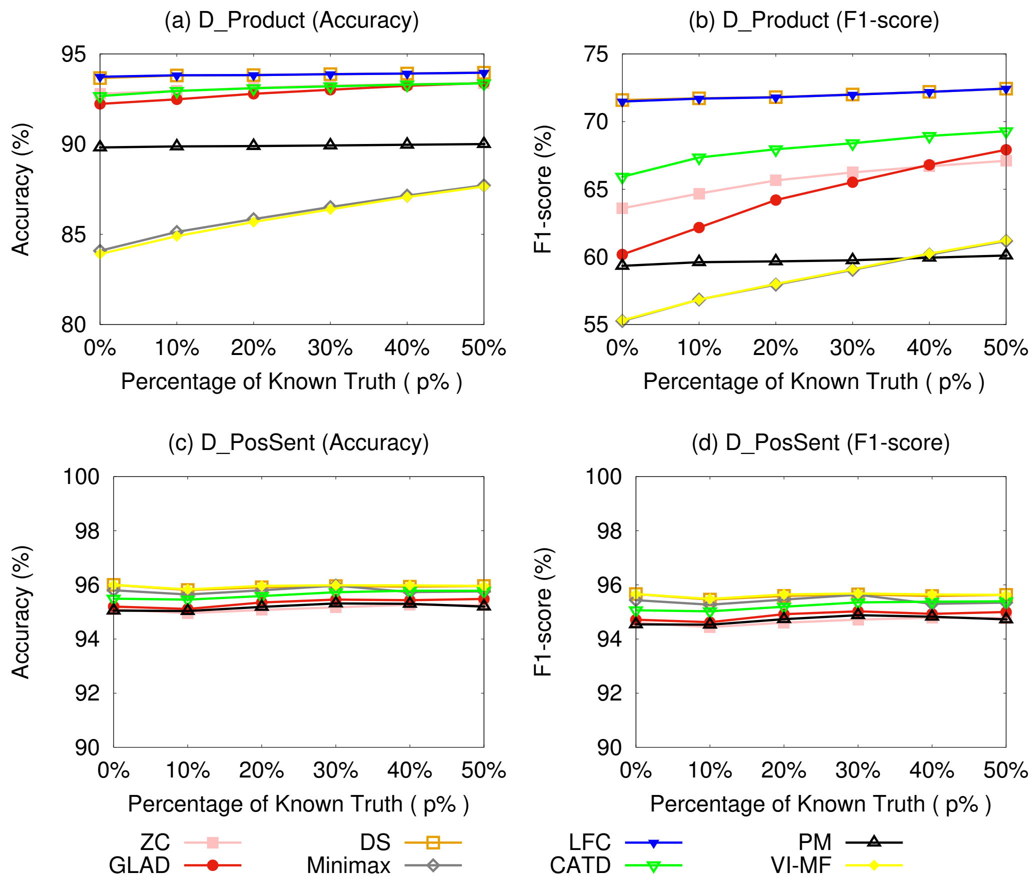


Figure 5.7: Varying Hidden Test on Decision-Making Tasks.

to incorporate the golden tasks into its iterative algorithm.

To incorporate the golden tasks, consider Algorithm 11, in step 1, we only update the truth of tasks with unknown truth; in step 2, we update each worker's quality by considering both the truth of golden tasks and other tasks' inferred truth in step 1. We show the quality of different methods by varying the size of golden tasks ($\mathcal{T}'$) in Figures 5.7, 5.8 and 5.9. For example, in Figure 5.7(a), on dataset `D_Product`, we randomly select $p\%$ in the task set $\mathcal{T}$ as the golden tasks ($\mathcal{T}'$). Then we take $\mathcal{T}'$ and workers' answers V as the input to different methods, and further test different methods' quality by comparing the inferred truth of $\mathcal{T} - \mathcal{T}'$ with their ground truth. We vary $p \in [0, 50]$, where

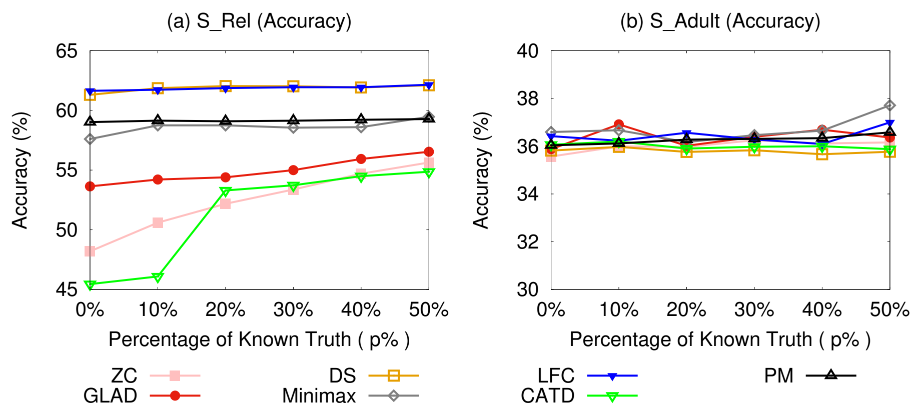


Figure 5.8: Varying Hidden Test on Single-Label Tasks.

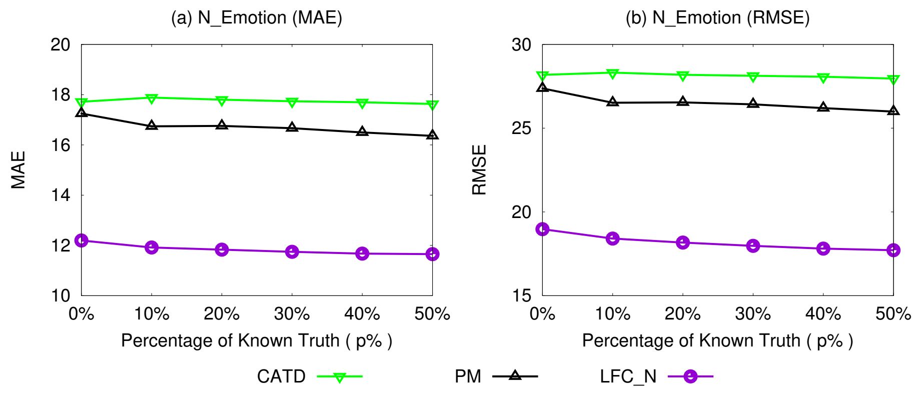


Figure 5.9: Varying Hidden Test on Numeric Tasks.

for each p, we repeat each experiment 100 times and record the average quality. Next, we analyze the results.

Decision-Making & Single-Label Tasks. In Figures 5.7 and 5.8, it can be seen that (1) generally starting from $p = 0$ (Tables 5.6 and 5.10), the quality of methods increase with p, since knowing more truth is beneficial to more accurate estimation. (2) The methods on dataset `D_PosSent` do not have significant gains with varying p, since each task is answered by multiple workers, and the inferred parameters are hard to be affected by the known truth. (3) Only a few methods (e.g., ZC, CATD) are sensitive to golden tasks, since most iterative methods are easy to fall into local optimum.

Numeric Tasks. In Figure 5.9, we compare with three methods (LFC_N, CATD, PM) in `N_Emotion` and we find that the errors (*MAE* and *RMSE*) decrease slightly with the increasing p.

Summary. (1) Generally the quality of different methods increase with more proportion (p%) of golden tasks. (2) Different methods have different improvements on different datasets. (3) Only 9 methods are easy to incorporate golden tasks in them.

Analyzing Different Factors in Truth Inference

Task Types. In terms of task types, we observe that the methods for decision-making tasks have been studied a lot and already have very good performance. Compared with decision-making tasks, the methods for single-label tasks do not perform well, e.g., the qualities for `S_Rel` and `S_Adult` (Table 5.7) are much lower than those for `D_Product` and `D_PosSent` (Table 5.6), since the methods for single-label tasks are more sensitive to workers with low qualities. For numeric tasks, the methods are not studied very well, and even the baseline method Mean outperforms others in dataset `N_Emotion`. This is because that on one hand, few methods specifically study numeric tasks; on the other hand, most methods cannot estimate workers' qualities for numeric tasks accurately.

Task Models. In terms of task models, GLAD and Minimax are the only two methods (Table 5.4) that consider specific task models (task difficulty and latent topics, respectively). However, they do not show improvements in quality compared with other methods with no task models. Moreover, they often take a long time to converge (e.g., $>$ 1000s in `S_Rel` and `S_Adult`). This is probably due to that their inference methods are not robust, and in some cases they cannot estimate the parameters in task models accurately. The incorporation of task models also leads to inefficiency.

Worker Models. In terms of worker models, in general, methods with confusion matrix (D&S, BCC, CBCC, LFC, VI-BP, VI-MF) perform better than methods with worker probability (ZC, GLAD, CATD, PM, KOS), since confusion matrix is more expressive than worker probability. Note that the quality of methods also vary a lot even if they apply the same worker model, e.g., for confusion matrix, the methods D&S, BCC, LFC are more robust than others (CBCC, VI-BP, VI-MF), since their techniques can infer workers' qualities more accurately. For other worker models, e.g., worker bias (Multi), worker variance (Multi, LFC_N), diverse skills (Multi, Minimax) and confidence (CATD), they do not achieve higher gains in quality. We also observe that not necessarily "*the more complex the model is, the higher quality the method will achieve*". For example, although Multi considers diverse skills, worker bias and variance in its worker models, the quality does not bring significant benefits. Ideally more complicated worker models lead to much higher quality; however, this introduces more computational complexity, and on one hand, it is challenging to estimate a large set of parameters accurately; on the other hand, it is hard to converge. Thus most methods with complicated worker models fail to achieve very good performance in quality and efficiency.

Inference Techniques. We analyze the techniques from quality, efficiency and interpretability, respectively. (1) In terms of quality, the methods with Optimization and PGM are more effective than the methods with Direct Computation, as they consider more parameters and study how to infer them iteratively. (2) In terms of efficiency, methods with Optimization and PGM are less efficient than methods with Direct Computation. Different optimization functions often vary significantly in efficiency, e.g., *Bayesian Estimator* is less efficient than *Point Estimation*, and some techniques (e.g., *Gibbs Sampling*, *Variational Inference*) often take a long time to converge. (3) In terms of interpretability, Optimization is easier to understand. The reason is that people can interpret the relations between worker's quality and task's truth in the self-defined optimization function. For PGM, it should conform to the model (Figure 5.1), which gives less freedom to express the optimization function. Moreover, it is hard to devise an easily solvable optimization function, and the developed iterative algorithms often lead to local optimum (e.g., Expectation Maximization [47,49]).

Suggestions

Based on the experimental results, we have the following suggestions.

Decision-Making & Single-Label Tasks. If one has sufficient workers' answers (e.g., with redundancy over 20), and wants a very simple implementation that attains reasonable results, then we recommend the baseline method, i.e., MV; if one wants an implementation with little overhead but attains very good results, then we recommend the classical method D&S [47], which is robust in practice; if one would like to try some extensions of D&S, then BCC [104] and LFC [161] are good choices; if one wants to learn more inference techniques and incorporate various task/worker models, we recommend the PGM method Multi [197] and the Optimization method Minimax [226].

Numeric Tasks. If one has sufficient workers' answers, we recommend the baseline method (i.e., Mean); if one wants to learn more advanced techniques and worker models, we recommend the PGM method LFC_N [161], and the Optimization method CATD [118].

5.7 Chapter Summary

We provide a detailed survey on truth inference problem in crowdsourcing and perform an in-depth analysis of 17 existing methods. We summarize a framework (Algorithm 11) and analyze the *task types*, *task models*, *worker models* and *inference techniques* in these methods. We also conduct sufficient experiments to compare these methods on various datasets with varying task types and sizes.

In the above three chapters, we have focused on studying the techniques in task assignment and truth inference components. In the next two chapters, we will study how to combine the techniques and benefit the real-world applications, i.e., question answering application and image tagging application, respectively. On one side, directly applying the techniques to these applications are not satisfactory. For example, in question answering application, each task may be related to various domains and each worker may have diverse qualities over different domains. The traditional techniques in task assignment and truth inference that omitted such factors may not work well in practice. On the other side, the existing methods and principles in task assignment and truth inference will inspire us in designing intuitive solutions in real-world applications.

Chapter 6

A Multi-Label Task Crowdsourcing System

6.1 Introduction

Crowdsourcing solutions have been proposed to solve problems that are hard for computers (e.g., sentiment analysis [127, 222], entity resolution [192, 199]). Consider a sentiment analysis problem: a product company, e.g., Amazon, collects many reviews from users on its products, and it aims to know users' sentiments about the reviews. Existing algorithms often cannot compute sentiments accurately [125]. Crowdsourcing solutions can be used, where for each review, we generate a task and ask the crowd to label the sentiment (e.g., selecting a label from "*positive*", "*neutral*" or "*negative*").

Existing crowdsourcing studies [92, 127, 199, 211, 222] focus mainly on single-label tasks, which require workers to select a single label (or choice), e.g., select one label from {*positive*, *neutral*, *negative*} in a sentiment analysis task. However, an object can have multiple labels. For example, in image tagging application, an image in Figure 6.1 has *tree*, *sky*, and *mountain* as labels. Moreover, for movie tagging, a movie *Matrix* can be labeled with *action* and *sci-fi*;

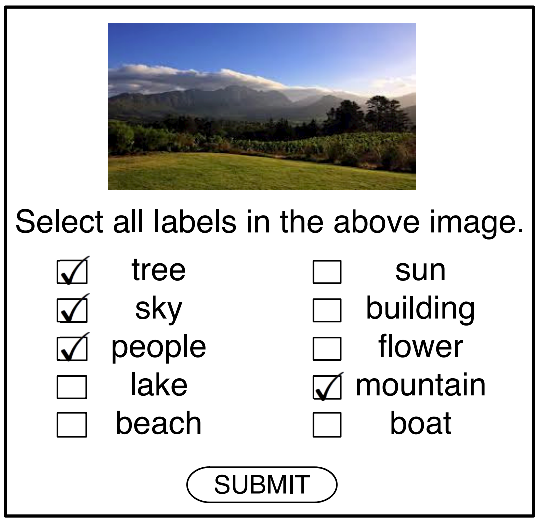


Figure 6.1: An Example Multi-Label Task.

similarly, a person *Barack Obama* can be labeled with a *president*, *lawyer*, and *politician*. Although we can transform a multi-label task to several single-label tasks, this simple approach can generate many tasks, incurring a high cost and latency. For example, the multi-label task in Figure 6.1 is transformed to 10 single-label tasks, where each task inquiries about whether or not the image contains a certain label (e.g., *tree*). As reported in [51], compared with single-label tasks, multi-label tasks enable six times of improvement in terms of human computation time, without sacrificing much quality.

Although some recent works [29, 56, 145, 147, 149, 198] focus on solving multi-label tasks in crowdsourcing, however, this problem is not well addressed. Workers may have different characteristics in multi-label tasks: a conservative worker would only select labels that the worker is certain of, while a venturous worker may select more labels. In order to determine the multi-label tasks' results, the key is to devise the so-called "*worker model*" to accurately express the behavior of the worker in answering multi-labels. Then we can identify the most trustworthy workers based on their models and put trust in their answers. However, existing works [29, 56, 145, 147, 149, 198] simply adapt the worker models in modeling workers' behaviors for single-label tasks [48, 92, 127, 149, 211, 222], which are not suitable in our scenario. In a multi-label task, there can be multiple labels for the worker. Consider the case where there are 20 labels, and only four of them are correct (i.e., the remaining 16 labels are wrong). A "bad" worker may select none of the correct labels, and instead submit a few (say two) wrong labels. Under the *true negative rate* (TNR) [92,149,222], his quality score is $(16 - 2)/16 = 87.5\%$. But the extremely high score of this worker cannot reflect his poor performance! The consequence is that we may be misled to believe in a bad answer. As we will explain, *none* of the existing worker models are suitable for multi-label tasks.

Furthermore, different from single-label tasks, the inherent correlations among labels exist in multi-label tasks. For example, in the 10 labels of Figure 6.1, consider one pairwise label dependency: if an image has label *sun*, then it is highly probable that it also has label *sky*. Similar correlations also hold for labels *boat* and *lake*. Label correlations can be regarded as *prior* information, which can be learned from existing machine learning works [24, 210, 213]. The label correlations can be used to improve the inference quality.

To address these limitations, in this chapter we perform a comprehensive investigation for the multi-label tasks. We develop a system called Comet (with its framework in Figure 6.2), where task publisher(s) can deploy multi-label tasks on our system. Comet interacts with crowdsourcing platforms, e.g., Amazon Mechanical Turk (AMT) [1], from which it collects workers' answers, stores them in database and conducts inference. It can also wisely select tasks to assign upon a worker's request. In summary, there are two kinds of requests from workers that need to be handled by Comet: (1) when a worker submits answers to Comet; (2) when a worker comes and requests tasks. We develop two components in Comet that process these two requests, respectively:

• **Truth Inference.** When a worker submit answers, the component infers the truth (or correct labels) of each task based on all workers' answers. Comet uses

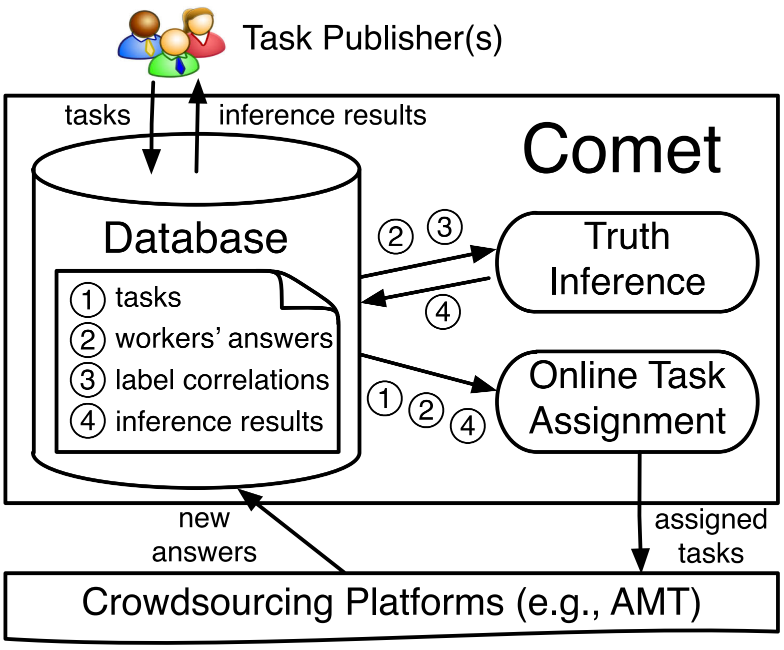


Figure 6.2: The Comet Framework.

a novel worker model, which can capture workers' diverse characteristics in answering multi-label tasks. As the truth of each task is unknown, each worker's model can only be estimated based on the collected answers. Comet conducts truth inference in an iterative approach, which can jointly infer all tasks' truth and workers' models with the following principle: a worker that selects correct labels often will be considered to have a higher quality; meanwhile, a label that is selected by high quality workers for a task is likely to be a correct label for the task. We also study how to speed-up the computation by designing an incremental approach. Comet leverages the known label correlations to improve the truth inference, by integrating them into the inference method.

• **Online Task Assignment.** When a worker requests tasks, the component targets at instantly assigning k tasks to the worker. A poor assignment may not only waste the budget and time, but also spoil the overall quality. Comet first measures the uncertainty of each task based on the collected answers, and then estimates how much uncertainty can be reduced *if* the task is really answered by the worker. Finally the k tasks with the highest reduction in uncertainty will be assigned. As the worker's answer to the task is unknown, to compute its reduction in uncertainty, all possible answers given by the worker should be considered, which is exponential (e.g., 2^ℓ answers for ℓ labels). Moreover, to select k tasks (say, out of n tasks), we have to consider all $\binom{n}{k}$ combinations. To reduce the computational complexity, we prove a theorem, which computes the optimal assignment in linear time.

To summarize, our contributions are:

• We perform a comprehensive study on crowdsourcing multi-label tasks, by addressing two fundamental problems: *Truth Inference Problem* and *(Online) Task Assignment Problem* (Section 6.2).

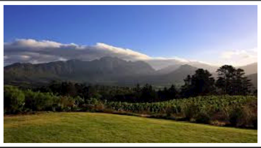
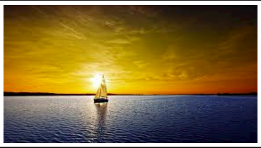
• For the first problem, we propose an effective worker model (Section 6.3), and devise a method that jointly infers each task's truth and each worker's model (Section 6.4). We further consider how to integrate label correlations into our method (Section 6.5);

• For the second problem, we develop an effective algorithm that judiciously selects k tasks with the largest amount of uncertainty reduction for the current worker, in linear time (Section 6.6);

• We develop Comet, and use two real-world datasets to perform experiments on two crowdsourcing platforms (AMT [1] and ChinaCrowd [3]). Results show that Comet outperforms existing state-of-the-art methods, and is robust under various scenarios. It achieves over 20% improvements on the two datasets performed by low-quality workers. We also conduct experiments on simulated data, in order to verify the scalability of Comet (Section 6.7).

For an overview of the chapter, we define two problems: *Truth Inference Problem* and *Task Assignment Problem* in Section 6.2. Then we address the two problems in Sections 6.3-6.5 and Section 6.6, respectively. We perform experi-

Table 6.1: An Example of Objects and Candidate Label Sets.

object	o_1= 	o_2= 
candidate label set	L_1={tree, sky, people, lake, beach, sun, building, flower, mountain, boat}	L_2={tree, sky, people, lake, beach, sun, building, flower, mountain, boat}
correct labels	{tree, sky, mountain}	{sky, lake, sun, boat}

ments in Section 6.7, and review related works in Section 6.8. Finally, we conclude in Section 6.9.

6.2 The Multi-Label Problem

6.2.1 Data Model

Definition 6.1 (Object, Candidate Label Set)**.** *Given n objects, $o_1, o_2, \cdots, o_n$, where each object o_i has a candidate label set L_i=$\{\ell_{i,1}, \ell_{i,2}, \ldots, \ell_{i,|L_i|}\}$, our target is to select the correct labels of each object o_i from its candidate label set L_i.*

For example, Table 6.1 shows two objects o_1, o_2 and their corresponding candidate label sets L_1 and L_2. Note that there are several options to generate candidate labels: (1) they can be collected from user data, e.g., user image tags in Flickr [8]; (2) we can leverage existing computer vision system (e.g., ConvNet [4]), where given an image, it can estimate the probability of each label being the correct label among a large set of labels, and we can keep the labels with high probabilities as our candidate labels. The correct labels for o_1 and o_2 are {`tree`, `sky`, `mountain`} and {`sky`, `lake`, `sun`, `boat`}, respectively. (Note that the correct labels are shown for illustration.) To effectively obtain the correct labels of each object, we resort to crowdsourcing, by asking workers to select labels of each object from its candidate label set.

Table 6.2: Workers' Answers for Objects.

object	**worker**	**answer**
o_1	w_1	{sky}
o_1	w_2	{tree, sky, people, mountain}
o_1	w_3	{tree, flower}
o_1	w_4	{tree, sky, flower, mountain}
o_2	w_1	{sky, boat}
o_2	w_2	{sky, beach, sun}
o_2	w_3	{lake, sun, building}

Table 6.3: Truth and Votes for All (Object, Label) Pairs.

(object, label)	**truth**	**all workers' votes**
$(o_1, \ell_{1,1})$	$t_{1,1} = \text{Y}$	$v_{1,1}^{w_1} = \text{N}$, $v_{1,1}^{w_2} = \text{Y}$, $v_{1,1}^{w_3} = \text{Y}$, $v_{1,1}^{w_4} = \text{Y}$
$(o_1, \ell_{1,2})$	$t_{1,2} = \text{Y}$	$v_{1,2}^{w_1} = \text{Y}$, $v_{1,2}^{w_2} = \text{Y}$, $v_{1,2}^{w_3} = \text{N}$, $v_{1,2}^{w_4} = \text{Y}$
$\cdots$	$\cdots$	$\cdots$
$(o_1, \ell_{1,10})$	$t_{1,10} = \text{N}$	$v_{1,10}^{w_1} = \text{N}$, $v_{1,10}^{w_2} = \text{N}$, $v_{1,10}^{w_3} = \text{N}$, $v_{1,10}^{w_4} = \text{N}$
$(o_2, \ell_{2,1})$	$t_{2,1} = \text{N}$	$v_{2,1}^{w_1} = \text{N}$, $v_{2,1}^{w_2} = \text{N}$, $v_{2,1}^{w_3} = \text{N}$
$\cdots$	$\cdots$	$\cdots$
$(o_2, \ell_{2,10})$	$t_{2,10} = \text{Y}$	$v_{2,10}^{w_1} = \text{Y}$, $v_{2,10}^{w_2} = \text{N}$, $v_{2,10}^{w_3} = \text{N}$

Definition 6.2 (Task, Worker, Answer). *A task contains an object o_i and its candidate label set L_i, and it asks the workers to select correct labels for o_i from L_i. (As there exists a 1-to-1 correspondence between tasks and objects, we use task and object interchangeably.) Let $\mathcal{T} = \{(o_1, L_1), (o_2, L_2), \ldots, (o_n, L_n)\}$ denote the set of tasks. To tolerate errors from the workers, each task can be answered by multiple workers. We denote $\mathcal{W}$ as the set of all workers, and W_i as a set of workers that answer object o_i.*

Example 13. *Based on Table 6.1, we can generate two tasks $\mathcal{T} = \{(o_1, L_1), (o_2, L_2)\}$. We show the first task in Figure 6.1, which contains an image (o_1) and a set of 10 labels (L_1). Workers will identify which labels the image has, by ticking '$\checkmark$' to the corresponding labels. Table 6.2 shows workers' answers, where each row represents a worker's selected labels (or answer) for an object. For example, the first row indicates that worker w_1 only selects* {`sky`} *for o_1. We can see that o_1 is answered by all 4 workers, and o_2*

is answered by 3 workers. Thus $\mathcal{W} = \{w_1, w_2, w_3, w_4\}, W_1 = \{w_1, w_2, w_3, w_4\}$, and $W_2 = \{w_1, w_2, w_3\}$.

Next, for each pair (o_i, $\ell_{i,j}$), we define its collected answers (called votes) from workers and its ground truth (called truth).

Definition 6.3 (Vote, Truth). *For a pair (o_i, $\ell_{i,j}$) ($1 \le i \le n$, $1 \le j \le |L_i|$), a vote $v^w_{i,j} = Y/N$ represents whether or not worker w selects label $\ell_{i,j}$ for o_i. To be specific, $v^w_{i,j} = Y$ (N) means that worker w selects (does not select) label $\ell_{i,j}$ for object o_i. For a pair (o_i, $\ell_{i,j}$) ($1 \le i \le n$, $1 \le j \le |L_i|$), we denote its truth as $t_{i,j} = Y/N$, which represents whether or not label $\ell_{i,j}$ is a correct label for object o_i, i.e., $t_{i,j} = Y$ (N) means that label $\ell_{i,j}$ is (not) a correct label for object o_i.*

Example 14. *Consider the two tasks in Table 6.1 and workers' answers in Table 6.2, we generate a new Table 6.3, which lists the truth and votes for all (o_i, $\ell_{i,j}$) pairs where $1 \le i \le n$ and $1 \le j \le |L_i|$. In each row, for a given pair (o_i, $\ell_{i,j}$), we get its truth $t_{i,j}$ from Table 6.1 and all its votes from Table 6.2. To make it simple, we adopt the label sequence of L_1 (L_2) as Table 6.1, i.e., $\ell_{1,1} =$ `tree`, $\ell_{1,2} =$ `sky`, ..., $\ell_{1,10} =$ `boat`. For example, for the pair $(o_1, \ell_{1,2})$: its truth $t_{1,2} = Y$, which means that label `sky` ($\ell_{1,2}$) is a correct label for o_1; it gets 4 votes from workers, e.g., $v^{w_3}_{1,2} = N$, which means that worker w_3 does not select `sky` ($\ell_{1,2}$) for o_1.*

6.2.2 Problem Definition

We now define the problems. The first important problem is to infer the truth (or correct labels of each object) based on workers' answers.

Definition 6.4 (Truth Inference Problem). *Given all workers' votes, i.e., $v^w_{i,j}$ for $1 \le i \le n$, $1 \le j \le |L_i|$, $w \in W_i$, infer the truth of each $(o_i, \ell_{i,j})$ pair, i.e., $t_{i,j}$ for $1 \le i \le n$, $1 \le j \le |L_i|$.*

To address this problem, we first propose a worker model to quantify a worker's quality on a task (Section 6.3), and then devise an inference method

Table 6.4: Notations Used in Chapter 6.

Notation	Description
o_i	the i-th object ($1 \le i \le n$)
L_i	$\{\ell_{i,1}, \ell_{i,2}, \dots, \ell_{i,\lvert L_i \rvert}\}$, or the candidate label set w.r.t. o_i
$\mathcal{T}$	$\{(o_1, L_1), (o_2, L_2), \dots, (o_n, L_n)\}$, or the set of tasks
$\mathcal{W}$	a set of workers that have answered tasks
W_i	a set of workers that have answered o_i
$v_{i,j}^{w}$	Y (N) means worker w labels (does not label) $\ell_{i,j}$ for o_i
$t_{i,j}$	Y (N) means $\ell_{i,j}$ is (not) a correct label for o_i
p_w, r_w	worker w's model, or the *Precision*, *Recall* for worker w
$V_{i,j}$	a set of votes for the pair $(o_i, \ell_{i,j})$
V	$\{V_{i,j} \mid 1 \le i \le n, 1 \le j \le \lvert L_i \rvert\}$, or the vote set
$q_{i,j}$	$\Pr(t_{i,j} = \text{Y} \mid V)$, or the (probabilistic) truth of $(o_i, \ell_{i,j})$
D_w	a set of votes given by worker w
m	$m = \sum_{i=1}^{n} \lvert L_i \rvert$, or the number of all $(o_i, \ell_{i,j})$ pairs

to infer the truth of each task based on the worker model (Section 6.4). We discuss how to utilize the label correlation to further improve the inference quality (Section 6.5).

The second problem studies how to judiciously select appropriate tasks to assign for the coming worker.

Definition 6.5 (Task Assignment Problem). *Given all workers' answers for objects collected so far, when a worker comes, which k objects should be assigned to the coming worker?*

We propose an effective algorithm that selects k tasks for each worker in linear time (Section 6.6).

Table 6.4 summarizes the notations used in the chapter.

Table 6.5: Contingency Table of A Worker.

	$t = \mathrm{Y}$	$t = \mathrm{N}$
$v = \mathrm{Y}$	True Positives (TP)	False Positives (FP)
$v = \mathrm{N}$	False Negatives (FN)	True Negatives (TN)

6.3 Worker Modeling

In this section, we first revisit different worker models (Section 6.3.1), and then point out the limitations of models used in existing works and propose our selection (Section 6.3.2).

To evaluate a worker's model (or quality), we should know the truth for all (object, label) pairs that the worker's votes. So in this section we assume that all truth have been known, and compare different ways of modeling a worker. In the next section (Section 6.4) we show how to compute each worker's model without knowing the truth.

6.3.1 Worker Models Revisited

For each worker, based on all (object, label) pairs that the worker votes, and their truth, we can generate a contingency table for the worker in Table 6.5. In the table, v represents the given votes and t represents their truth, and both can be obtained from Table 6.3. Then for each worker, based on Table 6.3, we can compute four statistics: True Positives (TP), False Positives (FP), False Negatives (FN) and True Negatives (TN). E.g., TP of a worker is the number of (object, label) pairs that the worker votes Y and the truth is Y.

Example 15. *We show how to compute the contingency table for worker w_3 based on Table 6.3. As worker w_3 has answered 2 objects, by giving votes Y for 5 pairs and votes N for 15 pairs. So TP+FP=5 and FN+TN=15. Among the five Y votes:* `tree,flower` *for o_1, and* `lake,sun,building` *for o_2 (i.e., $v_{1,1}^{w_3} = v_{1,8}^{w_3} = v_{2,4}^{w_3} = v_{2,6}^{w_3} = v_{2,7}^{w_3} = Y$), the truth of 3 labels are Y (i.e., $t_{1,1} = t_{2,4} = t_{2,6} = Y$), thus TP=3, FP=2. Similarly we*

Table 6.6: Each Worker's Contingency Table.

	TP	FP	FN	TN
w_1	3	0	4	13
w_2	5	2	2	11
w_3	3	2	4	11
w_4	3	1	0	6

Table 6.7: Workers' Qualities.

	Accuracy	*Precision*	*Recall*	TNR
w_1	80%	100%	43%	100%
w_2	80%	71%	71%	85%
w_3	70%	60%	43%	85%
w_4	90%	75%	100%	86%

can get FN=4 and TN=11 for worker w_3. We list all four workers' contingency-table values in Table 6.6. Note that as worker w_4 only answers o_1, so its contingency table is computed based on worker w_4's votes for o_1.

Based on the contingency table for a worker, there are several derivative parameters to model a worker's quality: *Accuracy*, *Precision*, *Recall* and *True Negative Rate* (TNR), defined as follows:

• ***Accuracy***: It is the probability that the worker's vote to an (object, label) pair is right: $\Pr(\ t = v\ ) = \frac{\text{TP+TN}}{\text{TP+FP+FN+TN}}$.

• ***Precision***. It is the probability that the worker's vote Y to an (object, label) pair is right (i.e., its truth is Y): $\Pr(\ t = \text{Y} \mid v = \text{Y}\ ) = \frac{\text{TP}}{\text{TP+FP}}$.

• ***Recall***. It is the probability that an (object, label) pair with truth Y is rightly voted by the worker (i.e., the worker's vote is Y): $\Pr(\ v = \text{Y} \mid t = \text{Y}\ ) = \frac{\text{TP}}{\text{TP+FN}}$.

• ***True Negative Rate* (TNR)**. It is the probability that an (object, label) pair with truth N is rightly voted by the worker (i.e., the worker's vote is N): $\Pr(\ v = \text{N} \mid t = \text{N}\ ) = \frac{\text{TN}}{\text{TN+FP}}$.

Based on a worker's contingency table in Table 6.6, the worker's different quality parameters can be derived in Table 6.7. For these parameters, existing works [92, 127, 149, 211, 219, 222] either use only one or a combination of them to model a worker's quality: *Accuracy* is used in [127, 211, 219] to model a worker's quality, while [92, 149, 222] use a combination of TNR and *Recall* to model a worker's quality. In the following section (Section 6.3.2), we first show the limitations of *Accuracy* and TNR used in existing works, and then propose our novel way of modeling a worker (i.e., *Precision* and *Recall*).

6.3.2 Selecting Worker Model

Limitations of *Accuracy* and TNR. The limitation of only using Accuracy to model a worker in [127,211,219] is that a single parameter cannot fully express a worker's characteristics. For example, in Table 6.2 we know that three workers answer o_2, and label `boat` gets one Y vote from w_1 and two N votes from w_2, w_3. Consider the three workers' Accuracy in Table 6.7, as w_1 and w_2 (with the same *Accuracy*) give contradictory votes, while w_3 (with *Accuracy* 70%) votes N, so label `boat` will be decided as not in o_2 if we only consider *Accuracy* of each worker.

Some works [92, 149, 222] model a worker's quality with two parameters: TNR and *Recall*. The limitation of TNR is that it is not expressive of workers in answering multi-label tasks, because TNR =TN/(TN+FP), and in multi-label tasks, the incorrect labels are large in size and sparsely distributed in semantics. Then the crowd workers will not select most of them, making TN much larger compared with FP (Table 6.6), resulting in a high value for TNR. As can be seen in Table 6.7, the TNR for all workers are very high ($\geq$ 85%), and the TNR of three workers (w_2, w_3, and w_4) are very close, making it hard to distinguish between different workers.

More observations can be found in Figure 6.3, which shows the histograms of 141 workers' respective qualities collected from real-world datasets in crowd-

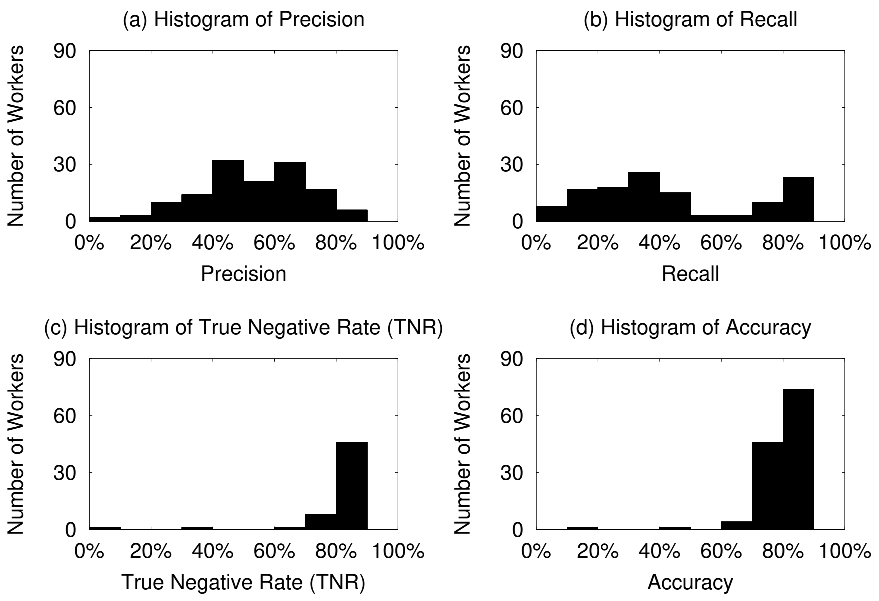


Figure 6.3: Histograms of Workers' Respective Qualities.

sourcing platforms (Section 6.7). In each figure, x-axis indicates respective quality value, and y-axis indicates the number of workers (in 141 ones) that fall in different ranges of quality values. It can be seen that the *Accuracy* and TNR are highly concentrated in high values and of a low spread, while *Precision* and *Recall* are of a wider spread and can capture the differences between workers. The reason is that *Accuracy* and TNR are related to TN, which dominates other three statistics: TP, FP and FN (also shown in Table 6.6); while *Precision* and *Recall*, two parameters unrelated to TN are more expressive in modeling a worker.

Our Selection: *Precision* and *Recall*. Based on the above analysis, we model the quality of a worker using two parameters: *Precision* and *Recall*, and the combination has not been considered in existing crowdsourcing works. Specifically, a worker with high *Precision* (e.g., w_1) means that the labels selected by the worker is likely to be correct (e.g., label `boat` for o_2); a worker with high *Recall* (e.g., w_4) means that the correct labels are mostly selected by the worker, which substantially indicates that a label not selected by the worker is unlikely to be correct.

Moreover, a conservative worker would only select labels that the worker is certain of, yielding *high Precision* but *low Recall* (such as w_1); on the other hand, a venturous worker may select more labels, resulting in *low Precision* but *high Recall* (such as w_4).

6.4 Iterative Truth Inference

This section studies the *Truth Inference Problem*, i.e., inferring the truth based on all workers' votes. We use *Precision* and *Recall* to model a worker's quality. We first introduce our general principle, which captures the inherent relation between the truth and workers' models (Section 6.4.1). We then apply the principle to developing an iterative method that solves the problem (Section 6.4.2). Finally we talk about how to speed up online computation (Section 6.4.3).

6.4.1 General Principle

The truth and workers' models have an inherent relation: if a label is selected by high-quality workers for an object, then the label is likely to be a correct label for the object; meanwhile, if a worker selects correct labels often, then the worker will be assigned with a high quality.

To apply the principle, we treat the truth and worker model as two sets of parameters, and compute them in an iterative way. To be specific, we denote the truth of each $(o_i, \ell_{i,j})$ $(1 \le i \le n, 1 \le j \le |L_i|)$ as $q_{i,j} \in [0,1]$, which is the probability that label $\ell_{i,j}$ is a correct label for o_i given all workers' votes (denoted as V, will clarify later), i.e., $q_{i,j} = \Pr(t_{i,j} = \mathrm{Y} \mid V)$; we denote the model of each worker w $(w \in \mathcal{W})$ as *Precision* $p_w \in [0,1]$ and *Recall* $r_w \in [0,1]$. Inspired by the Expectation-Maximization (EM) framework [49], which is widely adopted by existing crowdsourcing works [48, 89, 92, 116, 222], in each iteration, we devise the following two steps:

Step 1: we assume that p_w, r_w for each worker w is known, and infer the (prob-

abilistic) truth $q_{i,j}$ for each $(o_i, \ell_{i,j})$ pair, which is called *Inferring the Truth*;
Step 2: based on the computed truth $q_{i,j}$ for each $(o_i, \ell_{i,j})$ pair, we estimate each worker w's model p_w, r_w, which is called *Estimating Workers' Models*.

6.4.2 Iterative Method

Inferring the Truth

Based on the known (p_w, r_w) for each worker w, we infer the truth $q_{i,j} = \Pr(t_{i,j} = \mathrm{Y} \mid V)$ for each $(o_i, \ell_{i,j})$ pair (note that as $t_{i,j} = \mathrm{Y/N}$, we know that $\Pr(t_{i,j} = \mathrm{N} \mid V) = 1 - q_{i,j}$). To clarify, we denote $V = \{V_{i,j} \mid 1 \le i \le n, 1 \le j \le |L_i|\}$ as the vote set, where each element $V_{i,j} = \{(w, v)\}$ is a set of workers' votes for the pair $(o_i, \ell_{i,j})$, i.e., worker w votes v for $(o_i, \ell_{i,j})$, where v is Y or N. For example, from Table 6.3 we know that $V_{1,2} = \{(w_1, \mathrm{Y}), (w_2, \mathrm{Y}), (w_3, \mathrm{N}), (w_4, \mathrm{Y})\}$.

We assume that the votes are given independently (a typical assumption adopted in existing works [92, 127, 149, 211, 219, 222]), then $t_{i,j}$ is only related to $V_{i,j}$. Based on the Bayes' Theorem [23], we have

$$q_{i,j} = \Pr(t_{i,j} = \mathrm{Y} \mid V_{i,j}) \propto \Pr(V_{i,j} \mid t_{i,j} = \mathrm{Y}) \cdot \alpha, \tag{6.1}$$

where α is called *prior*, which represent general knowledge that the probability of a label is correct. Later we discuss how the value α can be set. Next we use the known workers' models to deduce $\Pr(V_{i,j} \mid t_{i,j} = \mathrm{Y})$ and $\Pr(V_{i,j} \mid t_{i,j} = \mathrm{N})$. For worker w, its model p_w and r_w can be represented as:

$$\begin{cases} p_w = \Pr(t_{i,j} = \mathrm{Y} \mid v^w_{i,j} = \mathrm{Y}) \ , \\ r_w = \Pr(v^w_{i,j} = \mathrm{Y} \mid t_{i,j} = \mathrm{Y}) \ . \end{cases} \tag{6.2}$$

(1) $\Pr(V_{i,j} \mid t_{i,j} = \mathrm{Y}) = \prod_{(w,v) \in V_{i,j}} \Pr(v^w_{i,j} = v \mid t_{i,j} = \mathrm{Y})$, that is, $\Pr(V_{i,j} \mid t_{i,j} = \mathrm{Y})$ depends on workers' votes. If worker w votes $v = \mathrm{Y}$, then $\Pr(v^w_{i,j} = \mathrm{Y} \mid t_{i,j} =$

$\mathrm{Y}) = r_w$ (Equation 6.2); otherwise, if $v = \mathrm{N}$, then $\Pr(v^w_{i,j} = \mathrm{N} \mid t_{i,j} = \mathrm{Y}) = 1 - \Pr(v^w_{i,j} = \mathrm{Y} \mid t_{i,j} = \mathrm{Y}) = 1 - r_w$. For ease of presentation, we use the indicator function $\mathbb{1}_{\{\cdot\}}$ which returns 1 if the argument is true; 0, otherwise. E.g., $\mathbb{1}_{\{2=5\}} = 0$ and $\mathbb{1}_{\{5=5\}} = 1$. Then

$$\Pr(V_{i,j} \mid t_{i,j} = \mathrm{Y}) = \prod_{(w,v)\in V_{i,j}} (r_w)^{\mathbb{1}_{\{v=\mathrm{Y}\}}} \cdot (1 - r_w)^{\mathbb{1}_{\{v=\mathrm{N}\}}}. \tag{6.3}$$

(2) We have $\Pr(V_{i,j} \mid t_{i,j} = \mathrm{N}) = \prod_{(w,v)\in V_{i,j}} \Pr(v^w_{i,j} = v \mid t_{i,j} = \mathrm{N})$.

We can prove Theorem 6.1.

Theorem 6.1. *$\Pr(v^w_{i,j} = Y \mid t_{i,j} = N) = \frac{\alpha\cdot(1-p_w)\cdot r_w}{(1-\alpha)\cdot p_w}$.*

Proof. To prove it, as discussed above, we have

$$\Pr(t_{i,j} = \mathrm{Y} \mid v^w_{i,j} = \mathrm{Y}) = \frac{\Pr(v^w_{i,j} = \mathrm{Y} \mid t_{i,j} = \mathrm{Y}) \cdot \alpha}{\Pr(v^w_{i,j} = \mathrm{Y} \mid t_{i,j} = \mathrm{Y}) \cdot \alpha + \Pr(v^w_{i,j} = \mathrm{Y} \mid t_{i,j} = \mathrm{N}) \cdot (1-\alpha)}.$$

Based on Equation 6.2, the above formula is indeed

$$p_w = \frac{r_w \cdot \alpha}{r_w \cdot \alpha + \Pr(v^w_{i,j} = \mathrm{Y} \mid t_{i,j} = \mathrm{N}) \cdot (1-\alpha)},$$

thus $\Pr(v^w_{i,j} = \mathrm{Y} \mid t_{i,j} = \mathrm{N}) = \frac{\alpha\cdot(1-p_w)\cdot r_w}{(1-\alpha)\cdot p_w}$.

□

Then we can derive $\Pr(V_{i,j} \mid t_{i,j} = \mathrm{N})$ as

$$\prod_{(w,v)\in V_{i,j}} \left(\frac{\alpha \cdot (1-p_w) \cdot r_w}{(1-\alpha) \cdot p_w}\right)^{\mathbb{1}_{\{v=\mathrm{Y}\}}} \cdot \left(1 - \frac{\alpha \cdot (1-p_w) \cdot r_w}{(1-\alpha) \cdot p_w}\right)^{\mathbb{1}_{\{v=\mathrm{N}\}}}. \tag{6.4}$$

Thus with known workers' models, we can compute each $q_{i,j}$ (Equation 6.1) based on Equations 6.3 and 6.4.

Example 16. *Suppose we have V based on Table 6.3 and workers' qualities in Table 6.7. We take $(o_1, \ell_{1,1})$: label* `tree` *($\ell_{1,1}$) in o_1 as an example and compute $q_{1,1} = \Pr(t_{1,1} = Y \mid V)$. As w_2, w_3, w_4 vote Y and w_1 votes N, based on Equation 6.3, $\Pr(V_{1,1} \mid t_{1,1} = Y) = (1 - r_{w_1}) r_{w_2} r_{w_3} r_{w_4} = 0.57 * 0.71 * 0.43 * 1 = 0.174$. Similarly, based on Equation 6.4, $\Pr(V_{1,1} \mid t_{1,1} = N) = 0.028$. Thus from Equation 6.1 we get $q_{1,1} = 86\%$, i.e.,* `tree` *is likely to be a correct label for o_1.*

Estimating Workers' Models

Note that if the truth is clearly known, we can derive all workers' contingency tables (e.g., Table 6.6) by simply counting TP, FP, FN and TN for each worker. However, in last step we get the probabilistic truth $q_{i,j}$ for each pair $(o_i, \ell_{i,j})$. Although we may easily decide $t_{i,j} = \text{Y(N)}$ by considering if $q_{i,j} \geq 0.5$ (or not), it cannot keep track of the probabilistic information which reflects the degree to be Y/N. Instead, we compute TP, FP, FN and TN for each worker by using the exact value of $q_{i,j}$. Formally, we denote D_w ($w \in \mathcal{W}$) as a set of votes given by worker w, and it contains a set of tuples $((o_i, \ell_{i,j}), v)$ representing that worker w votes v for the pair $(o_i, \ell_{i,j})$, where v is Y or N. For example, from Table 6.3 we know $D_{w_1} = \{((o_1, \ell_{1,1}), \text{N}), \ldots, ((o_2, \ell_{2,10}), \text{Y})\}$. Then for a worker w, TP is the number of $(o_i, \ell_{i,j})$ pairs that worker w votes Y and the truth is Y, i.e., $\text{TP} = \sum_{((o_i, \ell_{i,j}), v) \in D_w} \mathbb{1}_{\{v=\text{Y}\}} \cdot \mathbb{1}_{\{t_{i,j}=\text{Y}\}}$.

Thus for worker w, we can compute $\mathbb{E}[\text{ TP }]$ and similarly $\mathbb{E}[\text{ FP }]$, $\mathbb{E}[\text{ FN }]$, and $\mathbb{E}[\text{ TN }]$ as follows:

$$\begin{cases} \mathbb{E}[\,\mathrm{TP}\,] = \sum_{((o_i,\ell_{i,j}),v)\in D_w} \mathbb{1}_{\{v=\mathrm{Y}\}} \cdot q_{i,j}, \\ \mathbb{E}[\,\mathrm{FP}\,] = \sum_{((o_i,\ell_{i,j}),v)\in D_w} \mathbb{1}_{\{v=\mathrm{Y}\}} \cdot (\,1 - q_{i,j}\,), \\ \mathbb{E}[\,\mathrm{FN}\,] = \sum_{((o_i,\ell_{i,j}),v)\in D_w} \mathbb{1}_{\{v=\mathrm{N}\}} \cdot q_{i,j}. \end{cases} \tag{6.5}$$

For each $w \in \mathcal{W}$, the p_w and r_w are computed as

$$p_w = \frac{\mathbb{E}[\,\mathrm{TP}\,]}{\mathbb{E}[\,\mathrm{TP}\,] + \mathbb{E}[\,\mathrm{FP}\,]}\,,\ r_w = \frac{\mathbb{E}[\,\mathrm{TP}\,]}{\mathbb{E}[\,\mathrm{TP}\,] + \mathbb{E}[\,\mathrm{FN}\,]}. \tag{6.6}$$

Example 17. *In Table 6.3 we know that worker w_1 votes Y for $(o_1, \ell_{1,2})$, $(o_2, \ell_{2,2})$ and $(o_2, \ell_{2,10})$. Suppose the computed $q_{i,j}$ for those pairs are* $0.9, 0.8, 0.7$ *respectively, then based on Equation 6.5, we can estimate TP for worker w_1 as $\mathbb{E}[\ TP\ ] = 0.9 + 0.8 + 0.7 = 2.4$. Similarly we can estimate FP, FN, TN for worker w_1 and compute p_{w_1}, r_{w_1} following Equation 6.6.*

Iterative Computation Algorithm

We design an iterative method in Algorithm 12. It takes all workers' votes as input, and outputs all probabilistic truth and workers' models. With the initialized workers' models, it adopts an iterative approach, and the two steps are run in each iteration. Then it terminates if convergence is reached. Next we address the *prior*, *initialization*, *convergence* and *time complexity*, respectively.

Prior (α). To set the prior α (i.e., the probability that a label is Y), we can compute it in an unsupervised approach, i.e., after the two steps in each iteration, we can update α as $(\sum_{i=1}^{n} \sum_{j=1}^{|L_i|} q_{i,j}) / (\sum_{i=1}^{n} |L_i|)$.

Initialization. To initialize all workers' models, a direct way is to assign p_w and r_w with a fixed value $d \in [0, 1]$ for each worker w. In our experiments, we observe that it is robust when each worker is initialized as a decent worker, i.e., $p_w = r_w = d \geq 0.6$ for $w \in \mathcal{W}$.

Algorithm 12 Iterative Computation (Chapter 6).

```
Input: Workers' votes ($V,\ D_w$ for $w \in \mathcal{W}$), label correlations $\mathcal{M}(\cdot)$
Output: $q_{i,j}$ for $1 \le i \le n, 1 \le j \le |L_i|$, $(p_w, r_w)$ for $w \in \mathcal{W}$
1: Initialize $(p_w, r_w)$ for $w \in \mathcal{W}$;
2: while true do
3:    // Inferring the Truth
4:    for $1 \le i \le n, 1 \le j \le |L_i|$ do
5:       Compute $q_{i,j}$ using Equations 6.1, 6.3 and 6.4;
6:    end for
7:    // Considering Label Correlations
8:    for $1 \le i \le n, 1 \le j \le |L_i|$ do
9:       $S_{i,j} = \ln[\, q_{i,j}/(1 - q_{i,j})\,]$ ;
10:      Update $S_{i,j}$ to $S'_{i,j}$ based on Equation 6.7;
11:      $q_{i,j} = sig(\, S'_{i,j}\,) = 1/(1 + e^{-S'_{i,j}})$;
12:   end for
13:   // Estimating Workers' Models
14:   for $w \in \mathcal{W}$ do
15:      Compute $p_w$ and $r_w$ using Equations 6.5 and 6.6;
16:   end for
17:   // Check for Convergence
18:   if Converged then
19:      break;
20:   end if
21: end while
22: return $q_{i,j}$ for $1 \le i \le n, 1 \le j \le |L_i|$, $(p_w, r_w)$ for $w \in \mathcal{W}$;
```

Convergence. To check the convergence, a typical way is to see whether or not the change of parameters (i.e., all truth and worker models) in subsequent iteration is below some predefined threshold ε (e.g., 10^{-3}). In our experiments, we observe that our proposed approach is quick to converge (≤ 20 iterations).

Time Complexity. In each iteration, there are three main parts: (1) step 1: $\mathcal{O}(m \cdot |\mathcal{W}|)$; (2) step 2: $\mathcal{O}(m \cdot |\mathcal{W}|)$; (3) check for *Convergence*: $\mathcal{O}(m + |\mathcal{W}|)$. Suppose it takes c iterations to converge, the total time complexity is $\mathcal{O}(c \cdot m \cdot |\mathcal{W}|)$. In our experiments, we observe that it converges very quickly ($c \le 20$) for various datasets.

6.4.3 Speed-Up Computation

We now discuss how to incrementally update the parameters (i.e., the truth and workers' models). Assume we are faced with a situation that the workers' answers come in a high velocity, and we need an incremental method that instantly updates previously stored parameters when a new answer arrives. The basic idea is that upon receiving a worker's new answer, we only choose a small subset of *related* parameters to update, e.g., the truth of voted pairs, and all workers' models who have ever answered the voted pairs. The challenge is that (1) when a worker comes, which subset of parameters are to update; (2) what parameters should be stored for each pair and each worker, such that the update can be facilitated. In practice, we can integrate the incremental approach into the iterative approach in a *delayed* manner, i.e., we run iterative approach when receiving every 100 answers; and among the 100 answers, we only update parameters using the incremental approach.

The detailed algorithm is shown in Algorithm 13. The basic idea is that upon receiving a worker's answer (say worker w votes v for the pair $(o_i, \ell_{i,j})$), we only update the most important parameters, i.e., the truth of the pair and the models of workers who have answered the pair before. To be precise, we store the following parameters in order to facilitate incremental updates:
(1) for a worker $w \in \mathcal{W}$, we store three parameters: ETP_w, EFP_w, EFN_w, representing the expectations of TP, FP, FN, respectively;
(2) for a pair $(o_i, \ell_{i,j})$ $(1 \le i \le n, 1 \le j \le |L_i|)$, we store $m_{i,j}^{\mathrm{Y}}$ and $m_{i,j}^{\mathrm{N}}$, representing $\Pr(V_{i,j} \mid t_{i,j} = \mathrm{Y})$ and $\Pr(V_{i,j} \mid t_{i,j} = \mathrm{N})$ respectively.

Then if we want to compute any truth and worker's model, they can be directly returned based on the above stored parameters. For example, for the pair $(o_i, \ell_{i,j})$, we can get $q_{i,j} = \frac{m_{i,j}^{\mathrm{Y}}}{m_{i,j}^{\mathrm{Y}} + m_{i,j}^{\mathrm{N}}}$; for worker $w \in \mathcal{W}$, we can derive $p_w = \frac{\mathrm{ETP}_w}{\mathrm{ETP}_w + \mathrm{EFP}_w}$ and $r_w = \frac{\mathrm{ETP}_w}{\mathrm{ETP}_w + \mathrm{EFN}_w}$.

Algorithm 13 updates the stored parameters when it gets a new vote, i.e.,

Algorithm 13 Incremental Computation (Chapter 6).

```
Input: $(w, v, (o_i, \ell_{i,j}))$ (worker $w$ votes $v$ for the pair $(o_i, \ell_{i,j})$),
    $V$, $(m^{\mathrm{Y}}_{i,j}, m^{\mathrm{N}}_{i,j})$ for $1 \le i \le n$ and $1 \le j \le |L_i|$,
    ($\mathrm{ETP}_w$, $\mathrm{EFP}_w$, $\mathrm{EFN}_w$, $\mathrm{ETN}_w$) for $w \in \mathcal{W}$
Output: $V$, $(m^{\mathrm{Y}}_{i,j}, m^{\mathrm{N}}_{i,j})$ for $1 \le i \le n$ and $1 \le j \le |L_i|$,
    ($\mathrm{ETP}_w$, $\mathrm{EFP}_w$, $\mathrm{EFN}_w$, $\mathrm{ETN}_w$) for $w \in \mathcal{W}$
 1: $\mathrm{tempT} = m^{\mathrm{Y}}_{i,j}$, $\mathrm{tempN} = m^{\mathrm{N}}_{i,j}$;
 2: // (1) Inferring the Truth
 3: if $v = \mathrm{Y}$ then
 4:   $m^{\mathrm{Y}}_{i,j} = m^{\mathrm{Y}}_{i,j} \cdot \frac{\mathrm{ETP}_w}{\mathrm{ETP}_w + \mathrm{EFP}_w}$;
 5:   $m^{\mathrm{N}}_{i,j} = m^{\mathrm{N}}_{i,j} \cdot \frac{\mathrm{EFP}_w}{\mathrm{ETP}_w + \mathrm{EFN}_w}$;
 6: else
 7:   $m^{\mathrm{Y}}_{i,j} = m^{\mathrm{Y}}_{i,j} \cdot (1 - \frac{\mathrm{ETP}_w}{\mathrm{ETP}_w + \mathrm{EFP}_w})$;
 8:   $m^{\mathrm{N}}_{i,j} = m^{\mathrm{N}}_{i,j} \cdot (1 - \frac{\mathrm{EFP}_w}{\mathrm{ETP}_w + \mathrm{EFN}_w})$;
 9: end if
10: // (2) Estimating Workers' Models
11: // (2.1) Update the parameters for worker $w$
12: if $v = \mathrm{Y}$ then
13:   $\mathrm{ETP}_w = \mathrm{ETP}_w + \frac{m^{\mathrm{Y}}_{i,j}}{m^{\mathrm{Y}}_{i,j} + m^{\mathrm{N}}_{i,j}}$;
14:   $\mathrm{EFP}_w = \mathrm{EFP}_w + \frac{m^{\mathrm{N}}_{i,j}}{m^{\mathrm{Y}}_{i,j} + m^{\mathrm{N}}_{i,j}}$;
15: else
16:   $\mathrm{EFN}_w = \mathrm{EFN}_w + \frac{m^{\mathrm{Y}}_{i,j}}{m^{\mathrm{Y}}_{i,j} + m^{\mathrm{N}}_{i,j}}$;
17:   $\mathrm{ETN}_w = \mathrm{ETN}_w + \frac{m^{\mathrm{N}}_{i,j}}{m^{\mathrm{Y}}_{i,j} + m^{\mathrm{N}}_{i,j}}$;
18: end if
19: // (2.2) Update the parameters for workers who have voted the pair $(o_i, \ell_{i,j})$
    before
20: for $(w', v') \in V_{i,j}$ do
21:   if $v' = \mathrm{Y}$ then
22:     $\mathrm{ETP}_{w'} = \mathrm{ETP}_{w'} - \frac{\mathrm{tempT}}{\mathrm{tempT} + \mathrm{tempN}} + \frac{m^{\mathrm{Y}}_{i,j}}{m^{\mathrm{Y}}_{i,j} + m^{\mathrm{N}}_{i,j}}$;
23:     $\mathrm{EFP}_{w'} = \mathrm{EFP}_{w'} - \frac{\mathrm{tempN}}{\mathrm{tempT} + \mathrm{tempN}} + \frac{m^{\mathrm{N}}_{i,j}}{m^{\mathrm{Y}}_{i,j} + m^{\mathrm{N}}_{i,j}}$;
24:   else
25:     $\mathrm{EFN}_{w'} = \mathrm{EFN}_{w'} - \frac{\mathrm{tempT}}{\mathrm{tempT} + \mathrm{tempN}} + \frac{m^{\mathrm{Y}}_{i,j}}{m^{\mathrm{Y}}_{i,j} + m^{\mathrm{N}}_{i,j}}$;
26:     $\mathrm{ETN}_{w'} = \mathrm{ETN}_{w'} - \frac{\mathrm{tempN}}{\mathrm{tempT} + \mathrm{tempN}} + \frac{m^{\mathrm{N}}_{i,j}}{m^{\mathrm{Y}}_{i,j} + m^{\mathrm{N}}_{i,j}}$;
27:   end if
28: end for
29: $V_{i,j} = V_{i,j} \cup \{(w, v)\}$;
30: return $V$, $(m^{\mathrm{Y}}_{i,j}, m^{\mathrm{N}}_{i,j})$ for $1 \le i \le n$ and $1 \le j \le |L_i|$, ($\mathrm{ETP}_w$, $\mathrm{EFP}_w$, $\mathrm{EFN}_w$,
    $\mathrm{ETN}_w$) for $w \in \mathcal{W}$;
```

a worker w votes v for a pair $(o_i, \ell_{i,j})$. It shows how the the related parameters will be updated:

Inferring the Truth (lines 2-8). In this step, as the pair $(o_i, \ell_{i,j})$ gets a new vote, we update the parameters related to $(o_i, \ell_{i,j})$, i.e., $m^{\text{Y}}_{i,j}$ and $m^{\text{N}}_{i,j}$. From Equation 6.3 we know how to update $m^{\text{Y}}_{i,j}$ upon receiving a new vote v, that is, if $v = \text{Y}$, then $r_w = \frac{\text{ETP}_w}{\text{ETP}_w + \text{EFN}_w}$ will be multiplied to $m^{\text{Y}}_{i,j}$; otherwise, $1 - r_w$ will be multiplied to $m^{\text{Y}}_{i,j}$. We can derive similar update formula for $m^{\text{N}}_{i,j}$ from Equation 6.4. These updates can be reflected in lines 3-8.

Estimating Workers' Models (lines 9-24). In this step, we update the related workers' parameters. When worker w gives a vote, the worker's parameters (i.e., ETP_w, EFN_w, EFP_w, and ETN_w) will be updated. For example, based on Equation 6.5, we know that once a new vote $v = \text{Y}$ is received, then ETP_w will be added with the value $q_{i,j} = \frac{m^{\text{Y}}_{i,j}}{m^{\text{Y}}_{i,j} + m^{\text{N}}_{i,j}}$. Similarly, once a new vote $v = \text{N}$ is received, then ETP_w will be similarly updated (lines 11-16).

Besides worker w, we also know that $q_{i,j}$ has been updated, which will affect the parameters for the workers who have voted for $(o_i, \ell_{i,j})$ before. In order to update those affected parameters, we have to replace the old probability with the new one. So we first use tempT and tempN to temporally store $m^{\text{Y}}_{i,j}$ and $m^{\text{N}}_{i,j}$ (line 1). Then we update each worker who has voted for $(o_i, \ell_{i,j})$ before (lines 18-24). For example, for such a worker w', if the worker previously votes Y, then the worker's model $\text{ETP}_{w'}$ will be updated by first decreasing the old probability $(\frac{\text{tempT}}{\text{tempT}+\text{tempN}})$, and then adding the new one $(\frac{m^{\text{Y}}_{i,j}}{m^{\text{Y}}_{i,j} + m^{\text{N}}_{i,j}})$.

Finally we update $V_{i,j}$ by adding the new vote (line 25). Next we analyze the time complexity of Algorithm 13.

Time Complexity. For the time complexity of Algorithm 13, we only have to deal with lines 18-24 (an iteration), as other steps can be finished in constant time. For the iteration (i.e., lines 18-24), it enumerates all elements in $V_{i,j}$ and each iteration takes constant time. So the time complexity is $\mathcal{O}(|V_{i,j}|)$ (if receiving a new vote for $(o_i, \ell_{i,j})$). As $V_{i,j}$ stores all previous votes for $(o_i, \ell_{i,j})$, so the complexity is constrained by the maximum number of times a pair has been answered, which is $\mathcal{O}(|\mathcal{W}|)$. It is much more efficient compared with Algorithm 12, which costs $\mathcal{O}(c \cdot |\mathcal{W}| \cdot \sum_{i=1}^{n} |L_i|)$, especially when the number of all pairs (i.e., $\sum_{i=1}^{n} |L_i|$) is big.

6.5 Label Correlations

Since the labels of an object are not independent, we study how label correlations can facilitate inferring the truth. There are two sub-problems. The first is how to obtain the label correlations. The second is how to integrate the label correlations into our proposed method. Next we address them, respectively.

We can utilize existing label-correlation techniques [24, 210, 213] to generate the label correlations and regard them as *prior* input to our problem. Generally they can be classified into two categories: pairwise label correlations and higher order label correlations. Pairwise label correlations capture the relations between pairwise labels, which are mostly used because of its simplicity. For object o_i, the total number of pairwise label combinations is at most $|L_i|^2$. For example, the conditional dependency of two labels defines the probability that one label is correct for an object under the condition that the other label is correct. Higher oder label correlations capture the relations among subsets of labels, e.g., the co-existence probability among multiple labels, which are not frequently used mainly because of the introduced high complexity in parameters. Thus we focus on pairwise label correlations in this chapter, and leave higher order correlations for future work.

Label Correlation Function. It can be generalized that existing works [24, 210, 213] use a small subset of training data and derive a function $\mathcal{M}(\cdot)$, which takes two labels a, b as input, and outputs a score in $[-1, 1]$ that encodes the implication of label a's correctness to label b's correctness on an object. For example, $\mathcal{M}(\texttt{sun}, \texttt{sky})$=0.9 means that label sun's correctness strongly implies label sky's correctness, i.e., if sun is correct on an object, it is highly likely that sky is also correct; while $\mathcal{M}(\texttt{happy}, \texttt{sad})$=$-0.9$ means that label happy's correctness strongly implies label sad's incorrectness, i.e., if happy is correct on an object, it is highly likely that sad is *not* correct. Recently, some open-source tools, e.g., *word2vec* [202] takes a large text corpus as input and outputs a vector for each word, encoding the information of its adjacent words. Intuitively, the more frequent two words occur together in text corpus, the more similar their vectors are. We can also regard each label as a word and compute the cosine similarity of the two labels' vectors.

Using Label Correlations to Improve Inference. Our problem is "*Given the computed probabilistic truth $q_{i,j}$ for $1 \le i \le n$, $1 \le j \le |L_i|$ (i.e., the input), how to refine each $q_{i,j}$ by considering label correlations (i.e., $\mathcal{M}(\cdot)$)?*"

Since $\mathcal{M}(\cdot)$ may output negative values, directly acting it on $q_{i,j}$ may contradict to the probability constraint. Inspired by the *sigmoid* function, i.e., $sig(x) = \frac{1}{1+e^{-x}}$ ($x \in (-\infty, +\infty)$, $sig(\cdot) \in (0, 1)$), which is a monotonic increasing function, and it has been successfully used in various models as the mapping between a real value and a probability. So for each $q_{i,j}$, we consider label correlations and update as:
Step 1: we convert $q_{i,j}$ to a real value $S_{i,j}$;
Step 2: we act $\mathcal{M}(\cdot)$ on $S_{i,j}$ to derive a new $S'_{i,j}$;
Step 3: we revert $S'_{i,j}$ to a new probability $q_{i,j}$.

In the first step, based on the *sigmoid* function, we set $q_{i,j} = \frac{1}{1+e^{-S_{i,j}}}$ and get $S_{i,j} = \ln \frac{q_{i,j}}{1-q_{i,j}}$. The real value $S_{i,j} \in (-\infty, +\infty)$ represents the confidence that label $\ell_{i,j}$ is correct for o_i.

In the second step, our idea is to update $S_{i,j}$ based on the impact of related labels in o_i. Intuitively, suppose $\mathcal{M}(\ell_{i,k}, \ell_{i,j})$ is high, e.g., $\ell_{i,k}$=sun and $\ell_{i,j}$=sky, and $S_{i,k}$ is confident, i.e., if sun is likely to be a correct label for o_i, then sky is likely to be a correct label for o_i. So we update $S_{i,j}$ to $S'_{i,j}$, as follows:

$$S'_{i,j} = \alpha \cdot S_{i,j} + (1-\alpha) \cdot \sum\nolimits_{\ell_{i,k} \in RL} S_{i,k} \cdot \mathcal{M}(\ell_{i,k}, \ell_{i,j}) \; . \qquad (6.7)$$

We can see that $S'_{i,j}$ is the weighted sum of $S_{i,j}$, and an aggregated portion of related labels (RL). For each related label $\ell_{i,k} \in RL$, it multiples its confidence $S_{i,k}$ for o_i, and the implication of its correctness to $\ell_{i,j}$'s correctness, i.e., $\mathcal{M}(\ell_{i,k}, \ell_{i,j})$. There are two parameters: (1) the weight $\alpha \in [0,1]$ controls the impacts between itself ($S_{i,j}$) and related labels. From experiments (Section 6.7) we set $\alpha \in [0.6, 0.7]$, which gives itself more impact, but at the same time considers related labels. (2) RL is a set of related labels. We can roughly set it as all labels in o_i except itself ($\ell_{i,j}$), i.e., $RL = \{\ell_{i,k} \mid 1 \le k \le |L_i| \; \wedge \; k \ne j\}$. Although we can consider other choices, e.g., based on the definition of $\mathcal{M}(\cdot)$, we can focus on the labels $\ell_{i,k}$ whose confidence value is high (e.g., $q_{i,k} > 0.8$), however, we find that the simply way already performs very well.

In the third step, we revert the derived $S'_{i,j}$ to a new probability: $q_{i,j} = sig(S'_{i,j}) = 1/(1+e^{-S'_{i,j}})$.

As the three steps require $q_{i,j}$ ($1 \le i \le n, 1 \le j \le |L_i|$) as input, we can consider label correlations in our iterative method by running these three steps after *inferring the truth* in each iteration.

Time Complexity. In each iteration, we update all truth $q_{i,j}$ with $\mathcal{M}(\cdot)$. For each $q_{i,j}$, steps 1 and 3 take constant time; in step 2 (Equation 6.7), it considers at most $\mathcal{O}(|L_i|)$ related labels in o_i. Let $e = \max_{1 \le i \le n} |L_i|$, then it takes $\mathcal{O}(m \cdot e)$ for all pairs, which is dominated by other parts in last section (i.e., $\mathcal{O}(m \cdot |\mathcal{W}|)$) if there are enough workers.

Example 18. *Suppose workers' answers and qualities are known in Tables 6.2 and 6.7, and we take $\ell_{2,4}$ (`lake`) in o_2 as an example. In Section 6.4 we can get $q_{2,4}$=0.26. Then it is converted to $S_{2,4}$=-1.05. Suppose $\mathcal{M}$(`boat`,`lake`)=0.9, and for label `boat` ($\ell_{2,10}$), similarly we get $q_{2,10}$=0.99 and $S_{2,10}$=4.6.*[17] *To update $S_{2,4}$, for simplicity we only consider the most related label `boat`, and based on Equation 6.7 (α=0.7),*

[17]Since $p_{w_1} = 1$, then $q_{2,10} = 1$ and $S_{2,10} = +\infty$. For illustration we consider $q_{2,10} = 0.99$.

$S'_{2,4}$=0.7·$S_{2,4}$+0.3·$S_{2,10}$·0.9= 0.507. *Finally* $q_{2,4}$=$sig(S'_{2,4})$=0.62. *We can see that as* `boat` *is confident (99%) for* o_2*, and the implication of* `boat` *to* `lake` *is strong (0.9), this increases the probability (from 26% to 62%) that* `lake` *is correct for* o_2*.*

6.6 Online Task Assignment

In this section, we study the *Task Assignment Problem*: "Given the vote set V collected so far, when a worker w comes, which k objects (in all n ones) should be assigned to worker w?".

In selecting k objects, our basic idea is to estimate how much uncertainty can be reduced for each object, by considering *if* the task will be answered by w. Then we select the optimal k-object combination with the highest uncertainty reduction. However, to achieve the goal, there are two challenges here: (1) to select k out of n objects, we have to consider all $\binom{n}{k}$ combinations, which is exponential; (2) even for a single object o_i, to consider how worker w may answer it, there are still $2^{|L_i|}$ possible cases, which is exponential.

To address it, in the following, we first focus on selecting a single object, and then generalize to selecting multiple (e.g., k) objects.

Selecting a Single Object. When a worker w comes, to decide which object should be assigned to w, our idea is to select the object whose uncertainty can be reduced the most *if* the object is answered by worker w. To achieve the goal, for each object o_i ($1 \leq i \leq n$), we adopt the following three steps:
Step 1: we measure the object's uncertainty $U(o_i)$;
Step 2: we calculate the *expected* uncertainty if it is answered by w, denoted as $\mathbb{E}[\, U(o_i) \mid w \text{ answers } o_i \,]$;
Step 3: we compute its uncertainty reduction

$$\Delta U(o_i) = U(o_i) - \mathbb{E}[\, U(o_i) \mid w \text{ answers } o_i \,]. \tag{6.8}$$

We select the object whose uncertainty reduces most.

In the first step, we need to measure o_i's uncertainty based on workers' answers (i.e., V).

Definition 6.6 (Uncertainty). *We first define the uncertainty of a pair $(o_i, \ell_{i,j})$ based on Shannon Entropy [167], denoted as $U((o_i, \ell_{i,j})) = -[\ q_{i,j} \cdot \log q_{i,j} + (1 - q_{i,j}) \cdot \log(1 - q_{i,j})\ ]$. Then the uncertainty of an object o_i is defined as the sum of the uncertainty of all pairs in o_i, i.e., $U(o_i) = \sum_{j=1}^{|L_i|} U((o_i, \ell_{i,j}))$.*

In the second step, to calculate the expected uncertainty of object o_i by considering if it is answered by worker w, we first study how to compute the expected uncertainty of a pair $(o_i, \ell_{i,j})$ if worker w votes for that, denoted as $\mathbb{E}[\ U((o_i, \ell_{i,j})) \mid w \text{ votes for } (o_i, \ell_{i,j})\ ]$. The challenge is that we do not know worker w's vote (Y or N) for $(o_i, \ell_{i,j})$ before o_i is assigned. To address this issue, we estimate worker w's vote based on the truth probability $q_{i,j}$ and the worker's quality. If worker w has performed tasks before, then the worker's quality has been estimated and stored in the database already (Figure 6.2); otherwise, if w is a new worker, we can use the average quality (i.e., precision and recall) of workers that have performed on the tasks, which can be derived from the database.

Formally, we denote worker w's quality as p_w, r_w. To calculate the expected uncertainty of a pair $(o_i, \ell_{i,j})$, we (1) estimate the vote (Y/N) that worker w will give to the pair, and (2) estimate how the truth $q_{i,j}$ is updated if worker w votes Y/N for the pair. We address these two problems in Theorems 6.2 and 6.3.

Theorem 6.2. *The probability that a worker w will vote Y for $(o_i, \ell_{i,j})$ is $\Pr(v_{i,j}^w = Y \mid V) = r_w \cdot q_{i,j} + \frac{\alpha \cdot (1-p_w) \cdot r_w}{(1-\alpha) \cdot p_w} \cdot (1 - q_{i,j})$.*

Proof. Based on the Bayes' Theorem [23], considering $t_{i,j} \in \{\mathrm{Y}, \mathrm{N}\}$ we can derive that

$$\begin{aligned}\Pr(v_{i,j}^w = \mathrm{Y} \mid V) = &\Pr(v_{i,j}^w = \mathrm{Y} \mid t_{i,j} = \mathrm{Y}, V) \cdot \Pr(t_{i,j} = \mathrm{Y} \mid V) \\ &+ \Pr(v_{i,j}^w = \mathrm{Y} \mid t_{i,j} = \mathrm{N}, V) \cdot \Pr(t_{i,j} = \mathrm{N} \mid V).\end{aligned}$$

Given a known $t_{i,j}$, $v^w_{i,j}$ is independent of V, thus
$\Pr(v^w_{i,j} = \text{Y} \mid t_{i,j} = \text{Y}, V) = \Pr(v^w_{i,j} = \text{Y} \mid t_{i,j} = \text{Y}) = r_w$,
$\Pr(v^w_{i,j} = \text{Y} \mid t_{i,j} = \text{N}, V) = \Pr(v^w_{i,j} = \text{Y} \mid t_{i,j} = \text{N}) = \frac{\alpha\cdot(1-p_w)\cdot r_w}{(1-\alpha)\cdot p_w}$. Then we can derive

$$\Pr(v^w_{i,j} = \text{Y} \mid V) = r_w \cdot q_{i,j} + \frac{\alpha\cdot(1-p_w)\cdot r_w}{(1-\alpha)\cdot p_w} \cdot (1 - q_{i,j}),$$

which finalizes the proof. □

Theorem 6.3. *If worker w gives a new vote Y for $(o_i, \ell_{i,j})$, then $q_{i,j}$ is updated to $q^{\text{Y}}_{i,j} = q_{i,j} / [\, q_{i,j} + (1-q_{i,j}) \cdot \frac{\alpha\cdot(1-p_w)}{(1-\alpha)\cdot p_w}\,]$; otherwise, if the worker's new vote is N, then $q_{i,j}$ is updated to $q^{N}_{i,j} = q_{i,j} / [\, q_{i,j} + (1-q_{i,j}) \cdot \big(1 - \frac{\alpha\cdot(1-p_w)\cdot r_w}{(1-\alpha)\cdot p_w}\big) / (1 - r_w)\,]$.*

Proof. If worker w gives a new vote v for $(o_i, \ell_{i,j})$, from Equations 6.3 and 6.4, we know that $\Pr(V_{i,j} \mid t_{i,j} = \text{Y})$ is updated as $\Pr(V_{i,j} \mid t_{i,j} = \text{Y}) \cdot A$ and $\Pr(V_{i,j} \mid t_{i,j} = \text{N})$ is updated as $\Pr(V_{i,j} \mid t_{i,j} = \text{N}) \cdot B$, where A and B are denoted as follows:

$$\begin{cases} A = (r_w)^{\mathbb{1}_{\{v=\text{Y}\}}} \cdot (1-r_w)^{\mathbb{1}_{\{v=\text{N}\}}}, \\ B = \big(\frac{\alpha\cdot(1-p_w)\cdot r_w}{(1-\alpha)\cdot p_w}\big)^{\mathbb{1}_{\{v=\text{Y}\}}} \cdot \big(1 - \frac{\alpha\cdot(1-p_w)\cdot r_w}{(1-\alpha)\cdot p_w}\big)^{\mathbb{1}_{\{v=\text{N}\}}}. \end{cases}$$

Then based on Equation 6.1, $q_{i,j}$ should be updated as

$$\frac{\Pr(V_{i,j} \mid t_{i,j} = \text{Y}) \cdot A}{\Pr(V_{i,j} \mid t_{i,j} = \text{Y}) \cdot A + \Pr(V_{i,j} \mid t_{i,j} = \text{N}) \cdot B}.$$

By dividing $A \cdot [\ \Pr(V_{i,j} \mid t_{i,j} = \text{Y}) + \Pr(V_{i,j} \mid t_{i,j} = \text{N})\ ]$ both on its numerator and denominator, we can derive that $q_{i,j}$ is updated as $q_{i,j} / [\, q_{i,j} + (1-q_{i,j}) \cdot \frac{B}{A}\,]$.

Thus if worker w gives a new vote Y for $(o_i, \ell_{i,j})$, then $q_{i,j}$ is updated as $q^{\text{Y}}_{i,j} = q_{i,j} / [\, q_{i,j} + (1-q_{i,j}) \cdot \frac{\alpha\cdot(1-p_w)}{(1-\alpha)\cdot p_w}\,]$; otherwise, if the worker's new vote is N, then $q_{i,j}$ is updated as
$q^{\text{N}}_{i,j} = q_{i,j} / [\, q_{i,j} + (1-q_{i,j}) \cdot \big(1 - \frac{\alpha\cdot(1-p_w)\cdot r_w}{(1-\alpha)\cdot p_w}\big) / (1 - r_w)\,]$.

□

Based on the results in Theorems 6.2 and 6.3, we have

$$
\begin{aligned}
&\mathbb{E}[\ U((o_i, \ell_{i,j})) \mid w \text{ votes for } (o_i, \ell_{i,j})\ ] \\
&= U_{i,j}^{\mathrm{Y}} \cdot \Pr(v_{i,j}^{w} = \mathrm{Y} \mid V) + U_{i,j}^{\mathrm{N}} \cdot \Pr(v_{i,j}^{w} = \mathrm{N} \mid V), \text{ where} \\
&\begin{cases}
U_{i,j}^{\mathrm{Y}} = -[\ q_{i,j}^{\mathrm{Y}} \cdot \log q_{i,j}^{\mathrm{Y}} + (1 - q_{i,j}^{\mathrm{Y}}) \cdot \log(1 - q_{i,j}^{\mathrm{Y}})\ ], \\
U_{i,j}^{\mathrm{N}} = -[\ q_{i,j}^{\mathrm{N}} \cdot \log q_{i,j}^{\mathrm{N}} + (1 - q_{i,j}^{\mathrm{N}}) \cdot \log(1 - q_{i,j}^{\mathrm{N}})\ ].
\end{cases}
\end{aligned}
\tag{6.9}
$$

Note that $\Pr(v_{i,j}^{w} = \mathrm{Y} \mid V)$ and $\Pr(v_{i,j}^{w} = \mathrm{N} \mid V)$ can be derived from Theorem 6.2, and $q_{i,j}^{\mathrm{Y}}$, $q_{i,j}^{\mathrm{N}}$ can be derived from Theorem 6.3. Intuitively, Equation 6.9 considers the cases that worker w will vote Y (N) for $(o_i, \ell_{i,j})$, and the updated uncertainty $U_{i,j}^{\mathrm{Y}}$ ($U_{i,j}^{\mathrm{N}}$) if the vote is given.

Example 19. *Suppose workers' answers (V) and qualities are known in Tables 6.3 and 6.7. We can compute* $q_{1,1} = 0.86$. *Suppose a new worker* $w \notin \mathcal{W}$ *comes and we compute both* $U((o_1, \ell_{1,1}))$ *and* $\mathbb{E}[\ U((o_1, \ell_{1,1})) \mid w \textit{ votes for } (o_1, \ell_{1,1})\ ]$. *First* $U((o_1, \ell_{1,1})) = -(0.86 * \log 0.86 + 0.14 * \log 0.14) = 0.4$. *The new worker w's quality is estimated as the average quality:* $p_w = 0.77$, $r_w = 0.64$ *from Table 6.7. Based on Theorem 6.2, we get* $\Pr(v_{1,1}^{w} = Y \mid V) = 0.64 * 0.86 + (\frac{0.64}{0.77} - 0.64) * 0.14 = 0.58$ *and* $\Pr(v_{1,1}^{w} = N \mid V) = 1 - 0.58 = 0.42$. *Based on Theorem 6.3 we get* $q_{1,1}^{Y} = 0.86/(0.86 + 0.14 * \frac{0.23}{0.77}) = 0.95$, *and similarly* $q_{1,1}^{N} = 0.73$. *Then we can derive* $U_{1,1}^{Y} = -(0.95 * \log 0.95 + 0.05 * \log 0.05) = 0.2$ *and* $U_{1,1}^{N} = 0.58$. *Finally* $\mathbb{E}[\ U((o_1, \ell_{1,1})) \mid w \textit{ votes for } (o_1, \ell_{1,1})\ ] = 0.2 * 0.58 + 0.58 * 0.42 = 0.36$. *The uncertainty of* $(o_1, \ell_{1,1})$ *changes from 0.4 to 0.36 after being voted by w.*

Having known how to compute the expected uncertainty of a pair, we now compute the expected uncertainty of object o_i if it is answered by worker w, i.e., $\mathbb{E}[\ U(o_i) \mid w \text{ answers } o_i\ ]$. However, it is challenging to directly compute it, as it requires to enumerate all possible answers of worker w to o_i: $\{\mathrm{Y}, \mathrm{N}\}^{|L_i|}$, containing $2^{|L_i|}$ items.

We next prove the following theorem, which efficiently computes the expected uncertainty of an object o_i, reducing the complexity from exponential ($2^{|L_i|}$) to linear ($|L_i|$). The basic idea is that if we decompose the above formula as two parts, where each part considers that w votes Y (N) to a pair (e.g., $(o_i, \ell_{i,1})$), then we can verify that the expected uncertainty of the pair can be extracted from the above formula. Similarly the expected uncertainty of all pairs in o_i can be extracted and added independently.

Theorem 6.4. *The expected uncertainty of o_i ($1 \leq i \leq n$) is the sum of the expected uncertainties of all pairs in o_i, i.e., $\mathbb{E}[\ U(o_i) \mid w \text{ answers } o_i\ ] = \sum_{j=1}^{|L_i|} \mathbb{E}[\ U((o_i, \ell_{i,j})) \mid w \text{ votes for } (o_i, \ell_{i,j})\ ]$.*

Proof. Let $\sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_{|L_i|}\}$ denote worker w's one possible answer for o_i, where each $\sigma_j = \text{Y(N)}$ ($1 \leq j \leq |L_i|$) represents that worker w votes Y(N) for $(o_i, \ell_{i,j})$. Then for each $\sigma \in \{\text{Y}, \text{N}\}^{|L_i|}$, we have to compute the uncertainty of o_i if worker w gives answer σ, denoted as U_i^σ. Based on Definition 6.6, it aggregates the uncertainty of all pairs in o_i considering the answer σ, i.e.,

$U_i^\sigma = \sum_{j=1}^{|\sigma|} -[\ q_{i,j}^{\sigma_j} \cdot \log q_{i,j}^{\sigma_j} + (1 - q_{i,j}^{\sigma_j}) \cdot \log(1 - q_{i,j}^{\sigma_j})\ ]$,

where $q_{i,j}^{\sigma_j}$ (i.e., $q_{i,j}^{\text{Y}}$ or $q_{i,j}^{\text{N}}$) is defined in Theorem 6.3. If we assume that worker w will give independent votes to $(o_i, \ell_{i,j})$, then we can derive

$$\mathbb{E}[\ U(o_i) \mid w \text{ answers } o_i\ ] = \sum_{\sigma \in \{\text{Y,N}\}^{|L_i|}} U_i^\sigma \cdot \prod_{j=1}^{|L_i|} \Pr(v_{i,j}^w = \sigma_j \mid V).$$

Given the above formula, we can decompose its right hand side into the sum of two parts: the first part is the sum of $2^{|L_i|-1}$ elements, considering that $\sigma_{|L_i|} = \text{Y}$ (worker w votes Y for $(o_i, \ell_{i,|L_i|})$), and the second part is the sum of the remaining $2^{|L_i|-1}$ elements, considering that $\sigma_{|L_i|} = \text{N}$ (worker w votes N for $(o_i, \ell_{i,|L_i|})$), i.e.,

$$\begin{aligned}
&\mathbb{E}[\,U(o_i) \mid w \text{ answers } o_i\,] = \\
&\sum_{\sigma' \in \{\mathrm{Y},\mathrm{N}\}^{|L_i|-1}} (U_i^{\sigma'} + U_{i,|L_i|}^{\mathrm{Y}}) \Pr(v_{i,|L_i|}^w = \mathrm{Y} \mid V) \prod_{j=1}^{|L_i|-1} \Pr(v_{i,j}^w = \sigma'_j \mid V) \\
+ &\sum_{\sigma' \in \{\mathrm{Y},\mathrm{N}\}^{|L_i|-1}} (U_i^{\sigma'} + U_{i,|L_i|}^{\mathrm{N}}) \Pr(v_{i,|L_i|}^w = \mathrm{N} \mid V) \prod_{j=1}^{|L_i|-1} \Pr(v_{i,j}^w = \sigma'_j \mid V).
\end{aligned}$$

With a careful deduction, we can derive that

$$\begin{aligned}
&\mathbb{E}[\,U(o_i) \mid w \text{ answers } o_i\,] \\
&= \sum_{\sigma' \in \{\mathrm{Y},\mathrm{N}\}^{|L_i|-1}} U_i^{\sigma'} \cdot \prod_{j=1}^{|L_i|-1} \Pr(v_{i,j}^w = \sigma'_i \mid V) \\
&\quad + \mathbb{E}[\,U((o_i, \ell_{i,|L_i|})) \mid w \text{ votes for } (o_i, \ell_{i,|L_i|})\,].
\end{aligned}$$

The above formula shows that the expected uncertainty of the pair $(o_i, \ell_{i,|L_i|})$ can be extracted. Following the same way, the expected uncertainty of all pairs in o_i can be extracted and added independently. Thus we have

$$\begin{aligned}
&\mathbb{E}[\,U(o_i) \mid w \text{ answers } o_i\,] \\
&= \textstyle\sum_{j=1}^{|L_i|} \mathbb{E}[\,U((o_i, \ell_{i,j})) \mid w \text{ votes for } (o_i, \ell_{i,j})\,].
\end{aligned}$$

□

In the third step, we can compute the uncertainty reduction of object o_i: $\Delta U(o_i) = U(o_i) - \mathbb{E}[\,U(o_i) \mid w \text{ answers } o_i\,]$ and select a single object with the highest reduction $\Delta U(o_i)$. Note that label correlations are implicitly considered in the assignment, as it takes the derived $q_{i,j}$ to further compute the uncertainty and estimate worker's votes.

Selecting Multiple Objects. When a worker comes, existing crowdsourcing platforms such as AMT [1] can support to batch multiple tasks in a HIT (Human Intelligence Task), and assign for the coming worker. Similar to Theorem 6.4, we can easily extend our method to select k objects in Theorem 6.5, by retrieving top-k objects with highest reduction in uncertainty.

Theorem 6.5. *The optimal k-object combination is to select k objects with the highest reduction in uncertainty, i.e.,* $\Delta U(o_i)$.

Proof. As a worker gives answers independently, suppose we select a fixed set of k objects (denoted as $\mathcal{T}'$) to assign for the coming worker w, i.e., $\mathcal{T}' = \{o_i\}$ and $|\mathcal{T}'| = k$, then following the proof for Theorem 6.4, we can similarly prove that the expected uncertainty for these k objects can be regarded as the sum of expected uncertainties of individual object, i.e.,

$$\begin{aligned}&\mathbb{E}[\ \textstyle\sum_{o_i \in \mathcal{T}'} U(o_i) \mid w \text{ answers objects in } \mathcal{T}'\ ]\\ &= \textstyle\sum_{o_i \in \mathcal{T}'} \mathbb{E}[\ U(o_i) \mid w \text{ answers } o_i\ ].\end{aligned}$$

As our target is to select the optimal k-object combination, such that the uncertainty can be reduced the most. Based on the above Equation, we know that for a fixed set of objects $\mathcal{T}'$, their uncertainty reduction can be expressed as

$$\textstyle\sum_{o_i \in \mathcal{T}'} \Delta U(o_i).$$

Then in order to select the optimal k-object combination, we can treat each object independently, i.e., we compute each object o_i's uncertainty reduction $\Delta U(o_i)$, and select the top-k objects with the highest uncertainty reduction.

$\square$

Time Complexity. To select top-k objects with highest uncertainty reduction, we have to compute $\Delta U(o_i)$ for each object o_i. For an object o_i ($1 \leq i \leq n$), step 1 takes $\mathcal{O}(|L_i|)$ time (Definition 6.6); step 2 also takes $\mathcal{O}(|L_i|)$ time (Theorem 6.4); step 3 takes constant time (Equation 6.8). Then computing $\Delta U(o_i)$ for all objects takes $\mathcal{O}(m)$ time, where $m = \sum_{i=1}^{n} |L_i|$. As selecting top-k objects requires $\mathcal{O}(n)$ time (the problem of finding top-k elements in an n-array can be solved linearly using the PICK algorithm [26]), the total time complexity of task assignment problem is $\mathcal{O}(m)$.

6.7 Experiments

In this section, we evaluate Comet on both real-world datasets and simulated data. We perform experiments on two crowdsourcing platforms and examine the effectiveness and efficiency of Comet. We also evaluate the scalability of Comet on simulated data. Comet is implemented in Python 2.7 and evaluated on a machine with 8GB memory with Ubuntu OS.

6.7.1 Settings

Two Real-World Datasets

Image Tagging. `Corel5k dataset` [58] contains 5000 images, tagged with 260 labels in all. We select 300 images with the most labels, and 20 labels that are most frequently tagged: {water, sky, tree, people, grass, building, mountain, snow, flower, cloud, rock, stone, street, plane, bear, field, sand, bird, beach, boat}. We generate 300 tasks, where each task contains an image and 20 labels.

Email Tagging. `Enron dataset` [106] contains 1700 emails, tagged with 53 labels in all. For better human readability, we filter the emails whose size is >1KB. In the remaining 837 ones, we select 300 emails with the most labels, and 20 labels that are most frequently tagged: {forwarded email, arrangement, company business&strategy, new text&forwarded material, attachment, personal context, document checking, employment arrangement, secret message, internal project, California energy crisis, regulation, internal company operation, political influence, purely personal, point to url, alliance, internal company policy, empty message, current company image}. We generate 300 tasks, where each task contains an email (with contents in it) and 20 labels.

Two Crowdsourcing Platforms

ChinaCrowd [3]. ChinaCrowd is an emerging Chinese crowdsourcing platform, whose workers are mostly Chinese people. We perform experiments on it without quality control (i.e., no QC).

Amazon Mechanical Turk (AMT) [1]. AMT is a famous crowdsourcing platform. When a worker finishes a task, the task requester can give feedback to the platform on whether or not the worker's answer is approved. AMT provides each worker's historical approval rate (for past tasks) to help task requester do Quality Control. Thus we perform experiments on AMT with quality control (i.e., with QC), by setting that a worker is qualified to answer our tasks only if the worker's historical approval rate is $\geq 70\%$.

Collecting Workers' Answers

We publish two datasets, i.e., image tagging and email tagging (the ground truth of the two datasets are known for evaluation purposes) on the above two platforms, so there are 4 experiments in total, where each one corresponds to a dataset on a platform. In each experiment, we assign each task to 5 workers, and pay \$0.01 for a worker upon answering a task. Thus for each experiment, we pay $300 \times 5 \times \$0.01 = \$15$ for workers. After all experiments are accomplished, we collect 4 datasets of workers' answers.

We summarize the dataset statistics in Table 6.8, where we record the dataset, the platform, whether or not there is quality control (QC), #tasks, #collected answers, the total cost and #workers participated for each experi-

Table 6.8: The Statistics of Datasets.

Dataset	*Image Tagging*		*Email Tagging*	
Platform	AMT [1]	ChinaCrowd [3]	AMT [1]	ChinaCrowd [3]
QC	with QC	no QC	with QC	no QC
#tasks	300	300	300	300
#answers	300×5	300×5	300×5	300×5
cost	\$15	\$15	\$15	\$15
#workers	62	10	60	9

ment. Note that AMT [1] has more workers participating in tasks than ChinaCrowd [3], and this is reasonable as ChinaCrowd [3] is a new platform, while AMT has a lot more active workers.

Comparisons

Existing works [33, 92, 127, 141, 149, 222] model a worker differently. For each worker model, we select one representative:

Majority Vote (MV) [33,141]. It does not model a worker and treat each worker equally. Given the collected 5 votes for a pair, it decides to return Y/N that attains more votes (or ≥ 3 votes).

CDAS [127]. It uses *Accuracy* to model a worker, which is also used in [48, 211]. CDAS [127] leverages workers' answers for tasks to infer each worker's *Accuracy*.

DS [47, 92]: It uses both TNR and *Recall* to model a worker, which is also used in [149, 222]. DS [92] takes workers' answers as input, and iteratively infer all workers' models.

Comet: Our method with no label correlations.

Comet+: Our method that considers label correlations. We use a small set of *training data*, i.e., the ground truth of 5% tasks in the original dataset to derive $\mathcal{M}(\cdot)$. For two labels a and b, the output $\mathcal{M}(a, b)$ is the conditional probability that label b is correct given that label a is correct for an object. To get

$\mathcal{M}(\texttt{sun}, \texttt{sky})$, for instance, we calculate the fraction of images having both labels `sun` and `sky`, over those that have `sun`.

Metrics

We use different metrics to evaluate the effectiveness and efficiency when comparing different methods.

Effectiveness. We use three metrics *Precision*, *Recall*, and *F-score* to measure the quality of a method. Note that many existing works [92, 127, 149] use *Accuracy* to evaluate the quality of a method. However, it does not fit to the multi-label setting, because the number of incorrect labels is much higher than the correct labels. In this case, the harmonic-mean of *Precision* and *Recall*, i.e., *F-score*, is used [192, 195, 199, 222] to measure a method's quality: $\textit{F-score} = \frac{2 \cdot \textit{Precision} \cdot \textit{Recall}}{\textit{Precision} + \textit{Recall}}$.

Efficiency. We adopt the execution time.

6.7.2 Observing Workers' Real Qualities

In order to observe workers' real qualities in practice, we leverage the ground truth and use workers' answers for each dataset to compute each worker's real quality under different models. In Figure 6.4, each graph corresponds to the workers' real qualities in one dataset. Generally speaking, there are three ways to model a worker: (a) *Accuracy* [48, 127, 211], (b) TNR+*Recall* [92,149,222], and (c) Precision+*Recall* (our model). So in each graph, we draw three points for a worker, whose coordinates (x, y) are respectively (*Precision*, *Recall*) in a '$\times$', (TNR, *Recall*) in a '$\boxdot$' and (*Accuracy*, *Accuracy*) in a '$\odot$'. For example, Figure 6.4(a) shows workers' real qualities in image tagging collected from ChinaCrowd [3] (no QC). There are 10 workers that participate in tasks, and each worker corresponds to 3 points in the graph, where each point corresponds to one of the worker's model. From Figure 6.4 we can observe that *Accuracy* and TNR are all very high for workers, even for workers

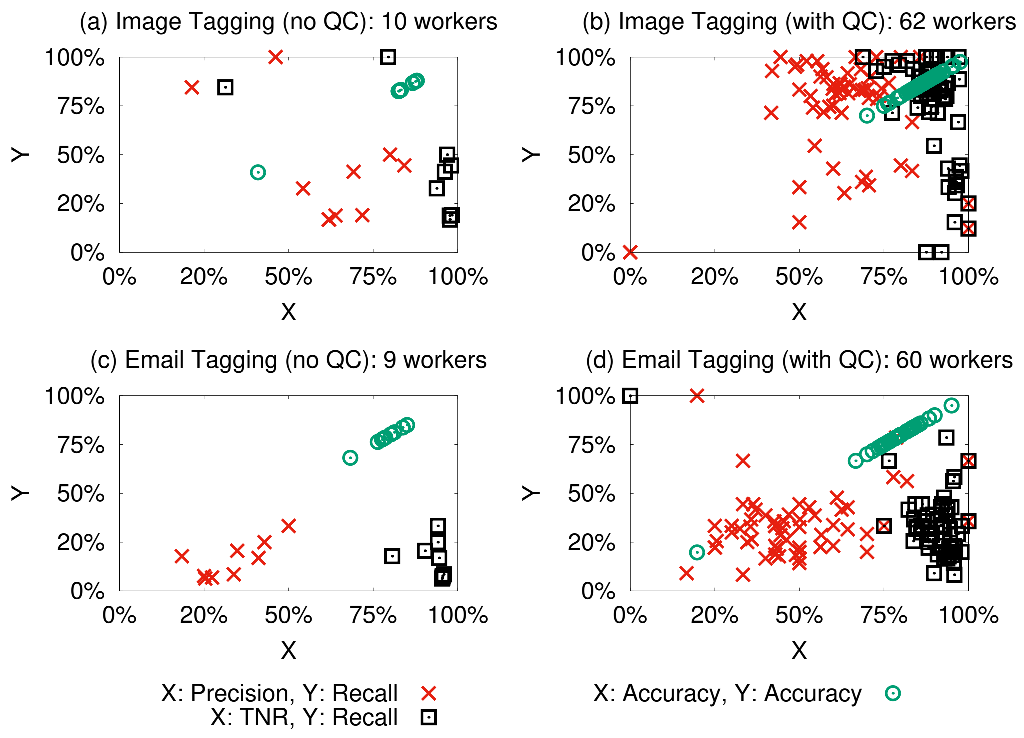


Figure 6.4: Observing Workers' Real Qualities on Collected Datasets.

with no QC: in the 4 graphs, '⊙'s are located in the top right diagonal, and '⊡' are located in the very right part of x-axis. Particularly, in Section 6.3, we have shown Figure 6.3, which draws the histograms of all 141 workers' qualities in the 4 datasets, and the same observations can be derived. Moreover, under our model (i.e., *Precision*, *Recall*), workers' models vary in different settings. As can be seen, generally workers in Figure 6.4(a) are of *high Precision*, *low Recall*; in Figure 6.4(b) are of *high Precision*, *high Recall*; in Figure 6.4(c) are of *low Precision*, *low Recall*; in Figure 6.4(d) are of *mediate Precision*, *low Recall*. On one hand, this verifies the need to model a worker using *Precision* and *Recall*, as it can capture different variants of workers; on the other hand, it also brings out the challenges to achieve better results under various workers, which will be shown next.

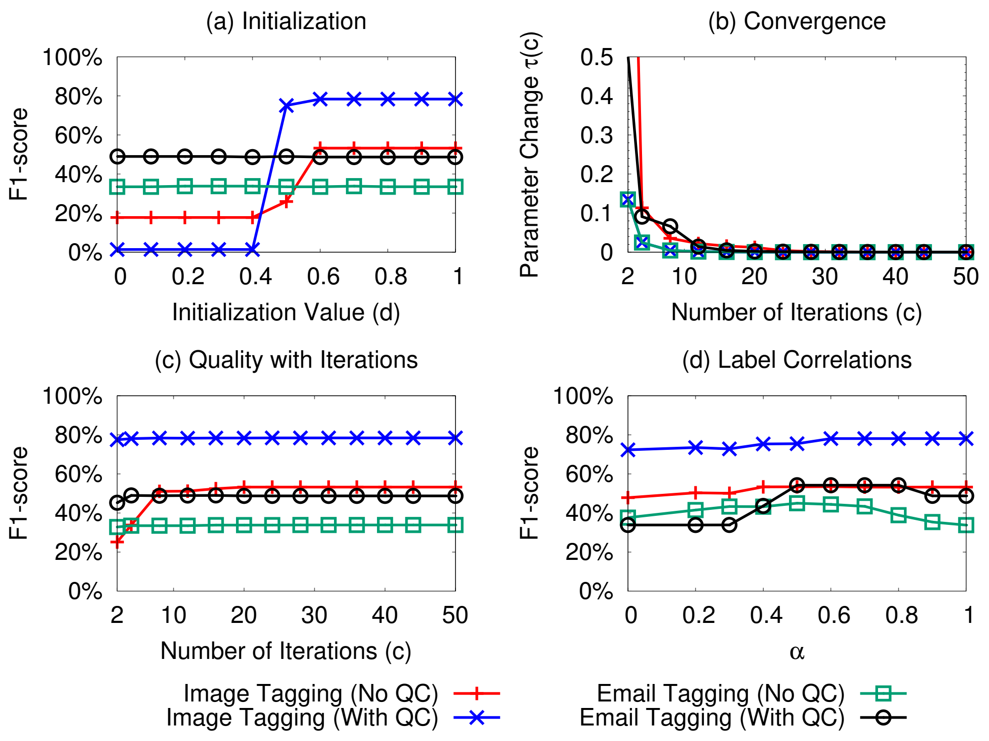


Figure 6.5: Truth Inference (Parameter Settings).

6.7.3 Truth Inference

For the *Truth Inference Problem*, we compare our proposed methods (Comet, Comet+) with three state-of-the-art methods (MV, CDAS, DS). We first evaluate how to set parameters, and then show the comparisons of different methods on all collected datasets.

Parameter Settings

We evaluate the parameters in our iterative method, i.e., initialization, convergence (Section 6.4) and label correlations (Section 6.5).

Initialization. Figure 6.5(a) shows the quality for different initializations on all datasets. To be specific, for each worker $w \in \mathcal{W}$, we initialize $p_w = r_w = d$

and vary $d \in [0, 1]$ in each dataset. We run enough (50) rounds to ensure it converges, and compute *F-score* for different d. It can be seen in Figure 6.5(a) that all datasets reach its highest quality if the worker is initialized as a decent worker, i.e., $d \geq 0.6$. So we initialize each worker $w \in \mathcal{W}$ as $p_w = r_w = 0.7$.

Convergence. In Figures 6.5(b)(c), we study the number of iterations to converge. We identify convergence if the change of parameters (all truth and worker models) between subsequent iteration is below some predefined threshold ε. For the c-th iteration, we denote the computed truth as $q_{i,j}^{(c)}$ and worker w's model as $p_w^{(c)}, r_w^{(c)}$. Then the parameter change for iteration c and $c-1$ is denoted as $\tau(c)$, i.e., the sum of average differences in truth and workers' models:

$$\tau(c) = \frac{\sum_{i,j} |q_{i,j}^{(c)} - q_{i,j}^{(c-1)}|}{\sum_{i=1}^{n} |L_i|} + \frac{\sum_w (|p_w^{(c)} - p_w^{(c-1)}| + |r_w^{(c)} - r_w^{(c-1)}|)}{|\mathcal{W}|}.$$

We vary $c \in [2, 50]$ and compute $\tau(c)$ for all datasets in Figure 6.5(b), which shows that $\tau(c)$ becomes much smaller as c increases. Especially, for all datasets, we have $\tau(c) \leq 10^{-3}$ as $c \geq 20$. Take a further step, in Figure 6.5(c) we show the quality (*F-score*) based on the computed truth in each iteration, which can be seen that the quality improves with the increasing c. In particular, the quality achieves highest and remains stable for different datasets as $c \geq 20$. Thus it is quick to converge ($\leq$ 20 iterations).

Label Correlations. In Figure 6.5(d), we observe the effect of varying label correlation parameter α in Equation 6.7. $\alpha \in [0, 1]$ measures the impact of $S_{i,j}$ (itself) and related labels. A larger α means more impact on $S_{i,j}$ and a smaller α means more impact on related labels. We vary $\alpha \in [0, 1]$ and observe the quality of all datasets in Figure 6.5(d). It can be seen that either α close to 1 or 0 results in a smaller quality, as it is not wise to totally agree with $S_{i,j}$ or related labels. We observe that the quality achieves highest as $\alpha \in [0.6, 0.7]$ for all datasets, which gives $S_{i,j}$ more impact, but at the same time considers related labels. Thus we

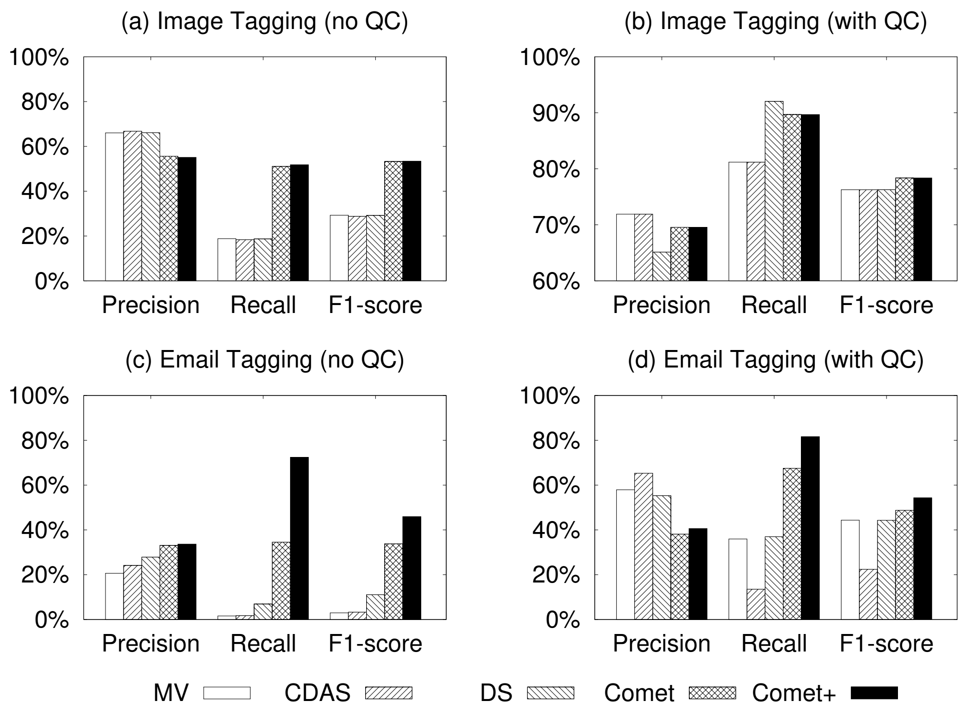


Figure 6.6: Truth Inference (Effectiveness Comparisons).

choose the parameter α as 0.7.

Comparisons

Effectiveness. Figure 6.6 shows both one-sided quality (*Precision* and *Recall*) and two-sided quality (*F-score*) for all 4 datasets. We can observe that (1) our proposed approaches perform much better on datasets with no QC, where Comet and Comet+ lead other competitors for more than 20%. As can be seen from Figures 6.6(a)(c), MV, CDAS and DS attain much higher *Precision* compared with *Recall*, resulting in low *F-score*. The reason is that low-quality workers still have high *Accuracy* and TNR, thus the methods will highly trust a poor worker's N vote for a pair, and the label will be selected by the methods only if multiple Y votes are given to the pair, resulting in the limited number of selected labels.

However, Comet can learn workers' diverse characteristics and make reasonable decisions, resulting in a high *F-score* (with balanced *Precision* and *Recall*). (2) For datasets with QC, Comet (Comet+) can also outperform state-of-the-arts on email tagging dataset. Even in the case that workers are of high qualities, especially for image tagging (can be seen in Figure 6.4(b)), we can still have marginal improvement. (3) By considering label correlations, Comet+ improves Comet a lot on email tagging. However, there is not much improvement on image tagging. The reason is that labeling an image is much easier. Based on the ground truth, we further analyze image tagging, finding that most of the strong label correlations are caught by the results of Comet (computed purely based on workers' answers), and there are only < 20 pairs that can be further benefited by considering $\mathcal{M}(\cdot)$, resulting in Comet+'s limited improvement. However, email tagging tasks are more difficult for workers and most of the label correlations are not caught by Comet (e.g., the high value of $\mathcal{M}(\cdot)$ on 'California energy crisis' and 'company business&strategy'). Similarly, we analyze the results of Comet on email tagging, finding that there are > 200 pairs that can be further benefited by considering $\mathcal{M}(\cdot)$, thus Comet+ can improve a lot in quality compared with Comet.

Efficiency. We compare the execution time in Figure 6.7. Given that truth inference can be addressed off-line, we can observe that all methods are efficient. To be specific, they all can be finished within 1.5s. MV and CDAS are more efficient as they do not need many computations; DS and Comet adopt iterative approaches which require more computations; Comet+ considers label correlations in the iterative approach, which is the least efficient. We will evaluate the scalability of Comet+ on simulated data.

6.7.4 Task Assignment

We evaluate the effectiveness and efficiency of Comet's task assignment (Section 6.6). As existing works [27,127,222] that study task assignment focuses

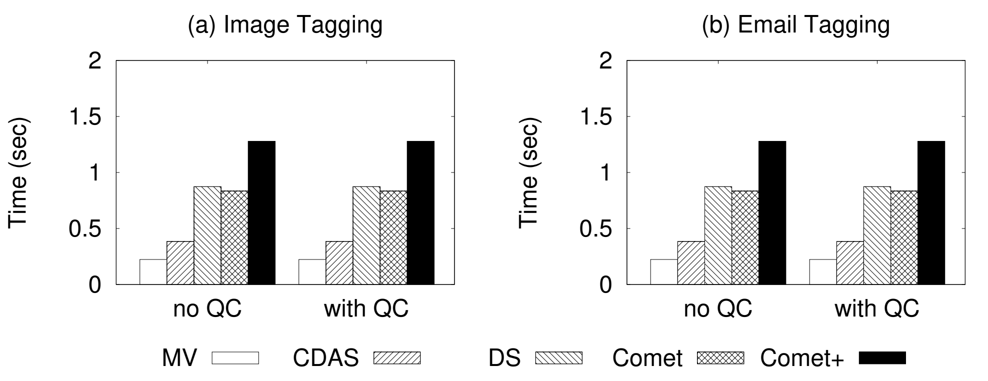


Figure 6.7: Truth Inference (Efficiency Comparisons).

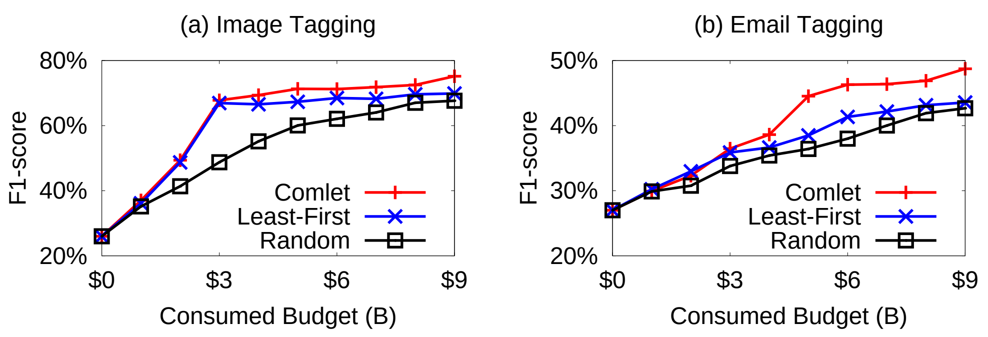


Figure 6.8: Task Assignment (Effectiveness Comparisons).

on single-label tasks, and they are not straightforward to extend to multi-label tasks. Thus we compare with two baseline assignment strategies for multi-label tasks. (1) Random: it randomly selects k tasks; (2) Least-First: it selects k tasks that have been assigned least times.

We use two datasets (image tagging and email tagging) and perform experiments on AMT [1] with quality control as specified in Section 6.7.1. For each dataset, we pay \$0.01 for a worker upon answering a task, and we set the total budget as \$9. That is, each dataset can collect 900 answers (i.e., 900×20 votes) from workers. When a worker comes, we select $k = 3$ tasks and batch them in a HIT for the coming worker. To control the effect of workers, for one dataset, we perform 3 experiments on AMT in parallel, where each one uses a specific assignment strategy.

Effectiveness. We compare the effectiveness of different assignment strategies in Figure 6.8. For a fair comparison, we use Comet's truth inference method for all assignment strategies. For each dataset, we compute the quality as the budget (denoted as B) is varied from \$0 to \$9. It can be seen that Comet outperforms both Least-First and Random. We can also observe in Figures 6.8(a)(b) that Least-First and Comet perform nearly the same in the beginning ($B \in [\$0, \$3]$), and the reason is that at first the two algorithms will typically assign all 300 tasks in the dataset to workers. However, Random performs bad in the beginning, as it chooses tasks randomly, which may select some tasks multiple times while leaving a few tasks not selected. As B increases, Comet can measure the reduction in uncertainty when selecting tasks, thus gradually outperforming other competitors. We can also see that when the budget is consumed (i.e., $B = \$9$), Comet leads more than 5% compared with the best of other competitors, and the quality comparison result can be generally summarized as follows: Comet>Least-First>Random.

Efficiency. We also test the task assignment efficiency for three strategies. For each dataset, we record the worst-case assignment time in all its assignments. In our results, we find that the assignment process of all the methods can be finished within 0.03s. Comet is the least efficient, as it computes the uncertainty reduction for each task, and chooses k tasks with the highest reduction in uncertainty; Least-First needs to decide the tasks that are answered with the least times, which is more expensive than Random.

6.7.5 Scalability on Simulated Data

We evaluate the scalability of truth inference and task assignment on simulated data. For statistical significance, we repeat each experiment for 1000 times and record the average time.

Truth Inference. In Figure 6.9(a), we evaluate the scalability of Comet's truth inference method (Algorithm 12). We generate n tasks, where each $|L_i| = 20$. We then generate $|\mathcal{W}|$ workers, where each task is assigned to 5 randomly selected workers (from $\mathcal{W}$), and workers' answers are randomly generated. We vary $n \in [0, 10^4]$, $|\mathcal{W}| \in \{10, 100, 500\}$ and run truth inference on randomly generated workers' answers. It can be seen from Figure 6.9(a) that (1) for a fixed $|\mathcal{W}|$, the time linearly increases with n; when n is big enough (e.g., 10^4), the time is within 70s, which is efficient (as the truth inference can be done off-line). (2) For a fixed n, the time does not change with varying $|\mathcal{W}|$. It may contradict to the complexity, i.e., $\mathcal{O}(c \cdot |\mathcal{W}| \cdot \sum_{i=1}^{n} |L_i|)$. However, it is an upper bound, and we make further analysis to explain it: for a fixed budget, a larger $|\mathcal{W}|$ means the participation of more workers, thus the average number of tasks a worker has answered is smaller, leading to the fact the computation for each worker is smaller. This may explain why the time is invariant with $|\mathcal{W}|$ in practice.

Task Assignment. In Figure 6.9(b), we evaluate the scalability of Comet's assignment strategy (Section 6.6). We generate n tasks, where each $|L_i| = 20$. Then we randomly generate each $q_{i,j} \in [0, 1]$, and randomly generate the coming worker w's quality $p_w \in [0, 1]$, $r_w \in [0, 1]$. We vary $n \in [0, 10^4]$, $k \in \{10, 50, 100\}$, and run Comet to assign k tasks for worker w, and record the time. It can be observed in Figure 6.9(b) that (1) for a fixed k, the time linearly increases with n, corresponding to the complexity $\mathcal{O}(\sum_{i=1}^{n} |L_i|)$; (2) for a fixed n, the time does not change if k varies, because after we compute the uncertainty reduction of each object, we can use the PICK algorithm [26] to select top-k objects, which is invariant with k.

6.8 Related Works

Since we have reviewed most of the related works of crowdsourcing in Chapter 2, this section only highlights the part related to multi-label tasks.

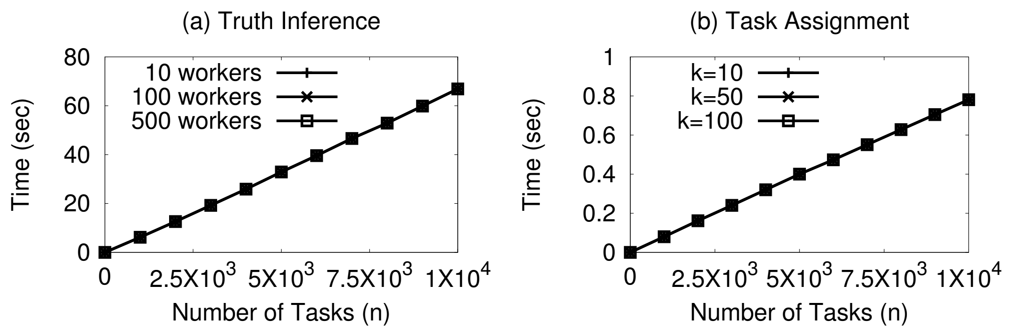


Figure 6.9: Scalability on Simulated Data.

In machine-learning field, multi-label tasks have been widely studied [24, 210, 213] and applied to many applications, e.g., text categorization [42], bioinformatics [22]. They take features of objects as input, and train a classifier based on the training data. Different from them, we address it in crowdsourcing, i.e., the truth is inferred based on workers' answers. Furthermore, the label correlations provided by [24,210,213] can better facilitate our truth inference. In crowdsourcing, multi-label tasks are addressed [149, 222] based on transforming each task to many *independent* single-label tasks, which will incur more latency and budget [51]. Although some recent works [56, 145, 147, 149, 198] focus on publishing multi-label tasks to crowdsourcing platforms, however, this problem is not well addressed and different characteristics of workers are not well captured. Our worker model can better capture a worker's quality in answering multi-label tasks. We also consider label correlations and online task assignment. Note that there are also some other crowdsourcing works [29, 39, 51, 89] on multi-label tasks, but with different objectives, e.g., [29, 39] study how to build a taxonomy tree for an item, and [89] addresses to label POI (points of interest) in spatial crowdsourcing.

6.9 Chapter Summary

In this chapter, we focus on image tagging application, and generalize to the study of multi-label tasks. To be specific, we examine two fundamental problems about multi-label tasks: *Truth Inference Problem* and *Task Assignment Problem*. In the first problem, we model a worker as *Precision* and *Recall*, and develop an iterative approach that captures the relations between truth and workers' qualities. We further incorporate label correlations in computing the truth. In the second problem, when a worker comes, we select the tasks whose uncertainty can be reduced the most for the worker. We perform experiments both on real-world datasets and simulated data, verifying that our proposed approaches are robust, outperforming existing methods with various workers, and also scalable to large datasets.

In future work, we plan to study the following directions: (1) we will study different User Interfaces (UIs) of multi-label tasks, and the relations among the number of labels, quality, and latency, etc; (2) we will also consider the frequency and importance of labels, e.g., the TF-IDF frequency of labels selected by workers, and also the preference of labels indicated by requests; (3) in our experiments, we assume that a small sample of data with ground truth are known in advance, which are used to obtain the correlation matrix; thus it will be an interesting future work to study the case if no ground truth is known in advance. (4) in image tagging application, each task and the candidate labels may be related to different domains, thus we will see whether capturing the domain information (will be discussed in the next chapter) can benefit such application.

In next chapter, we will study how to apply the techniques in task assignment and truth inference to another application, i.e., question answering application. To be specific, the tasks are of diverse domains and workers have diverse qualities over various domains. Thus it is of utter importance and also challenging to consider the domain aware worker model and task model to the design of task assignment and truth inference components.

Chapter 7

A Domain-Aware Task Crowdsourcing System

7.1 Introduction

To tackle complex tasks that are hard for computers (e.g., entity resolution [192, 199] and sentiment analysis [127, 222]), many crowdsourcing platforms (e.g., Amazon Mechanical Turk (AMT) [1] and CrowdFlower [5]) have been recently deployed. These platforms allow tasks to be performed by a huge number of Internet users (or *workers*) with different backgrounds. The increase in the importance of crowdsourcing has attracted a lot of research attention [27,47,48,63,70,88,89,116,214,219,222].

However, workers may yield low quality and a core problem in crowdsourcing is to infer high-quality results from the workers' answers. Different workers may have diverse qualities, and it is important to accurately model a worker's quality. An effective worker model can benefit many important problems in crowdsourcing and we examine three crucial aspects addressed in existing works [27,47,48,63,70,131,219,222] that help to infer high-quality results:

• *Worker Model*: How to represent the quality of a worker that effectively reflects her skills? Existing works simply treat it as a real value [48,70,219] or a matrix [47,222].

• *Truth Inference*: How to obtain the true answer (called *truth*) of a task? To improve the quality, a task may be performed by one or more workers, and thus an important issue is to infer the truth through aggregating workers' answers [47,48,63,131].

• *Task Assignment*: How to assign a task to appropriate workers? As pointed out by [27,63,222], this is often done based on worker model, which reflects her performance statistics shown in her previous tasks. Note that the assignment latency is crucial and *online* task assignment is required to achieve instant assignment.

A common drawback of existing solutions (e.g., solutions in Chapters 3-5) is that they often overlook the worker's ability in different aspects (or *domains*). As a matter of fact, in specific applications, e.g., question answering, workers may have a variety of expertise, skills, and cultural backgrounds; tasks may also be related to different domains, which we call *domain-aware tasks*. Let us consider two workers (A, an *NBA* fan, and B, a frequent moviegoer) and two tasks t_1 and t_2 (which ask workers to select labels of two photos about *Stephen Curry* and *Leonardo DiCaprio*, respectively). Intuitively, A should do better than B in the *sports* domain, while B is a better candidate than A in doing tasks related to *films*. Thus, t_1 and t_2 should be assigned to A and B respectively. However, existing works often neglect the domain information (e.g., the qualities of A and B are modeled as the same values for different tasks [48,70,219]).

The issues of incorporating domain knowledge in the crowdsourcing process have only been recently studied [63,131], where each worker has diverse qualities on different domains. These solutions, while promising, still have room for improvement:

• *Worker Model*: [63] examines the issues of inferring domains of workers and tasks. The solution relies on the text descriptions of tasks – tasks with large text similarity have a higher chance to be classified into the same domain. However, this solution can result in wrong domain classification. For example, the two tasks "*Is Stephen Curry a PF?*" and "*Has Golden State Warriors ever won championships?*" may not be similar (e.g., in terms of Jaccard similarity), yet they are in the *sports* domain. On the other hand, tasks "*Compare the height of Stephen Curry and Kobe Bryant.*" and "*Compare the height of Mount Everest and K2.*" may have a high text similarity, but they are in different domains (i.e., *sports* and *mountains*, respectively). In [131], machine-learning techniques are used to compute *latent domains* of tasks. However, due to the lack of semantics, these *latent domains* are hard to interpret, making it difficult to profile and understand a worker's ability.

• *Truth Inference*: In [63, 131], the problem of using domain information to infer truth has been studied. Typically they exploit a worker's diverse qualities on different domains, and then for a task, it will trust a worker's answer if the worker has high qualities on the domains in that task. However, the workers' qualities are either inaccurately estimated [131], or incorrectly leveraged to compute each task's truth [63]. For example, [131] estimates each task's *latent domains* and each worker's quality for those *latent domains* together, thus the estimation of worker's quality is highly affected by the inaccurate estimation of task's domains; [63] uses the weighted majority voting to infer each task's truth, which is easy to be misled by the answers given by multiple low-quality workers.

• *Task Assignment*: The only work that uses domain information in task assignment is [63]. However, it adopts a fairly simple task assignment method, in which each task is assigned to the *same* number of workers, and the difficulty level of a task is not considered. Moreover, it assigns tasks to a worker such that the worker has the highest qualities to accomplish, which omits the fact the assigned tasks may have already obtained confident and consistent answers.

Hence, there is a need of investigating how to make the best use of domain information in the crowdsourcing process. Our goal is to study an effective method of utilizing domain information to enhance the effectiveness of the three steps above. The main idea is to consult an existing knowledge base (or *KB*), such as Wikipedia [201] and Freebase [71] for obtaining domain information. These KBs are often associated with a large number of categories/topics information, organized in a systematic and hierarchical manner. For example, Freebase [71] contains over $57M$ concepts, encoded by $3G$ facts. We have developed a DOmain-Aware Crowdsourcing System, called DOCS, which taps into this large pool of information, learning the domains of workers and tasks more explicitly.

Figure 7.1 shows the architecture of DOCS, which contains three main modules: **Domain Vector Estimation (DVE)**, **Truth Inference (TI)** and **Online Task Assignment (OTA)**. A requester (who publishes tasks) can specify a set of tasks (with text descriptions) and a budget in DOCS. Then DOCS interacts with knowledge bases and crowdsourcing platforms, respectively. Finally after consuming the budget, the inferred truth for all tasks are returned to the requester. Next, we show how the three modules in DOCS work upon receiving a requester's tasks.

• **DVE**. This module is responsible for estimating the related domains of each task, based on the domain information in a KB. Specifically, an "entity-linking" algorithm [168] can be used, which extracts entities from the text description of each task. A *domain vector* is then computed for these entities, in order to capture how likely a task belongs to each domain mentioned in a KB.

After computing each task's domain vector, the tasks are published to crowdsourcing platforms (e.g., AMT [1]). By interacting with the crowd workers, in general, DOCS needs to handle two types of requests from workers: (1) a worker accomplishes tasks and submits answers; (2) a worker comes and re-

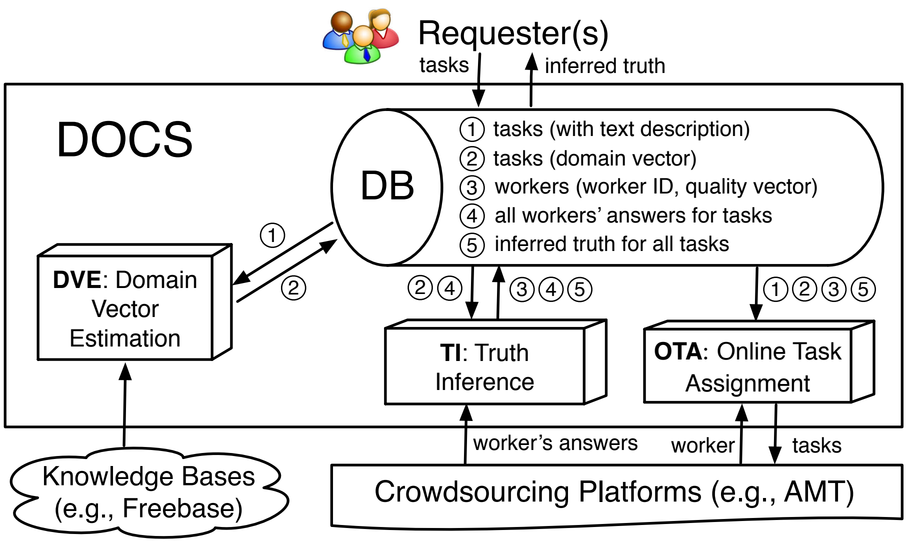

Requester(s)
tasks
inferred truth
DOCS
DB
① tasks (with text description)
② tasks (domain vector)
③ workers (worker ID, quality vector)
④ all workers' answers for tasks
⑤ inferred truth for all tasks
①
②
DVE: Domain Vector Estimation
②④
③④⑤
①②③⑤
TI: Truth Inference
OTA: Online Task Assignment
worker's answers
worker
tasks
Knowledge Bases (e.g., Freebase)
Crowdsourcing Platforms (e.g., AMT)

Figure 7.1: The Architecture of DOCS.

quests tasks.

• **TI**. When a worker accomplishes tasks and submits answers, the module first stores the worker's answers into database and then infers each task's truth and each worker's model based on two principles: (1) a worker's answer is trusted, if she is a domain expert on her submitted tasks; and (2) a worker is a domain expert if she often correctly answers tasks related to that domain.

• **OTA**. When a worker comes and requests new tasks, this module assigns tasks to her. A poor assignment may not only waste budget and time, but also hurt the quality of inference results which depend on workers' answers. To judiciously assign tasks, the module makes decisions based on three factors: (1) the worker's quality, (2) the domain vectors of tasks, and (3) how confident each task's truth can be inferred from previously received answers. Intuitively, we assign a task to the worker if the task's domains are the worker's expertise and its truth cannot be confidently inferred. The assignment is done *online*, i.e., tasks will be assigned to the worker instantly.

However, designing the three modules above is not straightforward. We technically address the above challenges as follows:

For **DVE**, although we can extract entities from a task based on existing entity linking algorithm [168], ambiguities exist when we link each extracted entity to real-world concepts (e.g., pages in Wikipedia). For example, the entity *Michael Jordan* can either refer to the famous basketball player (`https://en.wikipedia.org/wiki/Michael_Jordan`), or the computer scientist (`https://en.wikipedia.org/wiki/Michael_I._Jordan`) in Wikipedia. Suppose there are u entities in a task, and each entity can be linked to 3 concepts, then there are 3^u possible linkings from the entities to concepts, which is exponential. Thus deriving a domain vector for a task in a straightforward way involves aggregating an exponential number of such linkings. We propose an algorithm that can reduce the complexity from exponential to polynomial (Section 7.3).

For **TI**, it is challenging to infer each task's truth correctly, as it highly depends on workers' qualities (which are unknown). Intuitively, we exploit the inherent relations between workers' qualities and tasks' truth, and finally devise an iterative approach that collectively infers those parameters. We also study how to maintain each worker's quality in the long run and devise update policies for the incremental inference algorithms (Section 7.4).

Finally, for the **OTA** module, we have studied how to use the three factors above (that affect task assignment), in order to estimate the *benefit* of assigning each task to the worker, by considering *if* the task is answered by the worker; then we assign a set of k tasks that attain the highest benefits. There are several challenges. (1) How to define the *benefit* function? (2) How to estimate the answer given by the worker? (3) Typically for better user interaction, a set of k tasks (e.g., $k = 20$ in [195, 222]) will be batched together and assigned to the worker. To select the optimal k tasks out of all (say, n) tasks, there are $\binom{n}{k}$ possible combinations that have to be considered, then how to efficiently compute the optimal assignment? We have developed an optimal and linear algorithm to support this complex assignment process (Section 7.5).

To summarize, our main goal is to study the impact of using domain knowledge in the crowdsourcing process. We further examine how to use a knowledge base (KB) to realize this goal. We examine how to use a KB effectively and efficiently in the three key procedures of crowdsourcing, namely, (1) domain vector estimation (**DVE**), (2) truth inference (**TI**), and (3) online task assignment (**OTA**). To our understanding, no previous work has examined the use of domain knowledge in such a comprehensive manner. We present a simple architecture to integrate these processes, and our extensive experiments show that our solution outperforms state-of-the-art methods, i.e., [27,47,48,63,131,222].

7.2 Data Model

Definition 7.1 (Domain)**.** *Let $\mathcal{D} = \{d_1, d_2, \ldots, d_m\}$ denote the domain set with $|\mathcal{D}| = m$ domains.*

An example domain set is $\mathcal{D}$ ={*politics*, *sports*, *films*}. The domain set is used to interpret tasks and profile workers, which can be obtained by existing knowledge bases or question answering systems, e.g., main topics in Wikipedia [201], domains in Freebase [71], or categories in Yahoo Answers [204]. The reason for using general topics is that they can interpret a task and profile a worker in a fine-grained manner. We record the worker's familiar domains, which can be further used when the same worker comes in the future.

Definition 7.2 (Task, Domain Vector)**.** *A requester publishes n tasks, denoted as $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$. Each task $t_i \in \mathcal{T}$ has a text description, followed by ℓ_{t_i} possible choices. Each task t_i is modeled as a domain vector $\boldsymbol{r}^{t_i} = [\ r_1^{t_i}, r_2^{t_i}, \ \ldots\ , r_m^{t_i}\ ]$, where each $r_k^{t_i} \in [0,1]$ ($1 \leq k \leq m$) and $\sum_{k=1}^{m} r_k^{t_i} = 1$. The domain vector $\boldsymbol{r}^{t_i}$ represents the distribution that task t_i is related to each domain in $\mathcal{D}$. A higher value of $r_k^{t_i}$ means that task t_i is more related to domain d_k.*

In this chapter, we focus on multiple-choice tasks. Now let us consider a task t_1: *"Did Michael Jordan win more NBA championships than Kobe Bryant?"*, and

Table 7.1: Workers' Qualities and Answers for Task t_1.

Worker	**Worker's Quality**	**Worker's Answer for Task t_1**
w_1	$\boldsymbol{q}^{w_1} = [\,0.3, 0.9, 0.6\,]$	$v_1^{w_1} = 1$ (*'yes'*)
w_2	$\boldsymbol{q}^{w_2} = [\,0.9, 0.6, 0.3\,]$	$v_1^{w_2} = 2$ (*'no'*)
w_3	$\boldsymbol{q}^{w_3} = [\,0.6, 0.3, 0.9\,]$	$v_1^{w_3} = 2$ (*'no'*)

the same $\mathcal{D}$ as above. The task has $\ell_{t_1} = 2$ choices: {*yes*, *no*}. From the text description we know that the task is related to domains *sports* and *films* in $\mathcal{D}$ (note that *Michael Jordan* starred in the film "*Space Jam*" in 1996), and it is more relevant with *sports*, thus a reasonable domain vector for t_1 is $\boldsymbol{r}^{t_1} = [0, 0.78, 0.22]$ (we will show how to compute it in Section 7.3). We use bold font to represent a vector (e.g., $\boldsymbol{r}^{t_1}$) and the symbol $|\cdot|$ to get the size of a vector or set (e.g., $|\boldsymbol{r}^{t_1}| = |\mathcal{D}| = m$).

Definition 7.3 (Worker, Quality Vector). *Let $\mathcal{W}$ denote the worker set. Each worker $w \in \mathcal{W}$ is modeled as a quality vector $\boldsymbol{q}^w = [\,q_1^w, q_2^w, \ldots, q_m^w\,]$, where each $q_k^w \in [0,1]$ indicates the expertise (accuracy) of worker w in answering tasks in domain d_k ($1 \le k \le m$). A higher value q_k^w means that worker w has more expertise on domain d_k.*

Considering the same $\mathcal{D}$ above, if worker w is an enthusiastic sports-fan and movie-goer, while pays no attention to politics, then a proper quality vector for w is $\boldsymbol{q}^w = [0.3, 0.8, 0.8]$. Note that a worker can be an expert in multiple domains.

Definition 7.4 (Answer, Truth). *Workers can come to the DOCS and answer tasks. Let ℓ_{t_i} denote the number of possible answers for task t_i, and v_i^w denote the answer given by worker w for task t_i, i.e., $1 \le v_i^w \le \ell_{t_i}$. We assume that a worker can answer a task at most once. Each task t_i has a (ground) truth, or true answer, denoted as v_i^* ($1 \le v_i^* \le \ell_{t_i}$).*

For the above example task t_1 and $\ell_{t_1} = 2$, suppose three workers (w_1, w_2 and w_3) give their answers in Table 7.1: $v_1^{w_1} = 1$ (*yes*), and $v_1^{w_2} = v_1^{w_3} = 2$ (*no*). The truth of t_1 is $v_1^* = 1$ (as *Michael* won 6 *championships* while *Kobe* won 5). Note that the truth v_1^* is unknown to us and we infer v_1^* based on workers' answers.

Table 7.2 summarizes the notations used in the chapter.

Table 7.2: Notations Used in Chapter 7.

Notation	**Description**
domains and tasks	
m	the number of domains ($m = 26$ in DOCS)
d_i	the i-th domain ($1 \le i \le m$), $\mathcal{D} = \{d_1, d_2, \ldots, d_m\}$
t_i	the i-th task ($1 \le i \le n$), $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$
$\boldsymbol{r^{t_i}}$	domain vector for task $t_i \in \mathcal{T}$, $\boldsymbol{r^{t_i}} = [\, r_1^{t_i}, r_2^{t_i}, \ldots, r_m^{t_i} \,]$
entities and concepts (w.r.t. each entity) for a task t	
e_i	the i-th entity for a task t, and $E_t = \{e_1, e_2, \ldots, e_{\lvert E_t \rvert}\}$
c	$c = \max_{1 \le i \le \lvert E_t \rvert} \lvert \boldsymbol{p_i} \rvert$ ($c = 20$ in DOCS)
$\boldsymbol{p_i}$	the distribution of concepts for entity e_i ($1 \le i \le \lvert E_t \rvert$)
$\boldsymbol{h_{i,j}}$	the indicator concept vector (size m) for j-th concept in e_i
workers, answers, and truth	
$\mathcal{W}$	the set of workers, where each worker $w \in \mathcal{W}$
$\boldsymbol{q^w}$	worker w's quality vector, $\boldsymbol{q^w} = [\, q_1^w, q_2^w, \ldots, q_m^w \,]$
v_i^w	worker w's answer for task t_i, and $1 \le v_i^w \le \ell_{t_i}$
$V^{(i)}$	the set of workers' answers for task t_i, i.e., $V^{(i)} = \{v_i^w\}$
$\mathcal{T}^{(w)}$	the set of tasks answered by worker w, i.e., $\mathcal{T}^{(w)} = \{t_i\}$
v_i^*	truth for task t_i ($1 \le i \le n$) and $1 \le v_i^* \le \ell_{t_i}$

7.3 Domain Vector Estimation

We propose a two-step framework to compute $\boldsymbol{r^t}$ for task t.

Step 1: Extracting Entities, Concepts, and Indicator Vectors. Based on the advances in information retrieval [38,159,168,178], we can leverage existing "entity linking" techniques [168] to detect entities in a task. Each detected entity can be linked to a set of possible concepts, which forms a probability distribution where each concept is associated with a probability that indicates the link from entity to concept is correct (by considering the semantic meanings in the text). For each concept, we can then use the hierarchical structure of a knowledge base to compute an indicator vector, expressing the domains in $\mathcal{D}$ that are

Table 7.3: The Information Generated for Task t.

Entity	Concept (Page in Wikipedia)	Prob.	Indicator Vector
e_1: *Michael Jordan*	Michael_Jordan	$p_{1,1} = 0.7$	$\boldsymbol{h_{1,1}} = [\,0, 1, 1\,]$
	Michael_I._Jordan	$p_{1,2} = 0.2$	$\boldsymbol{h_{1,2}} = [\,0, 0, 0\,]$
	Michael_B._Jordan	$p_{1,3} = 0.1$	$\boldsymbol{h_{1,3}} = [\,0, 0, 1\,]$
e_2: *NBA*	National_Basketball_Association	$p_{2,1} = 0.8$	$\boldsymbol{h_{2,1}} = [\,0, 1, 0\,]$
	National_Bar_Association	$p_{2,2} = 0.2$	$\boldsymbol{h_{2,2}} = [\,0, 0, 0\,]$
e_3: *Kobe Bryant*	Kobe_Bryant	$p_{3,1} = 1.0$	$\boldsymbol{h_{3,1}} = [\,0, 1, 0\,]$

Note: the concept can be redirected to the corresponding wikipedia page by adding the prefix "`https://en.wikipedia.org/wiki/`". For example, "Michael_Jordan" corresponds to the url "`https://en.wikipedia.org/wiki/Michael_Jordan`".

related to the concept.

We use the task t_1 (denoted as t in this section) and $\mathcal{D}$ in Section 7.2 as an example. Table 7.3 shows the generated information. We denote $E_t = \{e_1, e_2, \ldots, e_{|E_t|}\}$ as the set of detected entities for task t, e.g., $|E_t| = 3$ and $e_1 =$ *Michael Jordan*. For an entity $e_i \in E_t$, the distribution of all its possible correct concepts is denoted as $\boldsymbol{p_i} = [\,p_{i,1}, p_{i,2}, \ldots, p_{i,|\boldsymbol{p_i}|}]$, where each $p_{i,j}$ $(1 \le j \le |\boldsymbol{p_i}|)$ is the probability that the link from e_i to its j-th concept is correct. For example, for e_2 (*NBA*), we get $\boldsymbol{p_2} = [0.8, 0.2]$ for its two concepts. For the j-th concept in e_i, the computed indicator vector is denoted as $\boldsymbol{h_{i,j}} = [\,h_{i,j,1}, h_{i,j,2}, \ldots, h_{i,j,m}\,]$, where each $h_{i,j,k} = 1$ (0) means that the j-th concept in e_i is related (unrelated) to domain d_k. For example, as *Michael_B._Jordan* is an American actor, thus it is only related to domain *films* (i.e., d_3), and $\boldsymbol{h_{1,3}} = [0, 0, 1]$.

Step 2: Computing Domain Vector. We aggregate all entities of a task to compute its domain vector, by considering the correctness probability from an entity to a concept and the indicator vector of each concept in an entity. However, it is prohibitively expensive to compute the best domain vector (see Section 7.3.1).

7.3.1 Challenges in Computing Domain Vector

For a task t, based on step 1 we have all detected entities E_t, the distribution $\boldsymbol{p_i}$ of concepts for an entity e_i, and each concept's indicator vector $\boldsymbol{h_{i,j}}$. To compute the domain vector $\boldsymbol{r^t}$, we consider all correct linkings between entities and concepts. For example, in Table 7.3, one possible linking from the three entities in E_t to concepts is: e_1–*"Michael_B._Jordan"*, e_2–*"National_Basketball_ Association"*, e_3–*"Kobe_Bryant"*. The correctness of the linking is $p_{1,3} \cdot p_{2,1} \cdot p_{3,1}$ = 0.08. Under this linking, the aggregated indicator vector is $\boldsymbol{h_{1,3}} + \boldsymbol{h_{2,1}} + \boldsymbol{h_{3,1}} = [0, 2, 1]$, which counts the number of related concepts in each domain, by considering all entities in task t. As the domain vector is a distribution, it is then normalized as $\frac{\boldsymbol{h_{1,3}}+\boldsymbol{h_{2,1}}+\boldsymbol{h_{3,1}}}{\sum_{k=1}^{m}(h_{1,3,k}+h_{2,1,k}+h_{3,1,k})} = [0, \frac{2}{3}, \frac{1}{3}]$. From the above analysis, we know that for a possible linking, we can derive its corresponding correctness probability and normalized vector.

For ease of presentation, we use $\boldsymbol{\pi} = [\pi_1, \pi_2, \ldots, \pi_{|E_t|}]$ to denote a possible linking, which means that e_i $(1 \le i \le |E_t|)$ is linked to the π_i-th possible concept of e_i. For example, the above linking corresponds to $\boldsymbol{\pi} = [3, 1, 1]$. Let $\Omega = \{\boldsymbol{\pi}\}$ denote a set containing all possible linkings, so $|\Omega| = \prod_{i=1}^{|E_t|} |\boldsymbol{p_i}|$. In this chapter, we assume the entity is linked into different concepts independently. We will consider the issues of correlation among concepts in the future. Then for each linking $\boldsymbol{\pi} \in \Omega$, we can derive its corresponding correctness probability $\Pr(\boldsymbol{\pi}) = \prod_{i=1}^{|E_t|} p_{i,\pi_i}$ and normalized vector $\boldsymbol{v_\pi} = (\sum_{i=1}^{|E_t|} \boldsymbol{h_{i,\pi_i}}) / (\sum_{k=1}^{m} \sum_{i=1}^{|E_t|} h_{i,\pi_i,k})$. As the normalized vector $\boldsymbol{v_\pi}$ is a distribution that reflects the degree of relatedness of task t to each domain w.r.t. the linking $\boldsymbol{\pi}$, thus by considering all possible $\boldsymbol{\pi} \in \Omega$, we define the domain vector $\boldsymbol{r^t}$ as the expected normalized vector, i.e.,

$$\boldsymbol{r^t} = \sum_{\boldsymbol{\pi} \in \Omega} \boldsymbol{v_\pi} \cdot \Pr(\boldsymbol{\pi}) = \sum_{\boldsymbol{\pi} \in \Omega} \frac{\sum_{i=1}^{|E_t|} \boldsymbol{h_{i,\pi_i}}}{\sum_{k=1}^{m} \sum_{i=1}^{|E_t|} h_{i,\pi_i,k}} \cdot \prod_{i=1}^{|E_t|} p_{i,\pi_i}. \tag{7.1}$$

Take the example in Table 7.3. By enumerating all possible $\boldsymbol{\pi} \in \Omega$ ($|\Omega|$=3 · 2 · 1=6) as Equation 7.1, the domain vector is computed as $\boldsymbol{r^t} = [0, 0.78, 0.22]$. However, directly computing $\boldsymbol{r_t}$ via Equation 7.1 is expensive. Let $c = \max_{1 \le i \le |E_t|} |\boldsymbol{p_i}|$, i.e., the maximum number of concepts in all entities, then it takes $\mathcal{O}(|\Omega| \cdot |E_t| \cdot m) = \mathcal{O}(c^{|E_t|} \cdot |E_t| \cdot m)$ time, which is exponential.

7.3.2 Our Solution

We devise our solution in Algorithm 14, which computes the domain vector accurately by reducing the complexity from $\mathcal{O}(c^{|E_t|} \cdot |E_t| \cdot m)$ (exponential) to $\mathcal{O}(c \cdot m^2 \cdot |E_t|^3)$ (polynomial). The basic idea is that although the number of possible linkings ($|\Omega|$) is exponential, the number of possible normalized vectors is bounded. For example, for the k'-th element in the normalized vector, i.e., $(\sum_{i=1}^{|E_t|} h_{i,\pi_i,k'}) / (\sum_{k=1}^{m} \sum_{i=1}^{|E_t|} h_{i,\pi_i,k})$, as each $h_{*,*,*} \in \{0,1\}$, if we consider its numerator and denominator respectively, there are at most $(|E_t|+1) \cdot (m \cdot |E_t|+1)$ possible values for that element. This inspires us to compute $\boldsymbol{r^t}$ from the perspective of normalized vectors in Algorithm 14.

To be specific, Algorithm 14 takes m iterations, where for the k-th iteration (lines 5-24), it computes the k-th element of $\boldsymbol{r^t}$, i.e., r^t_k. To achieve this, we use a hash-map (`M`), whose *keys* are the possible (numerator, denominator) combinations (denoted by (nm,dm)), and the corresponding *value* for a *key* (nm,dm) is the aggregated probability for nm/dm. In order to compute r^t_k, we consider each entity iteratively. In the i-th iteration (lines 7-17), it derives an `M`, whose *keys* are the possible (nm,dm) by considering the first i entities (i.e., from e_1 to e_i). In doing so, we leverage the derived `M` in last iteration (i.e., considering the first $i-1$ entities) and directly applies the $\boldsymbol{p_i}$ and $\boldsymbol{h_{i,*}}$ of the i-the entity e_i to generate a new temporary hash-map `tmpM`. To be specific, for each *key* (nm,dm) in `M`, it is updated based on concepts in e_i: for the j-th concept, the *key* (nm,dm) becomes new *key* (nm+$h_{i,j,k}$, dm+$x_{i,j}$), note $x_{i,j} = \sum_{k=1}^{m} h_{i,j,k}$, which is initially stored in line 1, and the *value* (or aggregated probability) is multiplied by $p_{i,j}$ and added to the *value* of new *key* in `tmpM`. At last `tmpM` is assigned to `M` for the next iteration (line 17). After all entities are considered ($|E_t|$ iterations), we can finally use the information in the derived `M` to compute r^t_k. (lines 19-24).

Running Example. Figure 7.2 shows how to compute r^t_2 in Table 7.3. The i-th layer shows the derived `M` after considering e_i. We use '*key:value*' to represent each data in the hash-map. Initially for $i = 1$ (e_1), its three concepts (with $\boldsymbol{p_1}$,

Algorithm 14 Domain Vector Computation (Chapter 7).

```
Input: $E_t$, $\boldsymbol{p_i}$ $(1 \leq i \leq |E_t|)$, $\boldsymbol{h_{i,j}}$ $(1 \leq i \leq |E_t|, 1 \leq j \leq |\boldsymbol{p_i}|)$
Output: $\boldsymbol{r^t}$
1: $x_{i,j} = \sum_{k=1}^{m} h_{i,j,k}$ for $1 \leq i \leq |E_t|, 1 \leq j \leq |\boldsymbol{p_i}|$; // pre-computation
2: $\boldsymbol{r^t} = [0, 0, \ldots, 0]$; // a vector of size $m$ with all 0 elements
3: M = hash-map(); // we use M[key] to visit the value of the key (a 2-tuple)
4: for $k = 1$ to $m$ (iterate over all domains) do
5:   M[ (0,0) ] = 1; // initialize the hash-map M
6:   for $i = 1$ to $|E_t|$ (iterate over all entities) do
7:     tmpM = hash-map(); // another hash-map, similar to M
8:     for (nm, dm) ∈ M (iterate over all keys in M) do
9:       $value$ = M[ (nm, dm) ]; // get the value of the key: (nm,dm)
10:      for $j = 1$ to $|\boldsymbol{p_i}|$ (iterate over all concepts for entity $e_i$) do
11:        if $(\text{nm} + h_{i,j,k}, \text{dm} + x_{i,j}) \notin$ tmpM then
12:          tmpM[ $(\text{nm} + h_{i,j,k}, \text{dm} + x_{i,j})$ ] = 0;
13:        end if
14:        tmpM[ $(\text{nm} + h_{i,j,k}, \text{dm} + x_{i,j})$ ] += $value \cdot p_{i,j}$;
15:      end for
16:    end for
17:    M = tmpM; // assign tmpM to M for next iteration
18:  end for
19:  for (nm, dm) ∈ M (iterate over all keys in M to compute the value of $r_k^t$) do
20:    if dm $\neq 0$ then
21:      $r_k^t$ += (nm/dm) · M[ (nm, dm) ]; // aggregate the value of $r_k^t$
22:    end if
23:  end for
24: end for
25: return $\boldsymbol{r^t}$;
```

$\boldsymbol{h_{1,*}}$) form M. For e_2, based on the derived M in last iteration, it is updated to a new one. For example, (1, 2):0.7 is updated to (1+$h_{2,1,2}$, 2+$x_{2,1}$):0.7·$p_{2,1}$=(2,3):0.56 and (1+$h_{2,2,2}$, 2+$x_{2,2}$):0.7·$p_{2,2}$=(1,2):0.14. Moreover, as (0,1):0.1 can also be similarly updated as (1,2):0.08, thus the *value* for the *key* (1,2) is aggregated as 0.14+0.08=0.22. After all 3 entities are considered, r_2^t = 0.78 is finally derived based on the final M. Following this, we can compute $\boldsymbol{r^t} = [0, 0.78, 0.22]$.

Time Complexity of Algorithm 14. First the calculation of $x_{*,*}$ takes $\mathcal{O}(c \cdot m \cdot |E_t|)$ time, where $c = \max_{1 \leq i \leq |E_t|} |\boldsymbol{p_i}|$. Then it computes each element in $\boldsymbol{r^t}$ iteratively. To compute a r_k^t $(1 \leq k \leq m)$, we iteratively consider $|E_t|$ entities:

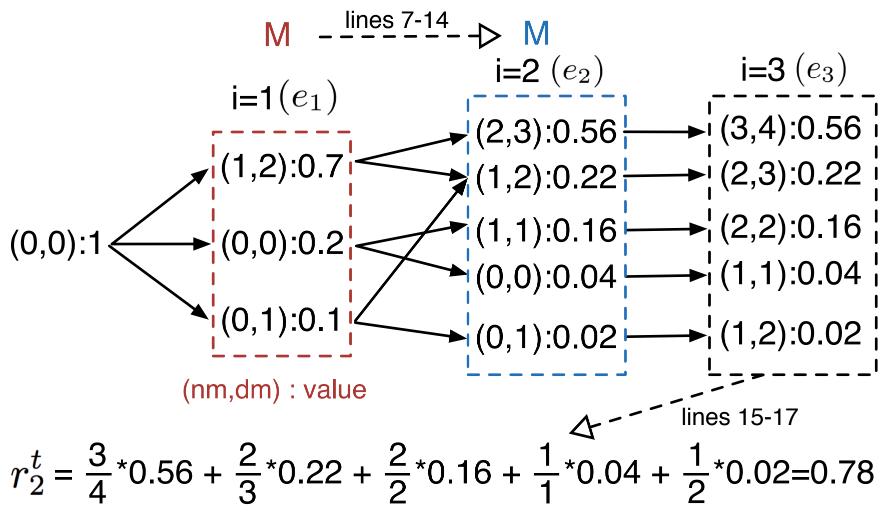


Figure 7.2: A Run of Computing r^t_2.

for the i-th iteration, we leverage the already derived M to further update itself to a new one, by considering all concepts in e_i. In each iteration, as the number of *keys* in M is $\mathcal{O}(m \cdot |E_t|^2)$, then it takes $\mathcal{O}(c \cdot m \cdot |E_t|^2)$. So computing a r^t_k takes $\mathcal{O}(c \cdot m \cdot |E_t|^3)$, and the total time complexity of deriving $\boldsymbol{r^t}$ is $\mathcal{O}(c \cdot m^2 \cdot |E_t|^3)$.

The Implementations of DVE in DOCS. We adopt Freebase [71], a large, reliable, and curated knowledge base. We construct $\mathcal{D}$ based on 26 domains in Yahoo Answers [204], which consists of a wide range of topics, such as *Sports*, *Politics*. We manually map each of the 26 domains to the respective domain(s) in Freebase. Next we discuss how to compute E_t, $\boldsymbol{p_i}$, $\boldsymbol{h_{i,j}}$ and $\boldsymbol{r^t}$.

We use an open-source entity linking tool Wikifier [38, 159], which can detect entities E_t in a task t. For each entity e_i, it links to the top 20 possible concepts (or pages) [159], by considering features such as the frequency of the linking and the string similarity between concepts and e_i. Then it computes a probability distribution $\boldsymbol{p_i}$ (of size 20), which indicates how probable each possible concept is correctly linked to e_i in the task. For each concept (in Wikipedia), which can be redirected to the corresponding Freebase concept using API, we compute the indicator vector $\boldsymbol{h_{i,j}}$ (of size 26) by considering whether or not each domain in $\mathcal{D}$ is related to the given concept in Freebase. Note that this can be obtained directly from the corresponding concept page in Freebase. Finally, based on E_t, $\boldsymbol{p_i}$ and $\boldsymbol{h_{i,j}}$ ($1 \le i \le |E_t|, 1 \le j \le |\boldsymbol{p_i}|$), Algorithm 14 is leveraged to compute the task t's domain vector $\boldsymbol{r^t}$.

7.4 Truth Inference

In this section, we study the truth inference problem. The **Input** of the problem includes: (1) tasks' domain vectors ($\boldsymbol{r^{t_i}}$ for $1 \le i \le n$) and (2) all workers' answers. The **Output** is each task's inferred truth (and we can also derive each worker's quality vector). To solve it, we first introduce an iterative approach (Section 7.4.1), and then discuss how to use it in practice (Section 7.4.2).

7.4.1 Iterative Approach

We observe that there are two kinds of relations between workers' qualities and tasks' truth: (1) *given a task t, if the worker's quality values for t's related domains are high, then her answer is likely to be the truth for t*; (2) *given a worker w, if w often answers tasks correctly related to a domain, then w has a high quality for that domain*. Based on these intuitions, we develop an iterative approach, which updates the sets of parameters for tasks and workers until convergence is reached. Here, we use $\boldsymbol{q^w}$ to denote a worker w's quality; $\boldsymbol{s_i} = [\, s_{i,1}, s_{i,2}, \ldots, s_{i,\ell_{t_i}} \,]$ is task t_i's probabilistic truth, where $s_{i,j}$ ($1 \le j \le \ell_{t_i}$) is the chance that the j-th choice is the truth for task t_i. We use $V^{(i)}$ to denote the set of workers' answers for task t_i. For example, in Table 7.1, $V^{(1)} = \{v_1^{w_1}, v_1^{w_2}, v_1^{w_3}\}$. We denote o_i as task t_i's true domain. Based on the domain vector $\boldsymbol{r^{t_i}}$, we have $\Pr(o_i = k) = r_k^{t_i}$.

To be specific, in our iterative approach, each iteration contains two steps: in step 1, each task's probabilistic truth $\boldsymbol{s_i}$ is inferred based on workers' qualities ($\boldsymbol{q^w}$); then step 2 reversely infers each worker's quality based on the tasks' probabilistic truth. It will iterate until convergence. Finally, we infer the truth for each task t_i as $v_i^* = \arg\max_{1 \le j \le \ell_{t_i}} s_{i,j}$ (detailed algorithm can be found in Algorithm 15). We first detail the two steps, and then analyze the *Initialization* and *Time Complexity*, respectively.

Algorithm 15 Iterative Truth Inference (Chapter 7).

```
Input: workers' answers $V^{(i)}$, tasks' domain vectors $\boldsymbol{r^{t_i}}$ $(1 \le i \le n)$
Output: truth $\boldsymbol{s_i}$ $(1 \le i \le n)$, worker's quality $\boldsymbol{q^w}$ $(w \in \mathcal{W})$
 1: Initialize $\boldsymbol{q^w}$ for $w \in \mathcal{W}$;
 2: while true do
 3:   // Step 1: Inferring the Truth
 4:   for $1 \le i \le n$ do
 5:     for $1 \le k \le m, 1 \le j \le \ell_{t_i}$ do
 6:       $\mathcal{M}^{(i)}_{k,j} = \Pr(v_i^* = j \mid o_i = k, V^{(i)})$; // Eq. 7.3 and 7.4
 7:     end for
 8:     $\boldsymbol{s_i} = \boldsymbol{r^{t_i}} \times \mathcal{M}^{(i)}$; // Matrix Multiplication
 9:   end for
10:   // Step 2: Estimating Worker Quality
11:   for $w \in \mathcal{W}$ do
12:     Compute $\boldsymbol{q^w}$ based on Eq. 7.5;
13:   end for
14:   // Check for Convergence
15:   if Converged then
16:     break;
17:   end if
18: end while
19: return $\boldsymbol{s_i}$ for $1 \le i \le n$ and $\boldsymbol{q^w}$ for $w \in \mathcal{W}$;
```

Step 1: Inferring the Truth ($\boldsymbol{q^w} \to \boldsymbol{s_i}$). In general, the probabilistic truth $\boldsymbol{s_i}$ can be expressed by considering all domains:

$$s_{i,j} = \Pr(v_i^* = j \mid V^{(i)}) = \sum_{k=1}^{m} r_k^{t_i} \cdot \Pr(v_i^* = j \mid o_i = k, V^{(i)}). \tag{7.2}$$

For simplicity, we denote $\mathcal{M}^{(i)}_{k,j} = \Pr(v_i^* = j \mid o_i = k, V^{(i)})$. Note that $\mathcal{M}^{(i)}$ is a matrix of size $m \times \ell_{t_i}$, where each row $\mathcal{M}^{(i)}_{k,\bullet}$ in it represents the distribution of truth computed for the k-th domain, then $\boldsymbol{s_i}$ can be computed by considering t_i's domain vector: $\boldsymbol{s_i} = \boldsymbol{r^{t_i}} \times \mathcal{M}^{(i)}$ via matrix multiplication. In order to compute

$\mathcal{M}_{k,j}^{(i)}$, we adopt two typical assumptions used in existing works [48,127,131]: (1) workers give their answers independently and (2) the priors are uniform (i.e., $\Pr(v_i^* = j) = 1/\ell_{t_i}$). Then we can derive

$$\mathcal{M}_{k,j}^{(i)} = \frac{\prod_{v_i^w \in V^{(i)}} \Pr(v_i^w \mid o_i = k, v_i^* = j)}{\sum_{j'=1}^{\ell_{t_i}} \prod_{v_i^w \in V^{(i)}} \Pr(v_i^w \mid o_i = k, v_i^* = j')}. \tag{7.3}$$

Since there are ℓ_{t_i} possible answers for task t_i, to capture worker w's ability, similar to [219,222], we compute the probability that the worker answers *incorrectly* (i.e., $1 - q_k^w$) for those $(\ell_{t_i} - 1)$ *incorrect* answers using a uniform distribution. Let $\mathbb{1}_{\{\cdot\}}$ denote an indicator function which returns 1 if the argument is true; 0 otherwise. For example, $\mathbb{1}_{\{2=5\}} = 0$ and $\mathbb{1}_{\{5=5\}} = 1$. Then we have

$$\Pr(v_i^w \mid o_i = k, v_i^* = j) = (q_k^w)^{\mathbb{1}_{\{v_i^w = j\}}} \cdot \Big(\frac{1 - q_k^w}{\ell_{t_i} - 1}\Big)^{\mathbb{1}_{\{v_i^w \neq j\}}}. \tag{7.4}$$

This means that the probability that w answers correctly for a task with domain k is q_k^w, and likewise, the probability that w gives a specific incorrect answer for that task is $(1 - q_k^w)/(\ell_{t_i} - 1)$.

Running Example. We use an example to show that step 1 can satisfy the 1st relation. We compute $\boldsymbol{s_1}$ for task t_1 in Table 7.3. As the domain vector $\boldsymbol{r^{t_1}} = [0, 0.78, 0.22]$, and we take workers' qualities $(\boldsymbol{q^{w_1}}, \boldsymbol{q^{w_2}}, \boldsymbol{q^{w_3}})$ and answers $(V^{(1)})$ in Table 7.1. We first compute each vector $\mathcal{M}_{k,\bullet}^{(1)}$, e.g., for $\mathcal{M}_{2,\bullet}^{(1)} = [\mathcal{M}_{2,1}^{(1)}, \mathcal{M}_{2,2}^{(1)}]$, we derive $\mathcal{M}_{2,1}^{(1)} = \frac{q_2^{w_1}(1-q_2^{w_2})(1-q_2^{w_3})}{q_2^{w_1}(1-q_2^{w_2})(1-q_2^{w_3}) + (1-q_2^{w_1}) q_2^{w_2} q_2^{w_3}} = 0.93$ and $\mathcal{M}_{2,2}^{(1)} = 0.07$. Similarly we derive $\mathcal{M}_{1,\bullet}^{(1)} = [0.03, 0.97]$ and $\mathcal{M}_{3,\bullet}^{(1)} = [0.28, 0.72]$. Then we compute $s_{1,j}$ $(1 \le j \le 2)$ as $s_{1,1} = \sum_{k=1}^{3} r_k^{t_1} \cdot \mathcal{M}_{k,1}^{(1)} = 0.79$, and similarly $s_{1,2} = 0.21$. Note that although two workers w_2, w_3 answer *"no"* to t_1, and only one worker w_1 answers *"yes"* to t_1, the computed truth $\boldsymbol{s_1} = [0.79, 0.21]$ tends to be *"yes"*. The reason is that (1) the task t_1 is more related to domain *"sports"* (0.78 in $\boldsymbol{r^{t_1}}$), and (2) w_1 has a high quality (0.9) for domain *"sports"* while w_2 and w_3 have low qualities (0.6 and 0.3) for it, making w_1's answer more reliable.

Step 2: Estimating Worker Quality ($s_i \to q^w$). We now estimate q^w based on the computed $s_{i,j}$ in step 1. As q_k^w denotes worker w's quality for the k-th domain, which is formally defined as the fraction of tasks in domain d_k that w has answered correctly: $q_k^w = \frac{\sum_{t_i \in \mathcal{T}^{(w)}} \mathbb{1}_{\{o_i=k\}} \cdot \mathbb{1}_{\{v_i^w = v_i^*\}}}{\sum_{t_i \in \mathcal{T}^{(w)}} \mathbb{1}_{\{o_i=k\}}}$, where $\mathcal{T}^{(w)}$ denotes the set of tasks answered by worker w, e.g., in Table 7.1, $\mathcal{T}^{(w_1)} = \{t_1\}$. However, q_k^w is hard to compute directly, as task t_i's true domain (o_i) and truth (v_i^*) are unknown. Fortunately we have their probabilistic representations, which facilitate us to compute their expected values, i.e., $\mathbb{E}[\mathbb{1}_{\{o_i=k\}}] = r_k^{t_i} \cdot 1 + (1 - r_k^{t_i}) \cdot 0 = r_k^{t_i}$, and similarly $\mathbb{E}[\mathbb{1}_{\{v_i^w = v_i^*\}}] = s_{i,v_i^w}$. Then we take the expectation of the numerator and denominator of q_k^w and derive

$$q_k^w = \big(\textstyle\sum_{t_i \in \mathcal{T}^{(w)}} r_k^{t_i} \cdot s_{i,v_i^w}\big) / \big(\textstyle\sum_{t_i \in \mathcal{T}^{(w)}} r_k^{t_i}\big). \tag{7.5}$$

Intuitively, q_k^w is computed by considering the tasks answered by worker w. To be specific, it considers (1) how much each answered task is related to domain d_k, i.e., $r_k^{t_i}$; and (2) how probable each task is answered correctly by worker w, i.e., s_{i,v_i^w}.

Running Example. We use an example to show that step 2 can satisfy the 2nd relation. Suppose a worker w_1 answers two tasks: t_1 and t_2 (ℓ_{t_1} = ℓ_{t_2} = 2), both with the first answer. Assume $s_{1,1} = 0.95$, and $s_{2,1} = 0.3$; for domain vectors, assume $r_2^{t_1} = 0.9$ and $r_2^{t_2} = 0.05$. Then we get $q_2^{w_1} = (r_2^{t_1} \cdot s_{1,1} + r_2^{t_2} \cdot s_{2,1}) / (r_2^{t_1} + r_2^{t_2})$ $= 0.92$ (Equation 7.5). Note that although w_1 answers poorly for t_2 ($s_{2,1}$ = 0.3), the worker's quality for domain d_2 is still very high ($q_2^{w_1}$ = 0.92). The reason is that t_2 is merely related to d_2 ($r_2^{t_2}$ = 0.05); moreover, w_1 answers accurately to t_1 ($s_{1,1}$ = 0.95), which is highly related to d_2 ($r_2^{t_1}$ = 0.9), making q_2^w very high.

Initialization. To initialize all workers' qualities, we can leverage each worker's answering performance for *golden tasks* (Section 7.5.2), i.e., tasks with known ground truth, which are used to test a worker's quality before a worker answers real published tasks.

Time Complexity. In step 1, let $\ell = \max_{1 \le i \le n} \ell_{t_i}$, then for each task t_i, computing each $\mathcal{M}_{k,j}^{(i)}$ takes $\mathcal{O}(\ell \cdot |V^{(i)}|)$, thus this step takes $\mathcal{O}(m\ell^2 \cdot \sum_{i=1}^{n} |V^{(i)}|)$ time in all; in step 2, to compute each q_k^w, it considers all the tasks answered by w, thus this step takes $\mathcal{O}(m \cdot \sum_{w \in \mathcal{W}} |\mathcal{T}^{(w)}|) = \mathcal{O}(m \cdot \sum_{i=1}^{n} |V^{(i)}|)$. Suppose it takes u iterations to converge, then the time complexity is $\mathcal{O}(um\ell^2 \cdot \sum_{i=1}^{n} |V^{(i)}|)$ in total. In practice, m and ℓ are constants, and $u \le 20$, thus the time complexity is linear to the number of answers.

7.4.2 Practical Truth Inference

We now study the practical issues about how to maintain workers' qualities for future use, and how to accelerate truth inference.

Maintenance of Workers' Qualities. As different requesters may publish different tasks to DOCS, the workers who have previously answered tasks may come again in the future. Thus we need to maintain workers' previous answering performance, which can be further used (e.g., initializing worker's quality) in tasks published by new requesters. Obviously, it is ineffective to store all of workers' previous completed tasks and answers. A straightforward way is to store each worker w's quality $\boldsymbol{q}^w$. However, this is insufficient, as each q_k^w ($1 \le k \le m$) is derived (Equation 7.5) based on w's answers for tasks in domain d_k, and if w answers very few tasks related to d_k, the computed q_k^w is not confident. Thus besides q_k^w, we also maintain another statistic u_k^w, i.e., the number of tasks w has answered that are related to d_k.

Specifically, in order to update a worker w's quality $\boldsymbol{q}^w$, e.g., q_k^w ($1 \le k \le m$), DOCS maintains two statistics in database: the quality q_k^w and its weight u_k^w, which is the expected number of tasks answered by w that are related to domain d_k, i.e., $u_k^w = \sum_{t_i \in \mathcal{T}^{(w)}} r_k^{t_i}$. Suppose worker w came to DOCS before and answered tasks. Let $\widehat{q}_k^w$ and $\widehat{u}_k^w$ ($1 \le k \le m$) denote two statistics stored for previous tasks, then by considering those computed for newly answered tasks (i.e., q_k^w and u_k^w for $1 \le k \le m$), Theorem 7.1 states how to update these two parameters correctly in DOCS.

Theorem 7.1 (Worker Quality Update). *If q_k^w and u_k^w are updated as $(\widehat{q}_k^w \cdot \widehat{u}_k^w + q_k^w \cdot u_k^w)/(\widehat{u}_k^w + u_k^w)$, and $(\widehat{u}_k^w + u_k^w)$, respectively, then the quality of worker w is updated correctly.*

Proof. Let $\widehat{\mathcal{T}}^{(w)}$ denote the set of tasks answered by w previously. If we consider $\mathcal{T}^{(w)}$, i.e., the set of newly published tasks answered by w. Then based on Equation 7.5, the worker w's quality q_k^w $(1 \le k \le m)$ can be updated as

$$\frac{\widehat{q}_k^w \cdot \sum_{t \in \widehat{\mathcal{T}}^{(w)}} r_k^t + \sum_{t_i \in \mathcal{T}^{(w)}} r_k^{t_i} \cdot s_{i,v_i^w}}{\sum_{t \in \widehat{\mathcal{T}}^{(w)}} r_k^t + \sum_{t_i \in \mathcal{T}^{(w)}} r_k^{t_i}}.$$

As $\sum_{t_i \in \mathcal{T}^{(w)}} r_k^{t_i} \cdot s_{i,v_i^w} = q_k^w \cdot \sum_{t_i \in \mathcal{T}^{(w)}} r_k^{t_i}$, thus by introducing another statistic $\widehat{u}_k^w = \sum_{t \in \widehat{\mathcal{T}}^{(w)}} r_k^t$ and $u_k^w = \sum_{t_i \in \mathcal{T}^{(w)}} r_k^{t_i}$, we can maintain worker's quality q_k^w and u_k^w as $(\widehat{q}_k^w \cdot \widehat{u}_k^w + q_k^w \cdot u_k^w)/(\widehat{u}_k^w + u_k^w)$, and $(\widehat{u}_k^w + u_k^w)$, respectively. □

Accelerating Truth Inference. When a worker submits answers, the **TI** module is run and the parameters are updated and stored in the database (Figure 7.1). As **TI** takes an iterative approach, it *could* be expensive to run until convergence. To alleviate this issue, we develop an incremental approach. The challenges are three-fold: (1) What are the parameters to update upon receiving an answer? (2) How can we update those parameters instantly? (3) What parameters should we store in order to facilitate such updates?

W.l.o.g., assume worker w answers a task t_i with the a-th choice. Upon receiving the answer, the basic idea is that we only update the parameters that are *most* related to task t_i and worker w, i.e., task t_i's truth and the qualities of workers who have answered task t_i. To facilitate such updates, we store the following parameters: (1) for a worker w, based on Theorem 7.1, we store its quality q_k^w and weight u_k^w $(1 \le k \le m)$; (2) for a task t_i, we store its matrix $\mathcal{M}^{(i)}$ and the probabilistic truth $\boldsymbol{s_i}$. The update policy is as follows:

• **Step 1: Inferring the Truth**. In this step, we only update task t_i's parameters, i.e., $\mathcal{M}^{(i)}$ (Equations 7.3-7.4) and $\boldsymbol{s_i} = \boldsymbol{r^{t_i}} \times \mathcal{M}^{(i)}$. In fact, the process can be further accelerated by storing another parameter $\widehat{\mathcal{M}}^{(i)}_{k,j}$, which records the numerator in Equation 7.3.

• **Step 2: Estimating Worker Quality**. In this step, we update the qualifies of worker w and those who have answered t_i before. To be specific, (1) for worker w, $q^w_k = (q^w_k \cdot u^w_k + s_{i,a} \cdot r^{t_i}_k)/(u^w_k + r^{t_i}_k)$ and $u^w_k = u^w_k + r^{t_i}_k$; (2) if worker w' ever answered task t_i with j-th answer before, then $q^{w'}_k = (q^{w'}_k \cdot u^{w'}_k - \widetilde{s}_{i,j} \cdot r^{t_i}_k + s_{i,j} \cdot r^{t_i}_k)/u^{w'}_k$, where $\widetilde{s}_{i,j}$ is the previous $s_{i,j}$ before update (i.e., step 1 above).

The complete algorithm is shown in Algorithm 16. It is not hard to prove that the above two steps are bounded in time $\mathcal{O}(m \cdot |V^{(i)}|)$, which is more efficient than the iterative approach. Note that the incremental approach may not achieve as high quality as the iterative one; however, its advantage is that the parameters can be updated instantly, which fits to the scenario when workers' answers arrive in a high velocity. In practice, we can run **TI** in a *delayed manner*, that is, the iterative approach will be run in every z submissions of answers (z = 100 in DOCS).

Next we discuss the detailed incremental algorithm. W.l.o.g., assume that worker w answers a task t_i with the a-th choice, and we represented it in a tuple $(w,\ t_i,\ a)$. To address the problem, our basic idea is that we only update the parameters that are mostly related to task t_i and worker w, i.e., task t_i's truth and the qualities of workers who have answered task t_i (including the worker w).

In order to facilitate such updates, we store the following parameters for a worker w and a task t_i, respectively:

• For a worker w, other than the worker w's quality ($\boldsymbol{q^w}$), we also store the aggregated weight of each domain for those tasks answered by w, denoted as $\boldsymbol{u^w} = [\ u^w_1, u^w_2, \ldots, u^w_m\ ]$ and each $u^w_k = \sum_{t_j \in \mathcal{T}^{(w)}} r^{t_j}_k$;

Algorithm 16 Incremental Truth Inference (Chapter 7).

```
Input: $(w, t_i, a)$; $\boldsymbol{r}^{t_i}$, $V^{(i)}$, $\boldsymbol{s_i}$, $\widehat{\mathcal{M}}^{(i)}_{k,j}$ and $\mathcal{M}^{(i)}_{k,j}$ for $1 \le k \le m, 1 \le j \le \ell_{t_i}$; $\boldsymbol{q^w}$ and $\boldsymbol{u^w}$ for $w \in \mathcal{W}$.
Output: $V^{(i)}$, $\boldsymbol{s_i}$, $\widehat{\mathcal{M}}^{(i)}_{k,j}$ and $\mathcal{M}^{(i)}_{k,j}$ for $1 \le k \le m$, $1 \le j \le \ell_{t_i}$; $\boldsymbol{q^w}$ and $\boldsymbol{u^w}$ for $w \in \mathcal{W}$.
1: $\widetilde{\boldsymbol{s}}_{\boldsymbol{i}} = \boldsymbol{s_i}$; // store the original $\boldsymbol{s_i}$
2: // Step 1: Inferring the Truth
3: for $1 \le k \le m$ do
4:   for $1 \le j \le \ell_{t_i}$ do
5:     if $j = a$ then
6:       $\widehat{\mathcal{M}}^{(i)}_{k,j} = \widehat{\mathcal{M}}^{(i)}_{k,j} \cdot q^w_k$;
7:     else
8:       $\widehat{\mathcal{M}}^{(i)}_{k,j} = \widehat{\mathcal{M}}^{(i)}_{k,j} \cdot \frac{1-q^w_k}{\ell_{t_i}-1}$;
9:     end if
10:   end for
11:   for $1 \le j \le \ell_{t_i}$ do
12:     $\mathcal{M}^{(i)}_{k,j} = \widehat{\mathcal{M}}^{(i)}_{k,j} / \sum_{j'=1}^{\ell_{t_i}} \widehat{\mathcal{M}}^{(i)}_{k,j'}$;
13:   end for
14: end for
15: $\boldsymbol{s_i} = \boldsymbol{r}^{t_i} \times \mathcal{M}^{(i)}$;
16: // Step 2: Estimating Worker Quality
17: // Update worker w's quality
18: for $1 \le k \le m$ do
19:   $q^w_k = (q^w_k \cdot u^w_k + s_{i,a} \cdot r^{t_i}_k) / (u^w_k + r^{t_i}_k)$;
20:   $u^w_k = u^w_k + r^{t_i}_k$;
21: end for
22: // Update qualities of other workers ($w'$) who have answered $t_i$ before
23: for $v^{w'}_i \in V^{(i)}$ do
24:   $j = v^{w'}_i$; // store the choice for illustration
25:   for $1 \le k \le m$ do
26:     $q^{w'}_k = (q^{w'}_k \cdot u^{w'}_k - \widetilde{s}_{i,j} \cdot r^{t_i}_k + s_{i,j} \cdot r^{t_i}_k) / u^{w'}_k$;
27:   end for
28: end for
29: $V^{(i)} = V^{(i)} \cup \{v^w_i\}$; // update the set $V^{(i)}$
30: return $V^{(i)}$, $\boldsymbol{s_i}$, $\widehat{\mathcal{M}}^{(i)}_{k,j}$ and $\mathcal{M}^{(i)}_{k,j}$ for $1 \le k \le m$, $1 \le j \le \ell_{t_i}$; $\boldsymbol{q^w}$ and $\boldsymbol{u^w}$ for $w \in \mathcal{W}$.
```

• For a task t_i, other than the matrix $\mathcal{M}^{(i)}$ and its probabilistic truth $\boldsymbol{s_i}$, we also store the numerator of the matrix $\mathcal{M}^{(i)}$, which can be observed in Equation 7.3, i.e., $\prod_{v_i^w \in V^{(i)}} \Pr(v_i^w \mid o_i = k, v_i^* = j)$. We denote it as $\widehat{\mathcal{M}}_{k,j}^{(i)}$, and from Equation 7.3 it can be inferred that $\mathcal{M}_{k,j}^{(i)} = \widehat{\mathcal{M}}_{k,j}^{(i)} / \sum_{j'=1}^{\ell_{t_i}} \widehat{\mathcal{M}}_{k,j'}^{(i)}$.

Based on the above discussions, we develop an incremental algorithm in Algorithm 16. Specifically, similar to the iterative approach (Algorithm 15), it contains two steps:

Step 1: Inferring the Truth (lines 2-15). We first update each numerator of $\mathcal{M}_{k,j}^{(i)}$, i.e., $\widehat{\mathcal{M}}_{k,j}^{(i)}$ by considering worker w's answer a to task t_i: if $j = a$, then q_k^w will be multiplied; otherwise, $\frac{1-q_k^w}{\ell_{t_i}-1}$ will be multiplied (lines 4-8). Then we update $\mathcal{M}^{(i)}$ based on the updated $\widehat{\mathcal{M}}^{(i)}$ (lines 9-10). Finally the probabilistic truth $\boldsymbol{s_i}$ will be updated based on the updated $\mathcal{M}^{(i)}$ (line 11).

Step 2: Estimating Worker Quality (lines 16-28). For this step, we first update the model for worker w (lines 13-16). Based on Equation 7.5, if we store $u_k^w = \sum_{t_j \in \mathcal{T}^{(w)}} r_k^{t_j}$, then we can derive $q_k^w \cdot u_k^w = \sum_{t_j \in \mathcal{T}^{(w)}} r_k^{t_j} \cdot s_{j,v_j^w}$. If we consider the new answer a for task t_i, then worker w's quality should be updated as

$$q_k^w \leftarrow \frac{\sum_{t_j \in \mathcal{T}^{(w)}} r_k^{t_j} \cdot s_{j,v_j^w} + s_{i,a} \cdot r_k^{t_i}}{\sum_{t_j \in \mathcal{T}^{(w)}} r_k^{t_j} + r_k^{t_i}} = \frac{q_k^w \cdot u_k^w + s_{i,a} \cdot r_k^{t_i}}{u_k^w + r_k^{t_i}}.$$

For the workers who have answered task t_i before (lines 17-20), we first store the original $\widetilde{\boldsymbol{s}}_{\boldsymbol{i}}$ (line 1), and then update the worker's quality by considering the original and updated $\widetilde{s}_{i,j}$ (lines 18-20).

After these two steps, we update $V^{(i)}$ (line 21) and return the updated parameters (line 22).

Time Complexity. For the time complexity of Algorithm 16, the first step takes $\mathcal{O}(m \cdot \ell_{t_i})$ time, and for the second step, the update of worker w's quality (lines 13-16) takes $\mathcal{O}(m)$ time, and the update of other workers who answered answered task t_i (lines 17-20) takes $\mathcal{O}(m \cdot |V^{(i)}|)$ time. So in total it takes

$\mathcal{O}(m \cdot \max\{\ell_{t_i}, |V^{(i)}|\})$ time. Considering that m and ℓ_{t_i} are often constants, so the time complexity is constrained by the number of answers already obtained for task t_i (bounded by $|\mathcal{W}|$).

7.5 Online Task Assignment

When a worker comes to crowdsourcing platforms such as AMT [1], it instantly interacts with DOCS for task assignment. Specifically, AMT will pass the unique worker ID to us, then we dynamically assign a set of k tasks (e.g., $k = 20$ in [195, 222]) to the coming worker in AMT. In this section, we address two problems in **OTA**. First, if the worker has already completed some tasks, we select k tasks and assign them to the worker (Section 7.5.1). Second, if the worker is new, we investigate how to select representative *golden tasks* to test the worker's quality (Section 7.5.2).

7.5.1 Online Task Assignment

If the coming worker w has answered tasks before, we can retrieve related parameters from database. The **Input** of the problem includes (1) worker w's quality ($\boldsymbol{q^w}$), and (2) tasks' current information (i.e., $\mathcal{M}^{(i)}$ and $\boldsymbol{s_i}$ for $1 \leq i \leq n$). The **Output** of the problem is to select k tasks for worker w, from the task set $\mathcal{T} - \mathcal{T}^{(w)}$, i.e., the set of tasks not answered by worker w before.

To assign tasks, on one hand, we assign tasks with domains that the worker is good at; on the other hand, we have to evaluate if a task is really beneficial to be assigned. For example, for a task t_i that is confident enough based on previous answers (e.g., $\boldsymbol{s_i} = [0.99, 0.01]$), then even if the coming worker is a domain expert for the task, it is of very minor benefit to assign the task.

Following the above discussions, we design an assignment framework, where for each task t_i, it estimates the *benefit* of assigning it to the coming worker, i.e., $\mathtt{B}(t_i)$; we then select k tasks that attain the highest benefits to the worker. In the following, we first focus on the problem of assigning $k = 1$ task, and then address the general problem of assigning k tasks.

Task Assignment for One Task ($k = 1$)

To address this, an important problem is how to estimate the benefit of a task, by considering *if* the task is answered by the worker? Recall that for a task t_i, we use a distribution $\boldsymbol{s_i}$ of size ℓ_{t_i} to capture its truth. Intuitively, the more concentrated $\boldsymbol{s_i}$ is (e.g., for a certain choice j ($1 \leq j \leq \ell_{t_i}$), the value $s_{i,j}$ is close to 1 while other $s_{i,j'}$ for $j' \neq j$ are close to 0), the more confident to derive the truth; otherwise, if $\boldsymbol{s_i}$ tends to be a uniform distribution, then the truth is ambiguous. By capturing this idea, we apply entropy [167] as the measure to define the ambiguity of a distribution $\boldsymbol{s_i}$, i.e., $\mathcal{H}(\boldsymbol{s_i}) = -\sum_{j=1}^{\ell_{t_i}} s_{i,j} \cdot \ln s_{i,j}$. The higher the value $\mathcal{H}(\boldsymbol{s_i})$ is, the more ambiguous $\boldsymbol{s_i}$ is. We then define the benefit function $\texttt{B}(t_i)$.

Definition 7.5 (Benefit function $\texttt{B}(\cdot)$)**.** *For a task t_i, when a worker w comes, let $\widehat{\boldsymbol{s_i}}$ denote the distribution after w answers the task, then the benefit of assigning t_i to worker w is defined as how much ambiguity can be reduced based on the assignment, i.e.,* $\texttt{B}(t_i) = \mathcal{H}(\boldsymbol{s_i}) - \mathcal{H}(\widehat{\boldsymbol{s_i}})$.

However, the computation of $\texttt{B}(t_i)$ is challenging, as $\widehat{\boldsymbol{s_i}}$ is unknown before t_i is really answered by w. Then to estimate $\widehat{\boldsymbol{s_i}}$, we have to consider the following two questions:
Q1: what answer the worker *may* give to the task, and
Q2: how the truth *will change* if the worker gives an answer.

Next we solve Q1 and Q2, respectively.

Solutions to Q1. In estimating the answer that will be given by w, we denote it as a random variable v_i^w ($1 \leq v_i^w \leq \ell_{t_i}$) and estimate it based on the collected answers, i.e., $\Pr(v_i^w = a \mid V^{(i)})$. By considering all possible true domain o_i and truth v_i^* for task t_i, we can express $\Pr(v_i^w = a \mid V^{(i)})$ as follows:

$$\sum_{k=1}^{m} \Pr(o_i = k) \sum_{j=1}^{\ell_{t_i}} \Pr(v_i^w = a \mid o_i = k, v_i^* = j) \Pr(v_i^* = j \mid o_i = k, V^{(i)}).$$

Then we can derive the following theorem that solves Q1.

Theorem 7.2. *The probability that worker w will give the a-th choice to task t_i is*

$$\Pr(v_i^w = a \mid V^{(i)}) = \sum_{k=1}^{m} r_k^{t_i} \cdot [\, q_k^w \cdot \mathcal{M}_{k,a}^{(i)} + \frac{1-q_k^w}{\ell_{t_i}-1} \cdot (1 - \mathcal{M}_{k,a}^{(i)}) \,]. \tag{7.6}$$

Proof. Note that we have shown that $\Pr(v_i^w = a \mid V^{(i)})$ can be expressed as the following equation, i.e.,

$$\sum_{k=1}^{m} \Pr(o_i = k) \sum_{j=1}^{\ell_{t_i}} \Pr(v_i^w = a \mid o_i = k, v_i^* = j) \Pr(v_i^* = j \mid o_i = k, V^{(i)}),$$

which is essentially

$$\begin{aligned}
&\textstyle\sum_{k=1}^{m} r_k^{t_i} \cdot \sum_{j=1}^{\ell_{t_i}} \mathcal{M}_{k,j}^{(i)} \cdot [\, (q_k^w)^{\mathbb{1}_{\{j=a\}}} \cdot (\frac{1-q_k^w}{\ell_{t_i}-1})^{\mathbb{1}_{\{j\neq a\}}} \,] \\
=&\textstyle\sum_{k=1}^{m} r_k^{t_i} \cdot [\, q_k^w \cdot \mathcal{M}_{k,a}^{(i)} + \sum_{j\neq a} \frac{1-q_k^w}{\ell_{t_i}-1} \cdot \mathcal{M}_{k,j}^{(i)} \,] \\
=&\textstyle\sum_{k=1}^{m} r_k^{t_i} \cdot [\, q_k^w \cdot \mathcal{M}_{k,a}^{(i)} + \frac{1-q_k^w}{\ell_{t_i}-1} \cdot (1 - \mathcal{M}_{k,a}^{(i)}) \,].
\end{aligned}$$

□

Solutions to Q2. We talk about how the truth $\boldsymbol{s_i}$ will be updated. Suppose worker w gives the a-th choice to task t_i, and let $\mathcal{M}^{(i)|a}$ denote the updated matrix of $\mathcal{M}^{(i)}$. Based on Equations 7.3 and 7.4, we can derive the formula of each element in $\mathcal{M}^{(i)|a}$ in Theorem 7.3.

Theorem 7.3. *If worker w gives the a-th choice to task t_i, then*

$$\mathcal{M}_{k,j}^{(i)|a} = \frac{\mathcal{M}_{k,j}^{(i)} \cdot (q_k^w)^{\mathbb{1}_{\{j=a\}}} \cdot (\frac{1-q_k^w}{\ell_{t_i}-1})^{\mathbb{1}_{\{j\neq a\}}}}{\sum_{j'=1}^{\ell_{t_i}} \mathcal{M}_{k,j'}^{(i)} \cdot (q_k^w)^{\mathbb{1}_{\{j'=a\}}} \cdot (\frac{1-q_k^w}{\ell_{t_i}-1})^{\mathbb{1}_{\{j'\neq a\}}}}. \tag{7.7}$$

Proof. Note that $\mathcal{M}_{k,j}^{(i)|a}$ means that if the coming worker w answers the a-th choice for task t_i, the task's original matrix item $\mathcal{M}_{k,j}^{(i)}$ will become. Let $V^{(i)}$ denote the previous answers given to t_i (without worker w's answer a). From Equation 7.3 we know that $\mathcal{M}_{k,j}^{(i)|a}$ can be updated to

$$\frac{\Pr(v_i^w = a \mid o_i = k, v_i^* = j) \prod_{v_i^{w'} \in V^{(i)}} \Pr(v_i^{w'} \mid o_i = k, v_i^* = j)}{\sum_{j'=1}^{\ell_{t_i}} \Pr(v_i^w = a \mid o_i = k, v_i^* = j') \prod_{v_i^{w'} \in V^{(i)}} \Pr(v_i^{w'} \mid o_i = k, v_i^* = j')}.$$

Also, from Equation 7.3 we know that $\mathcal{M}_{k,j}^{(i)}$ can be expressed as

$$\mathcal{M}_{k,j}^{(i)} = C \cdot \prod\nolimits_{v_i^{w'} \in V^{(i)}} \Pr(v_i^{w'} \mid o_i = k, v_i^* = j),$$

where C, or the denominator of Equation 7.3, is not related to subscript j in $\mathcal{M}_{k,j}^{(i)}$, thus for any $1 \le j' \le \ell_{t_i}$, we similarly derive

$$\mathcal{M}_{k,j'}^{(i)} = C \cdot \prod\nolimits_{v_i^{w'} \in V^{(i)}} \Pr(v_i^{w'} \mid o_i = k, v_i^* = j').$$

Considering how $\mathcal{M}_{k,j}^{(i)|a}$ can be updated above, we can derive

$$\begin{aligned}
\mathcal{M}_{k,j}^{(i)|a} &= \frac{\Pr(v_i^w = a \mid o_i = k, v_i^* = j) \cdot \mathcal{M}_{k,j}^{(i)} \cdot C^{-1}}{\sum_{j'=1}^{\ell_{t_i}} \Pr(v_i^w = a \mid o_i = k, v_i^* = j') \cdot \mathcal{M}_{k,j'}^{(i)} \cdot C^{-1}} \\
&= \frac{\mathcal{M}_{k,j}^{(i)} \cdot (q_k^w)^{\mathbb{1}_{\{j=a\}}} \cdot \left(\frac{1-q_k^w}{\ell_{t_i}-1}\right)^{\mathbb{1}_{\{j \neq a\}}}}{\sum_{j'=1}^{\ell_{t_i}} \mathcal{M}_{k,j'}^{(i)} \cdot (q_k^w)^{\mathbb{1}_{\{j'=a\}}} \cdot \left(\frac{1-q_k^w}{\ell_{t_i}-1}\right)^{\mathbb{1}_{\{j' \neq a\}}}}.
\end{aligned}$$

□

Then we can update the truth $\boldsymbol{s_i}$ as $\widehat{\boldsymbol{s_i}} = \boldsymbol{r^{t_i}} \times \mathcal{M}^{(i)|a}$, by considering that worker w gives the a-th choice to t_i.

The solutions to Q1 and Q2 tell us how to compute the probability that w gives the a-th choice to t_i, and the updated truth $\widehat{\boldsymbol{s_i}}$ if the answer is really given. Considering all possible answers, we define $\mathcal{H}(\widehat{\boldsymbol{s_i}})$ in an expected manner, i.e.,

$$\mathcal{H}(\widehat{\boldsymbol{s_i}}) = \sum\nolimits_{a=1}^{\ell_{t_i}} \mathcal{H}(\, \boldsymbol{r^{t_i}} \times \mathcal{M}^{(i)|a}\,) \cdot \Pr(v_i^w = a \mid V^{(i)}). \tag{7.8}$$

Then for each task t_i, we can compute $\mathcal{H}(\widehat{\boldsymbol{s}}_{\boldsymbol{i}})$ via Equations 7.6, 7.7, 7.8, and derive $\mathtt{B}(t_i)$ in Definition 7.5. We select the task with the highest benefit, i.e., $\mathrm{argmax}_{t_i \in \mathcal{T} - \mathcal{T}^{(w)}} \mathtt{B}(t_i)$.

Task Assignment for k Tasks

To select k tasks out of the set $\mathcal{T} - \mathcal{T}^{(w)}$, it has two challenges:

Challenge I. For a fixed set of k tasks, denoted as $\mathcal{T}_k$ ($\mathcal{T}_k \subseteq \mathcal{T} - \mathcal{T}^{(w)}$ and $|\mathcal{T}_k| = k$), we need to consider all possible answers given by worker w. W.l.o.g., we assume $\mathcal{T}_k$ contains the first k tasks in $\mathcal{T}$ (i.e., $t_i \in \mathcal{T}_k$ for $1 \leq i \leq k$). Let $\phi = [\ \phi_1, \phi_2, \ldots, \phi_k\ ]$ denote one possible combination of answers given by w for $\mathcal{T}_k$, and $1 \leq \phi_i \leq \ell_{t_i}$. Then upon receiving the answers ϕ, following Definition 7.5, the benefit of k tasks is changed to

$$\mathtt{B}_{\phi}(\mathcal{T}_k) = \textstyle\sum_{i=1}^{k} \left[\ \mathcal{H}(\boldsymbol{s_i}) - \mathcal{H}(\boldsymbol{r}^{\boldsymbol{t_i}} \times \mathcal{M}^{(i)|\phi_i})\ \right]. \tag{7.9}$$

Let all possible combinations of answers for $\mathcal{T}_k$ form a set $\Phi = \{\phi\}$ and $|\Phi| = \prod_{i=1}^{k} \ell_{t_i}$. Then if we consider all $\phi \in \Phi$, the expected benefit of assigning $\mathcal{T}_k$, denoted as $\mathtt{B}(\mathcal{T}_k)$ is expressed as

$$\mathtt{B}(\mathcal{T}_k) = \textstyle\sum_{\phi \in \Phi} \mathtt{B}_{\phi}(\mathcal{T}_k) \cdot \prod_{i=1}^{k} \Pr(v_i^w = \phi_i \mid V^{(i)}). \tag{7.10}$$

From the above analysis we know that even for a fixed k-task set $\mathcal{T}_k$, computing its benefit (Equation 7.10) needs to require exponential number of combinations in Φ (as $|\Phi| = \prod_{i=1}^{k} \ell_{t_i}$).

Challenge II. Moreover, we need to select the optimal k tasks (i.e., with the highest value $\mathtt{B}(\mathcal{T}_k)$) out of the set $\mathcal{T} - \mathcal{T}^{(w)}$. This process requires enumerating all $\binom{n}{k}$ possible cases in the worst case.

To address the first challenge, fortunately we can prove the following theorem, which reduces the complexity from exponential to linear. The basic idea is that if we consider all possible answers for one task (e.g., t_1), then it can be safely proved that $\mathtt{B}(t_1)$ can be extracted from $\mathtt{B}(\mathcal{T}_k)$, and similarly other $\mathtt{B}(t_i)$ for $t_i \in \mathcal{T}_k - \{t_1\}$ can also be extracted and added independently.

Theorem 7.4. $\mathtt{B}(\mathcal{T}_k) = \sum_{t_i \in \mathcal{T}_k} \mathtt{B}(t_i)$.

Proof. W.l.o.g., similar to the analysis in main content, we regard $\mathcal{T}_k$ as a set that contains k first tasks in $\mathcal{T}$, i.e., $\mathcal{T}_k = \{t_i \mid 1 \le i \le k\}$. Then we need to prove $\mathtt{B}(\mathcal{T}_k) = \sum_{i=1}^{k} \mathtt{B}(t_i)$. In this case, $|\Phi| = \sum_{i=1}^{k} \ell_{t_i}$, and we decompose $\mathtt{B}(\mathcal{T}_k)$ (Equation 7.10) into two parts: the first part focuses on benefit related to t_1, and the second part focuses on benefits related to other tasks (i.e., t_i for $2 \le i \le k$). For each part, we represent it as a summation over ℓ_{t_1} components, where the j-th component ($1 \le j \le \ell_{t_1}$) considers that $\phi_1 = j$, i.e.,

$$
\begin{aligned}
\mathtt{B}(\mathcal{T}_k) = & \sum_{j=1}^{\ell_{t_1}} \sum_{\phi \in \Phi \text{ s.t. } \phi_1 = j} \big(\, \mathcal{H}(\boldsymbol{s_1}) - \mathcal{H}(\boldsymbol{r}^{\boldsymbol{t_1}} \times \mathcal{M}^{(1)|j}) \,\big) \cdot \\
& \qquad\qquad \Pr(v_1^w = j \mid V^{(1)}) \cdot \prod_{i=2}^{k} \Pr(v_i^w = \phi_i \mid V^{(i)}) \\
& + \sum_{j=1}^{\ell_{t_1}} \sum_{\phi \in \Phi \text{ s.t. } \phi_1 = j} \Big[\, \sum_{i=2}^{k} \big(\mathcal{H}(\boldsymbol{s_i}) - \mathcal{H}(\boldsymbol{r}^{\boldsymbol{t_i}} \times \mathcal{M}^{(i)|\phi_i})\big) \,\Big] \cdot \\
& \qquad\qquad \Pr(v_1^w = j \mid V^{(1)}) \cdot \prod_{i=2}^{k} \Pr(v_i^w = \phi_i \mid V^{(i)}) \\
= & \sum_{j=1}^{\ell_{t_1}} \big(\, \mathcal{H}(\boldsymbol{s_1}) - \mathcal{H}(\boldsymbol{r}^{\boldsymbol{t_1}} \times \mathcal{M}^{(1)|j}) \,\big) \cdot \Pr(v_1^w = j \mid V^{(1)}) \\
& + \sum_{\phi \in \Phi \text{ s.t. } \phi_1 = j} \Big[\, \sum_{i=2}^{k} \big(\mathcal{H}(\boldsymbol{s_i}) - \mathcal{H}(\boldsymbol{r}^{\boldsymbol{t_i}} \times \mathcal{M}^{(i)|\phi_i})\big) \,\Big] \cdot \prod_{i=2}^{k} \Pr(v_i^w = \phi_i \mid V^{(i)}) \\
= & \ \mathtt{B}(t_1) \\
& + \sum_{\phi \in \Phi'} \Big[\, \sum_{i=2}^{k} \big(\mathcal{H}(\boldsymbol{s_i}) - \mathcal{H}(\boldsymbol{r}^{\boldsymbol{t_i}} \times \mathcal{M}^{(i)|\phi_i})\big) \,\Big] \cdot \prod_{i=2}^{k} \Pr(v_i^w = \phi_i \mid V^{(i)}),
\end{aligned}
$$

where Φ' contains all possible answers for the first k tasks other than t_1, i.e., $|\Phi'| = \prod_{i=2}^{k} \ell_{t_i}$, and we denote each $\phi \in \Phi'$ as $\phi = [\, \phi_2, \phi_3, \ldots, \phi_k \,]$, where each $1 \le \phi_i \le \ell_{t_i}$ ($2 \le i \le k$).

Then we know that $\texttt{B}(\mathcal{T}_k) = \texttt{B}(t_1) + \texttt{B}(\mathcal{T}_{k-1})$, where $\mathcal{T}_{k-1} = \{t_i \mid 2 \leq i \leq k\}$. Based on mathematical induction, we can derive $\texttt{B}(\mathcal{T}_k) = \sum_{i=1}^{k} \texttt{B}(t_i)$. Thus we have proved that if $\mathcal{T}_k$ contains the first k tasks in $\mathcal{T}$, we have $\texttt{B}(\mathcal{T}_k) = \sum_{i=1}^{k} \texttt{B}(t_i)$. It can be generalized to any $\mathcal{T}_k$, which is our theorem, i.e., $\texttt{B}(\mathcal{T}_k) = \sum_{t_i \in \mathcal{T}_k} \texttt{B}(t_i)$. □

Theorem 7.4 implies that to compute the benefit for k tasks, we can compute each $\texttt{B}(t_i)$ (Definition 7.5 and Equations 7.6, 7.7, 7.8) and add up individual benefit. Then in order to address the second challenge, i.e., to select the k tasks with highest $\texttt{B}(\mathcal{T}_k)$, we only need to select top k tasks with the highest values of $\texttt{B}(t_i)$, from the set $\mathcal{T} - \mathcal{T}^{(w)}$.

Time Complexity. To compute the benefit for each task (Equation 7.8), it should compute Equations 7.6, 7.7, which take $\mathcal{O}(m\ell^2)$ in all, where $\ell = \max_{1 \leq i \leq n} \ell_{t_i}$. Then computing benefits for all tasks take $\mathcal{O}(nm\ell^2)$. As selecting top k values in a size n list can be addressed linearly (e.g., PICK algorithm in [26]), the time complexity for task assignment is $\mathcal{O}(nm\ell^2)$. Considering that m and ℓ are often constants, the complexity is linear to the number of tasks.

7.5.2 Selecting Golden Tasks

For a new worker, to test the worker's quality for different domains, we select some tasks with ground truth, called *golden tasks*. The golden tasks are selected after **DVE** is done. Then for each new worker from AMT [1], we assign the same golden tasks to her, and initialize her quality by comparing her answers with the ground truth for these tasks. However, we can manually label the ground truth (or refer to experts) for only a limited number of tasks. Thus given n tasks and a number $n' \ll n$, the problem is how to select the most representative n' tasks (out of n tasks) as *golden tasks*?

In order to profile each worker in a fine granularity w.r.t. n tasks, we consider two intuitive guidelines. (1) Each selected golden task should accurately capture a certain domain. For example, for the k-th domain, the selected task t_i should guarantee that its domain vector $\boldsymbol{r}^{t_i}$ has a high value of $r_k^{t_i}$. (2) The distribution of selected golden tasks w.r.t. domains should approximate the distribution of all tasks' aggregated domain vectors. Let n'_k denote the number of tasks selected for the k-th domain, and $\sum_{k=1}^{m} n'_k = n'$. Then the distribution $\boldsymbol{\sigma} = [\frac{n'_1}{n'}, \frac{n'_2}{n'}, \ldots, \frac{n'_m}{n'}]$ should approximate the distribution $\boldsymbol{\tau} = [\frac{\sum_{i=1}^{n} r_1^{t_i}}{n}, \frac{\sum_{i=1}^{n} r_2^{t_i}}{n}, \ldots, \frac{\sum_{i=1}^{n} r_m^{t_i}}{n}]$.

Suppose we know n'_k for $1 \leq k \leq m$, then following the first guideline, for each domain d_k, we can select top n'_k tasks with the highest values of $r_k^{t_i}$, i.e., highly related to d_k. Then the remaining problem is how to decide each n'_k for $1 \leq k \leq m$?

Following guideline 2, to define the similarity of two distributions $\boldsymbol{\sigma}$ and $\boldsymbol{\tau}$, a widely-used metric is *KL-divergence* [108]: $D(\boldsymbol{\sigma}, \boldsymbol{\tau}) = \sum_i \sigma_i \cdot \ln(\sigma_i / \tau_i)$. It can be proved in [108] that $D(\cdot, \cdot) \geq 0$ and the lower the value is, the similar the two distributions are. Thus we aim to minimize $D(\boldsymbol{\sigma}, \boldsymbol{\tau})$ w.r.t. constraints as follows:

$$\begin{aligned} \min_{\{n'_k\}} \ & \sum\nolimits_{k=1}^{m} \frac{n'_k}{n'} \cdot \ln \frac{n'_k \cdot n}{n' \cdot \sum_{i=1}^{n} r_k^{t_i}} \\ \text{s.t.} \ & \sum\nolimits_{k=1}^{m} n'_k = n' \ \text{ and } \ n'_k \in \mathbb{N} \ \text{ for } \ 1 \leq k \leq m. \end{aligned} \tag{7.11}$$

It is not hard to prove that solving Equation 7.11 is NP-hard, due to the fact that it is in general an "Integer Programming Problem" [143]. Despite its hardness, we devise an approximation algorithm in Algorithm 17.

The general idea is to let each $n'_k / n' \approx \sum_{i=1}^{n} r_k^{t_i} / n$ for $1 \leq k \leq m$ w.r.t. constraints in Equation 7.11. To do this, we first derive a lower-bound for each n'_k and set $n'_k = \lfloor \sum_{i=1}^{n} r_k^{t_i} / n \cdot n' \rfloor$. Then a procedure is run for $n' - \sum_{k=1}^{m} n'_k$ times, and each time it conducts $n'_{ind} = n'_{ind} + 1$, where ind $(1 \leq ind \leq m)$ is the choice of the domain with the minimum objective value if selected, i.e.,
$ind = \min_k \{\frac{n'_k+1}{n'} \cdot \ln \frac{(n'_k+1) \cdot n}{n' \cdot \sum_{i=1}^{n} r_k^{t_i}} + \sum\limits_{j \neq k} \frac{n'_j}{n'} \cdot \ln \frac{n'_j \cdot n}{n' \cdot \sum_{i=1}^{n} r_j^{t_i}}\}$. Finally for each domain

Algorithm 17 Golden Tasks Selection (Chapter 7).

```
Input: $n'$ (#golden tasks to be selected), tasks $t_i$ ($1 \le i \le n$)
Output: $G$
1: for $1 \le k \le m$ do
2:     $n'_k = \lfloor \sum_{i=1}^{n} r_k^{t_i} / n \cdot n' \rfloor$;
3: end for
4: while $n' - \sum_{k=1}^{m} n'_k > 0$ do
5:     $ind = \min_k \{ \frac{n'_k+1}{n'} \cdot \ln \frac{(n'_k+1)\cdot n}{n' \cdot \sum_{i=1}^{n} r_k^{t_i}} + \sum_{j \ne k} \frac{n'_j}{n'} \cdot \ln \frac{n'_j \cdot n}{n' \cdot \sum_{i=1}^{n} r_j^{t_i}} \}$;
6:     $n'_{ind} = n'_{ind} + 1$;
7: end while
8: $G = \emptyset$;
9: for $1 \le k \le m$ do
10:     $G = G \cup \{$a set of $n'_k$ tasks with the highest values of $r_k^{t_i}\}$;
11: end for
12: return $G$;
```

d_k, we select top n'_k tasks with the highest values of $r_k^{t_i}$, and then obtain all our selected n' golden tasks.

Time Complexity. To solve Equation 7.11, first computing the lower-bounds take $\mathcal{O}(m)$ time. For a real number x, we know $x \le \lfloor x \rfloor + 1$, then $n' = \sum_{k=1}^{m} \frac{\sum_{i=1}^{n} r_k^{t_i}}{n} \cdot n' \le \sum_{k=1}^{m} \lfloor \frac{\sum_{i=1}^{n} r_k^{t_i}}{n} \cdot n' \rfloor + m$,

thus the procedure is run at most m times, which takes $\mathcal{O}(m^2 \cdot n)$ time. After solving Equation 7.11, selecting top n'_k tasks for different domains takes $\mathcal{O}(m \cdot n)$ time. So in total it takes $\mathcal{O}(m^2 \cdot n)$ time.

7.6 Experiments

We evaluate DOCS on both real and simulated datasets. The settings are introduced in Section 7.6.1. Unless stated otherwise, real datasets are used to evaluate both the effectiveness and efficiency of the three modules: **DVE** (Section 7.6.2), **TI** (Section 7.6.3), and **OTA** (Section 7.6.4). We implement DOCS in Python 2.7 with the Django web framework on a 16GB memory Ubuntu server.

7.6.1 Settings

Real-World Datasets. We conduct experiments on AMT [1] with four real-world datasets.

ItemCompare Dataset (Item) [63]. It contains 360 tasks with 4 domains: *NBA*, *Food*, *Auto* and *Country*, where each domain has 90 tasks. For each task, it asks workers to compare between two items. The task descriptions in each domain are highly similar, for example, in domain *Food*, each task compares which food (e.g., 'Chocolate' and 'Honey') contains more calories.

4-Domain Dataset (4D). It contains 400 tasks, which cover 4 domains: *NBA*, *Car*, *Film* and *Mountain*, where each domain has 100 tasks. Different from dataset Item, in 4D the task descriptions in each domain are not that similar, e.g., in domain *NBA*, the tasks vary in different forms: we ask the position of a player; compare the height (or age) of two players; compare which team wins more championships, etc. We manually label the ground truth.

Yahoo QA Dataset (QA). It [158] includes queries to a search engine in 2012-2014, and each query has a *best answer* in Yahoo Answers [204]. We select 1000 queries from the dataset, where for each query, we generate tasks related to the *best answer*. For example, "*Where does chili originate from, Texas or Turkey?*", where the *best answer* ('*Texas*') and the correct domain ('*Food&Drink*') can be extracted from the corresponding webpage [6].

SFV Dataset (SFV). It [131] contains 328 tasks, where each task asks the attribute of a person (e.g., the age of *Bill Gates*), and each task also shows a set of choices collected from different QA systems [97,223], from which the workers select one as the correct choice. The ground truth is provided by [131].

Answer Collection. We publish tasks for the four datasets on AMT [1], which interacts with workers using Human Intelligence Task (HIT). When a worker comes, we batch $k = 20$ tasks in a HIT (same as [195, 222]) to the worker, and pay \$0.1 for the worker upon finishing the HIT. We assign each task to 10 different workers, so each dataset costs $\frac{360\times10}{20}\times\$0.1$ = \$18, \$20, \$50 and \$16.4, respectively. We select 20 *golden tasks* (Section 7.5.2) for each dataset.

(1) In **TI** (Section 7.6.3), to make a fair comparison, we collect workers' answers as above and compare our solution (Section 7.4.1) with the existing methods on the *same* collected answers for each dataset.

(2) In **OTA** (Section 7.6.4), as the assigned tasks for each coming worker may be totally different for different methods, to ensure that the same set of workers are used in comparisons, similar to [222], we assign tasks to a coming worker *in parallel* using different assignment methods. To be specific, there are 6 methods (see below) in comparison for task assignment, and when a worker comes, we use each method to assign 3 tasks, so $3 \times 6 = 18$ tasks are batched in a HIT (in random order) and assigned to the coming worker. We ensure that each method collects the same number of answers (e.g., 360×10 for dataset Item) in total. Then we compare with them on *respective* collected answers for each dataset.

Comparisons. We compare DOCS with existing methods: iCrowd (IC) [63], FaitCrowd (FC) [131], Majority Vote (MV), ZenCrowd (ZC) [48], David&Skene (D&S) [47], AskIt! [27], QASCA [222]. As different methods focus on different perspectives, we compare with different methods in different modules.

(1) In **DVE** (Section 7.6.2), we compare with IC [63] and FC [131], which try to exploit the domain(s) of each task.

(2) In **TI** (Section 7.6.3), we compare with MV, ZC [48], D&S [47], IC [63] and FC [131], which study truth inference.

(3) In **OTA** (Section 7.6.4), we perform end-to-end comparisons with methods AskIt! [27], IC [63], QASCA [222] and two baseline methods (Baseline and D-Max) that study online task assignment.

Evaluation Metric. For effectiveness, we use *Accuracy* to evaluate the quality of a method, and it measures the percentage of tasks whose truth are inferred correctly by the method. For efficiency, we measure the *Execution Time* of a method.

7.6.2 Evaluating Domain Vector Estimation

Evaluating the Accuracy of Domain Detection. We compare with two methods IC [63] and FC [131], which exploit diverse domains in a task. To be specific, IC uses Latent Dirichlet Analysis (LDA [25]) to model diverse domains in a task and compute the cosine similarity between pairwise tasks. It first manually sets the number of latent domains (m'), and then uses a generative model to capture how each task's text can be generated. Finally it learns a distribution (size m' vector) for each task, which indicates how it is related to each domain. Similarly, FC uses TwitterLDA [216], which is an adaptation of LDA [25] and focuses more on short texts (e.g., Tweets). It also sets the number of latent domains (m'') and then learns a specific domain for each task. To summarize, the two models used by IC and FC (1) only consider the text in each task, and (2) manually set the number of latent domains and learn a domain vector for each task w.r.t. latent domains. Different from them, DOCS (1) considers knowledge base (i.e., Freebase [71] with rich contextual and semantic information), and (2) learns a domain vector that captures the *explicit* domains (i.e., 26 domains in Yahoo Answers [204]), rather than the *latent* domains for each task. Next, we compare DOCS with IC and FC on four datasets.

• Datasets Item & 4D. For dataset 4D, there are 4 domains, and we manually set the latent m' and m'' as 4 to favor them. DOCS uses the default 26 explicit domains. After computing the domain vector for each task, we regard the domain with the highest probability as the detected domain for each task. Then we manually check for IC and FC on how each latent domain can be mapped to the 4 domains *NBA*, *Car*, *Film* and *Mountain*: for the domain vectors in tasks with true domain (say *NBA*), if we find that the probabilities in a latent dimension is very high, then we map that latent dimension to *NBA*. In DOCS, we can verify that the 4 domains are mapped to *Sports*, *Cars*, *Entertain* and *Science* in Yahoo

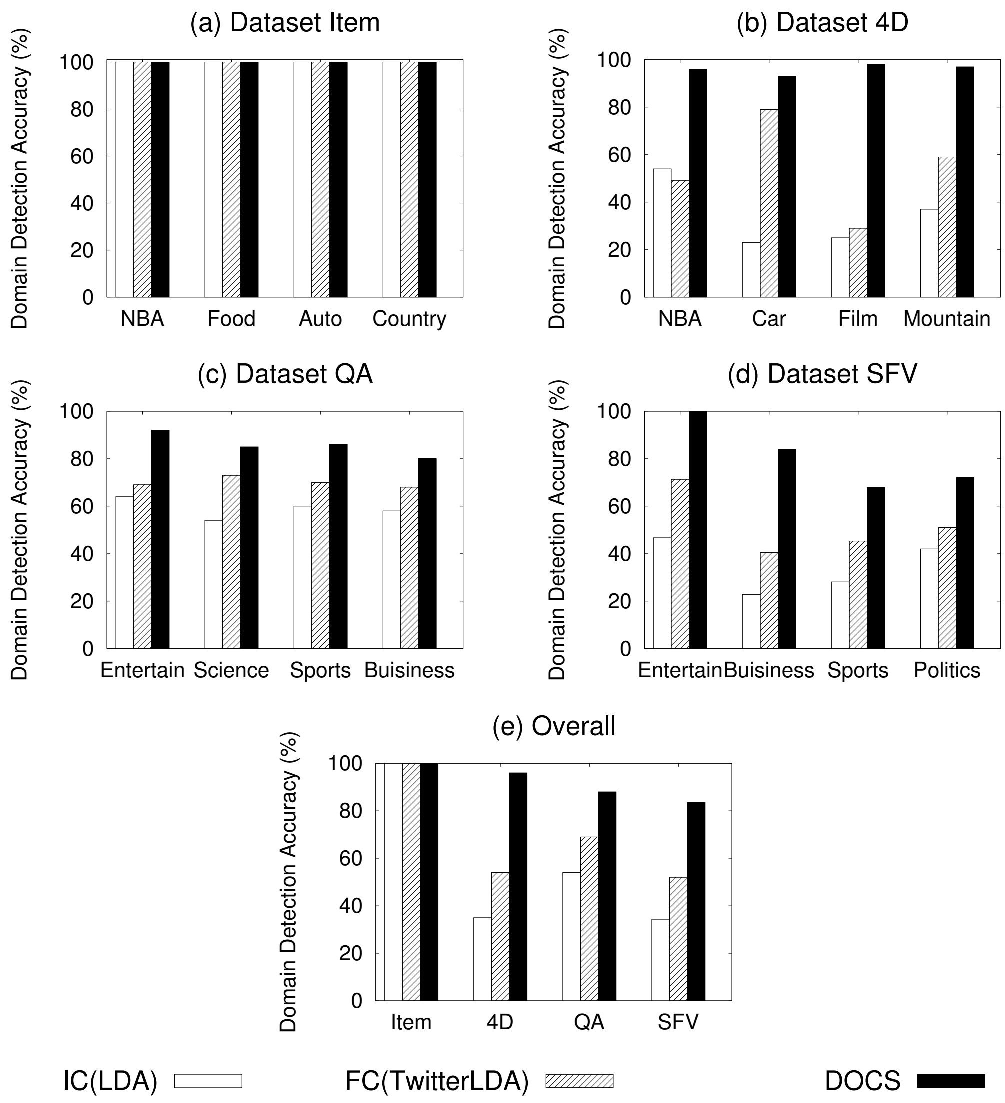


Figure 7.3: The Domain Detection Accuracy of Different Methods.

Answers [204], respectively. Finally, we compute the domain detection accuracy per domain (the percentage of tasks that are detected correctly in the domain). Figures 7.3(a)(b) show the domain detection accuracy for each domain, by comparing with different methods. We also record their overall domain detection accuracy in Figure 7.3(e). It can be observed that in Item, the accuracy is very high ($\sim$100%) in all domains for three methods; however, in dataset 4D, our method DOCS performs much better compared with IC and FC, and the reason is that they use topic model-based methods (LDA [25] and TwitterLDA [216]), which perform well if the tasks in each domain have high string similarities (e.g., in Item, tasks in each domain compare two given items on the same metric); however, in 4D, as task descriptions vary in each domain, they cannot detect the correct domain. For example, both of the two methods detect the tasks that compare the heights between two basketball players and two mountains in the same domain, mainly because they have high string similarities. However, DOCS can detect the difference between them based on the *semantic* information of players and mountains, yielding the overall detection accuracy above 95%.

• Datasets QA & SFV. For dataset QA, although there are 26 domains in Yahoo Answers, most of the queries are related to *Entertain*, *Science*, *Sports* and *Business* (as most collected search engine queries [158] are related to them), so we only focus on tasks in those four domains and set the latent m' and m'' as 4. For dataset SFV, since each task asks a specific attribute for a person (e.g., age of *Bill Gates*), we manually label the ground truth of the person as his/her most renowned domain (e.g., *Business*). As most tasks are related to domains *Entertain*, *Business*, *Sports* and *Politics*, we focus on those tasks and set $m' = m'' = 4$. Similarly we compute the domain detection accuracy for the three methods as above in Figures 7.3(c)(d) and record the overall domain detection accuracy in Figure 7.3(e). It can be seen that in real-world question answering tasks, as tasks in each domain are not that similar in strings syntactically, IC and FC perform very bad (FC performs better than IC as it favors more on short texts); however, DOCS can capture the semantics in each domain and derive the correct domain accurately. As a result, it achieves over 20% improvement in overall accuracy.

Analysis on Multiple Domains. Note that for each task, the ground truth is only one domain. However, in real-world datasets (e.g., QA), each task can be related to multiple domains. Based on the computed domain vectors, we pick out those tasks whose domain vectors have more than one mode (or peak). Among them, we find some interesting cases. For example, in the task *"Is there*

Table 7.4: The Efficiency of Heuristics on Domain Vector Estimation.

Dataset	**Top-20 (Default)**		**Top-10**		**Top-3**	
	Alg. 14	Enum.	Alg. 14	Enum.	Alg. 14	Enum.
Item	27.3s	>1 day	7.5s	>1 day	0.6s	1.3s
4D	28.6s	>1 day	8.8s	>1 day	0.8s	1.4s
QA	58.1s	>1 day	23.8s	>1 day	2.6s	264.7s
SFV	46.3s	>1 day	17.7s	>1 day	1.9s	406.8s

a name for the whistle song that the Harlem Globetrotters are known for?", it is both related to domains '*Entertain*' and '*Sports*', and our computed domain vector has both high probabilities on those two domains. Similarly, we find that the task *"Who owns the Atalanta calcio (soccer/football) team in italy?"* is related to both '*Business*' and '*Sports*'; while the task *"What is the name of Simpson's episod, where Russia becomes Soviet Union?"* is related to both '*Entertain*' and '*Politics*'. Note that our framework also considers the relatedness of each domain to each task. In the future, it might be interesting to develop metrics on evaluating how a method can compute a task's multiple domains correctly.

Evaluating the Efficiency of DVE. We next compare the efficiency of *Enumeration* ($\mathcal{O}(c^{|E_t|} \cdot |E_t| \cdot m)$) and Algorithm 14 ($\mathcal{O}(c \cdot m^2 \cdot |E_t|^3)$) on different heuristics. In DOCS, we set $m = 26$, and E_t is the set of detected entities by Wikifier [159], which extracts top $c = 20$ related concepts for each entity. The efficiency of different methods on different datasets is shown in Table 7.4. It can be seen that on all four datasets, Algorithm 14 completes within one minute; on the other hand, *Enumeration* needs more than 1 day to finish. We have also compared the efficiency with two heuristics, which remove low-probability concepts, and only extract top $c = 10$ and $c = 3$ concepts for each entity. It can seen in Table 7.4 that although *Enumeration* performs fine on small datasets with few entities (e.g., 4D, Item), it is outperformed significantly (more than 100 times) by Algorithm 14 on QA and SFV. The reason is that *Enumeration* takes more time in QA-based tasks with a large number of entities.

7.6.3 Evaluating Truth Inference

In this section we evaluate **TI** in DOCS. We first exploit different aspects of our **TI**, and then compare our **TI** with other competitors. Finally we perform a case study on a dataset (Item).

Convergence. We run the iterative **TI** on the collected answers and identify the change of parameters between adjacent iterations. Let $\bar{s}_{i,j}$ and $\bar{q}_k^w$ denote the probabilistic truth and worker quality in the last iteration, then the change of parameters Δ between the adjacent two iterations is defined as $\Delta = \sum_{i=1}^{n} \sum_{j=1}^{\ell_{t_i}} \frac{|s_{i,j} - \bar{s}_{i,j}|}{n \cdot \ell_{t_i}} + \sum_{w \in \mathcal{W}} \sum_{k=1}^{m} \frac{|q_k^w - \bar{q}_k^w|}{|\mathcal{W}| \cdot m}$. We vary the number of iterations and record Δ in Figure 7.4(a). It can be seen that Δ drops significantly in the first 10 iterations and remains steady (convergence) ever since. In practice, we can terminate **TI** within a few (say 20) iterations.

Varying #Golden Tasks. As we initialize each worker's quality using golden tasks, in Figure 7.4(b) we vary the number of golden tasks in $[0, 40]$ and observe the change of accuracy on different datasets. It can be seen that the quality is significantly improved with a small number of golden tasks, as the iterative approach requires a good initialization. However, when the golden tasks are aplenty (say 20), the accuracy remains steady. For practical usage, we set the number of golden tasks as 20, which works well in experiments.

Varying #Collected Answers. As we collect each dataset by assigning each task to exactly 10 workers, in Figure 7.4(c), we vary the number of collected answers in $[1, 10]$ for each task and observe the accuracy of **TI** on different datasets. It can be seen that the accuracy becomes better as more answers are collected. However, for some dataset such as Item, it remains stable as ≥ 8 answers are collected. We will study the estimation of stable point in future.

Worker Quality Estimation. We next examine, when the worker answers more tasks, whether a worker's quality is closer to the worker's true quality. We first compute each worker w's true quality $\widetilde{q}^w$ by comparing the worker's answers

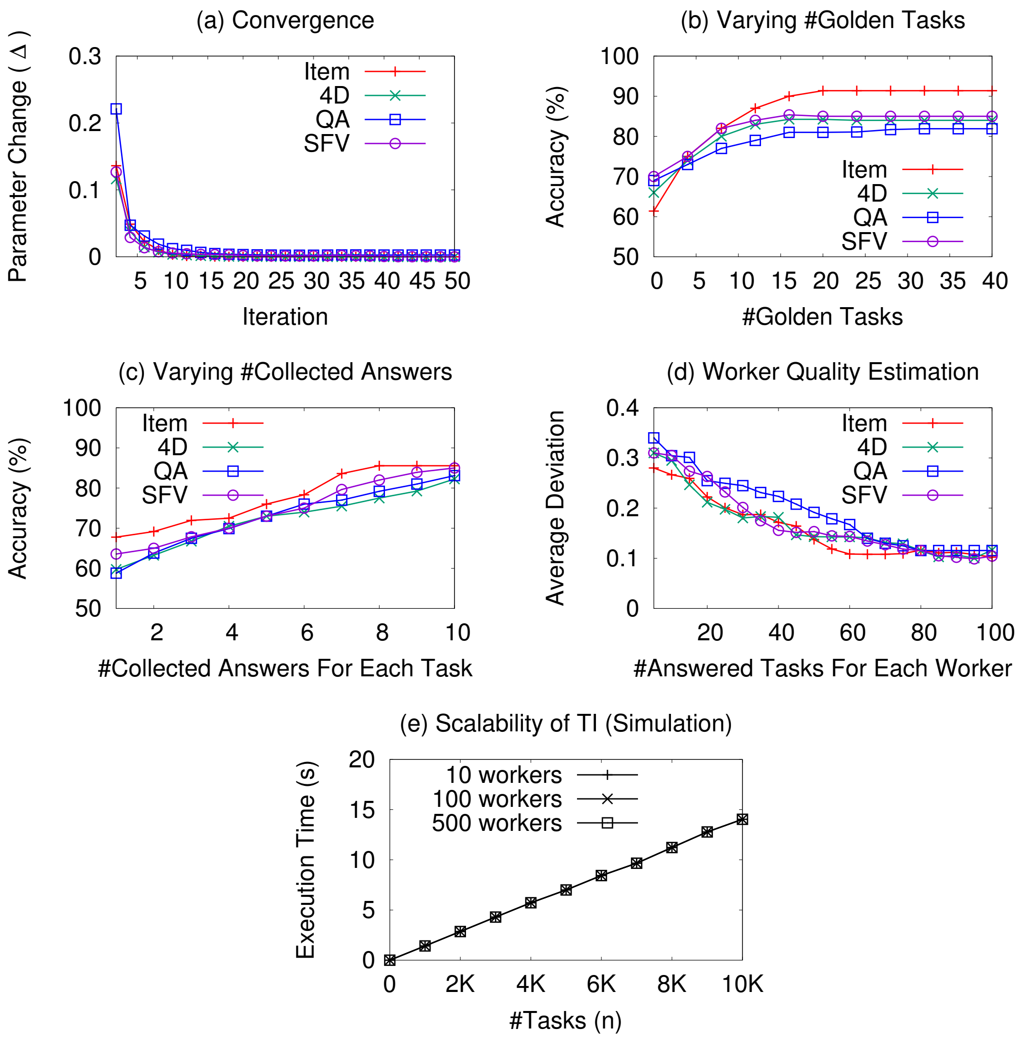


Figure 7.4: Exploiting Different Aspects of Truth Inference in DOCS.

with the ground truth. Here, $\widetilde{q}^w_k$ $(1 \leq k \leq m)$ is the fraction of the number of correctly answered tasks by worker w, among all her answered tasks in domain d_k. Then, we vary the number of collected answers (in $[1, 50]$) for each worker, and run **TI** to compute each worker w's quality $\boldsymbol{q^w}$. Finally, we record the average deviation, defined as $\sum_{w \in \mathcal{W}} \sum_{k=1}^{m} \frac{|\widetilde{q}^w_k - q^w_k|}{m \cdot |\mathcal{W}|}$, in Figure 7.4(d). As workers answer more tasks, the average deviation decreases. Also, when 80 or more tasks are answered, the deviation becomes consistently low.

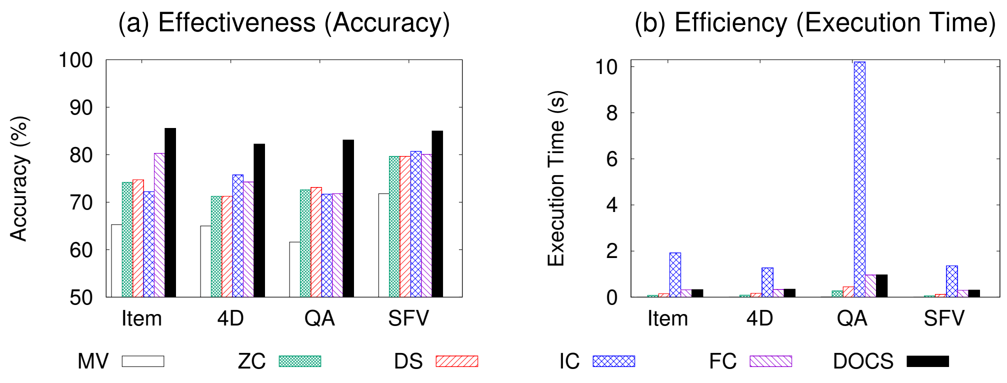


Figure 7.5: Truth Inference Comparisons.

Scalability of TI (Simulation). We create n tasks with $m = 20$. Then the worker set $\mathcal{W}$ is generated, and each task is assigned to 10 randomly selected workers from $\mathcal{W}$. We vary $n \in [0, 10\text{K}]$, $|\mathcal{W}| \in \{10, 100, 500\}$ and run **TI** on randomly generated workers' answers. Figure 7.4(e) records the time. It can be seen that the time linearly increases with n, and the worker size is invariant with time, which corresponds to the complexity $\mathcal{O}(cm\ell^2 \cdot \sum_{i=1}^{n} |V^{(i)}|)$. Given that truth inference can be done offline, it is efficient, as it takes less than 15s with large data ($n = 10\text{K}$ and $|\mathcal{W}| = 500$).

Comparing DOCS with Competitors on TI. We compare with other five competitors (i.e., MV, ZC, D&S, IC and FC) on truth inference: (1) MV regards the truth of a task as the answer given by the highest number of workers; (2) ZC [48] models each worker's quality as a value, and adapts EM framework [49] to iteratively compute worker's quality and truth; (3) D&S [47] models each worker's quality as a matrix, and also adapts EM framework [49] to iteratively compute worker's quality and truth; (4) IC [63] models a worker's quality for each task, and derives the truth using weighted majority voting; (5) FC [131] models a worker's quality as a vector of size m'', indicating the worker's quality for different *latent* domains, and it iteratively drives truth and worker's quality.

To make a fair comparison, we initialize the workers' qualities of all other competitors using the *same golden tasks*. Note that as IC and FC exploit the domains of each task, and the domain detection accuracy is not satisfactory (Figure 7.3), to do a more challenging job, we initially assign the *ground truth* of each task's domain to IC and FC, and then compute the truth of each task by them. We show the comparison results on both the effectiveness (*Accuracy*) and efficiency (*Execution Time*) of all datasets in In Figures 7.5(a)(b). We have the following observations: (1) MV is surpassed by other competitors, as it does not model a worker's quality and regard each worker as equal. (2) ZC and D&S model a worker as a value or matrix, which does not consider a worker's quality for different domains, and that is why they are outperformed by more advanced methods. (3) IC, FC and DOCS model a worker's qualities for different domains, and DOCS outperforms other competitors on all datasets. Note that although IC and FC have been assigned with the ground truth of each task's domain in truth inference, we still outperform them a lot because our designed approach can capture the inherent relations between workers' qualities and tasks' truth accurately; while for FC, the modeled relations cannot capture the inherent relations very well, and for IC, it uses weighted majority voting, whose result is easy to be misled by low-quality workers. (4) For efficiency (Figure 7.5(b)), MV is the fastest as it can directly compute the truth of each task, while others adopt an iterative approach. IC is the least efficient as it first takes a preliminary offline computation, which will then facilitate online inference. As truth inference can be done offline, all methods can work well in practice. Dataset QA costs more time as it is larger than others.

We perform a case study on Item to show workers' qualities.

Worker's True Quality Across Domains. Similar to *Worker Quality Estimation*, we first compute each worker w's *true quality* $\widetilde{\boldsymbol{q}}^{\boldsymbol{w}}$ for four domains (q_k^w). In each domain ($1 \leq k \leq 4$), we split each worker w's quality q_k^w into 10 bins: it falls into the i-th bin ($1 \leq i \leq 9$) if $q_k^w \in [\frac{i-1}{10}, \frac{i}{10})$, and in the 10-th bin if $q_k^w \in [0.9, 1.0]$. Then for each domain, we record the number of workers that fall into each bin in Figure 7.6(a). It can be seen that most workers are good at answering tasks

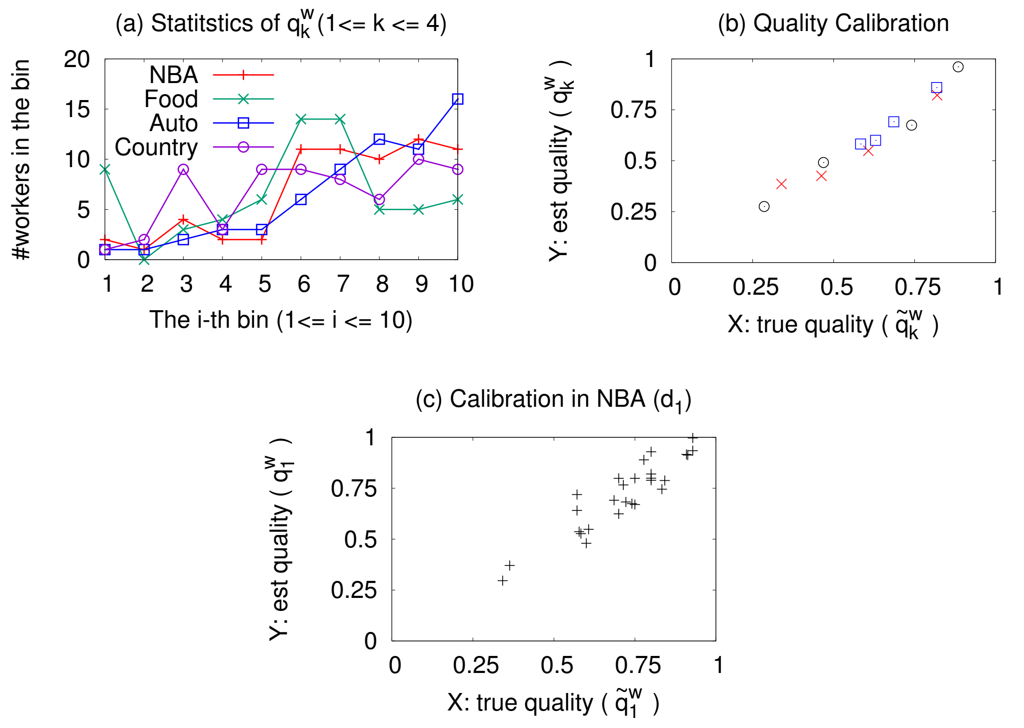


Figure 7.6: Case Studies of Worker Qualities on Dataset Item.

related to *Auto* (as there are ≥15 workers with quality $\geq$ 0.9); while workers have relatively low qualities on answering tasks related to *Food*. This means that it is necessary to select domain experts in all workers.

Worker Quality Calibration. Similar to the above, we first compute each worker w's *true quality* $\tilde{q}^w$. Then we study whether the estimated quality q_k^w by DOCS is close to the true quality $\tilde{q}_k^w$. In Figure 7.6(b), we select 3 workers who have answered the highest number of tasks and study their qualities. Specifically, the three workers are identified by labels '$\times$', '$\boxdot$', '$\odot$'; each worker w corresponds to 4 points, where each point $(\tilde{q}_k^w, q_k^w)$ corresponds to a domain d_k $(1 \leq k \leq 4)$. In the ideal case, all points should lie on the line $Y = X$, which means that the quality is estimated the same as the true quality. We observe that (1) a worker has diverse qualities in different domains. For example, the worker with label '$\odot$' has high qualities on two domains, and low qualities on

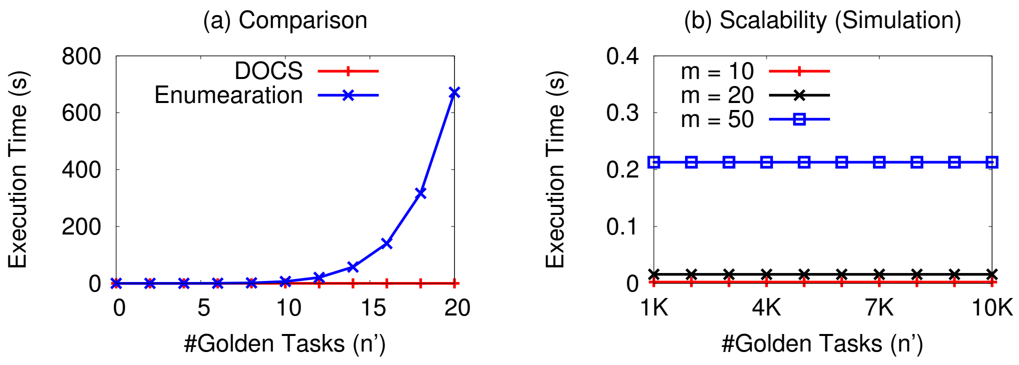


Figure 7.7: Golden Tasks Selection (Simulation).

other domains. (2) We can estimate a worker's quality accurately, as the points drawn in the figure lie very close to the line $Y = X$. For the domain *NBA* (d_1), we further plot the points for all workers who have performed more than one HIT (i.e.,$>$ 20 tasks) in Figure 7.6(c). We can observe that in general, q_1^w is close to $\widetilde{q}_1^w$.

7.6.4 Evaluating Online Task Assignment

We first evaluate golden tasks selection, and then compare with other competitors in task assignment on respective collected datasets.

Evaluating Golden Tasks Selection (Simulation). The key of selecting golden tasks (Section 7.5.2) is to solve Equation 7.11, which enumerates all possible vectors $[n'_1, n'_2, \ldots, n'_m]$ such that $\sum_{k=1}^{m} n'_k = n'$ and $n'_k \in \mathbb{N}$ $(1 \leq k \leq m)$. This is called *"Compositions of n' with size m (0 is allowed)"* [41], and it consists of $\binom{n'+m-1}{m-1}$ possible cases. By enumerating all cases, we can derive the optimal vector which obtains the minimum *KL-divergence*, i.e., D_{opt}. We can also compute D based on our solution (Section 7.5.2). We set m = 10, vary $n' \in [0, 20]$, and for each n', a distribution of size m, i.e., $\boldsymbol{\tau}$ is randomly generated. Then we record the time of both methods in Figure 7.7(a). It can be seen that the time for Enumeration increases exponentially with n', and when n' = 20, it takes more than 600s; while DOCS is efficient, which takes $<$0.01s. We also compute the ap-

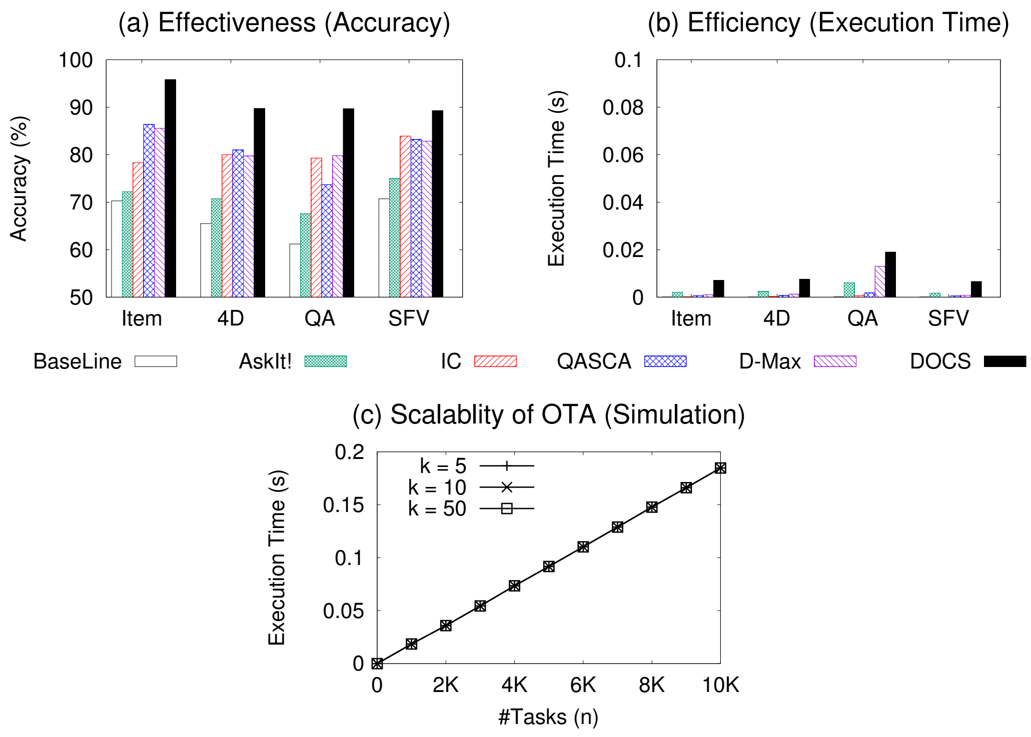


Figure 7.8: Online Task Assignment Comparisons.

proximation ratio, defined as $\gamma = |D - D_{opt}|/D_{opt}$ over all experiments, and the average γ is within 0.1%, which means that our computed results are very close to optimum. Next, we evaluate the scalability of our solution in Figure 7.7(b). We vary $n' \in [1\text{K}, 10\text{K}]$, $m \in \{10, 20, 50\}$ and observe the execution time. It can be seen that for a given m, the time is invariant with n', because the method that solves Equation 7.11 takes $\mathcal{O}(m^2 \cdot n)$, i.e., independent of n'.

Comparing DOCS with Competitors on OTA. We compare with other five competitors (i.e., Baseline, AskIt!, IC, QASCA, and D-Max) that address task assignment. Note that task assignment also requires truth inference method to derive worker's quality and infer task's truth: (1) Baseline uses MV to infer truth and randomly selects k tasks to assign to the coming worker; (2) AskIt! [27] uses MV to infer truth and leverages an entropy-like method to select k tasks for assignment; (3) IC [63] uses weighted majority voting to infer truth, and intuitively, it selects k tasks for assignment such that the coming worker has the highest quality to answer, with the constraints that each task should be answered with the same number of times (in our scenario, exactly 10 times) in the end; (4) QASCA [222] uses D&S [47] to infer truth, and it selects k tasks such that the estimated quality (Accuracy in our scenario) can be improved most, and assigns them to the worker. (5) D-Max uses **TI** (Section 7.4) to infer truth, and it selects k tasks for assignment such that the coming worker has the matching domain to answer.

When comparing with different methods, we follow the instructions in Section 7.6.1 and assign $k = 3$ tasks using each method in parallel. There are 360×10 (3.6K), 4K, 10K, and 3.28K assignments in total for the four datasets. We show the Accuracy after all assignments and record the worst case assignment time in Figures 7.8(a)(b). We have the following observations: (1) Baseline performs worst as it randomly assigns tasks and does not consider the tasks' truth information or the coming worker's quality; (2) AskIt! considers tasks' truth in assignment, but does not take worker's quality into account; (3) QASCA performs better as it considers both the tasks' truth and worker's quality in assignment, but it does not take the domain information of tasks and workers into account; (4) IC captures a worker's quality for answering different tasks; however, it selects k tasks such that the worker has the highest quality to answer, which may assign tasks that are already confident enough. Furthermore, it restricts that each task should be answered with the same times, which does not consider that the assignments for the easy tasks can be saved for hard tasks; (5) although D-Max uses the **TI** (Section 7.4) to infer truth, it selects k tasks with the matching domain to the coming worker, which may assign tasks that are already confident enough; (6) DOCS performs the best, outperforming the best of other competitors consistently on all datasets. The reason is that we consider three factors: tasks' truth, worker's quality and the domain information. We estimate the benefit *if* a task will be assigned and answered by the worker, and selects the optimal k tasks which lead to the highest benefits; (7) all methods can finish the assignment efficiently, as it can be seen in Figure 7.8(b) that the worst case assignment is within 0.02s.

Scalability of OTA (Simulation). We generate n tasks with $m = 20$. Then we randomly generate the coming worker w's quality and the matrix $\mathcal{M}^{(i)}$ for each task t_i. Finally k tasks are assigned to worker w by running methods in Section 7.5.1. We vary $n \in$ [0, 10K], $k \in \{5, 10, 50\}$ and record the time in Figure 7.8(c). It can be seen that the assignment time increases linearly with n, which corresponds to the complexity $\mathcal{O}(nm\ell^2)$. We can also observe that the assignment time is independent of k. This is because that we use PICK algorithm [26] to select top k tasks with highest benefits, which is slightly affected by k. The assignment is efficient, and it can be finished within 0.2s with large data (n = 10K and k = 50).

7.7 Related Works

Since we have reviewed most of the related works of crowdsourcing in Chapter 2, this section only highlights the part related to domain aware task model and task assignment.

Domain Aware Task Model. Most existing crowdsourcing works [27,47,48,149, 222] do not differentiate between tasks. Recently, [200] models the difficulty in tasks, while [63] and [131] exploit the diverse domains in each task using topic models (i.e., LDA [25] and TwitterLDA [216]). However, [63,131] require a user to input the number of *latent* domains and cannot capture a task's related domain(s) explicitly and correctly, without considering the semantics in texts. We leverage the knowledge base (i.e., Freebase [71]), which has rich contextual and semantics information and can capture a task's diverse domain(s) in an explicit and accurate way. Note that there are other works [208,217] that model a task and worker's diverse domains, but they do not consider knowledge base and leverage the external information, e.g., an answer with thumbs-up and thumbs-down voted by other workers. However, in reality the external information is hard to get, thus we do not assume them available and only leverage tasks' text descriptions and workers' answers. There are also some crowdsourcing works [15, 40] that consider knowledge bases, but they focus on different perspectives, e.g., [40] studies data cleaning and [15] focuses on crowd mining.

Domain Aware Task Assignment. Knowledge-intensive crowdsourcing solutions require external information, e.g., workers' wages and acceptance ratio in [164], and a complete skill taxonomy tree, a worker's exact skills and the required skills of tasks in [138], which are hard to obtain in real crowdsourcing platforms (e.g., AMT [1]). While our work considers typical crowdsourcing settings used in existing platforms.

7.8 Chapter Summary

In this chapter, we first focus on question answering application, and generalize to the study of domain-aware tasks. To be specific, we have built a system DOCS, which contains three main modules: **DVE**, **TI** and **OTA**. After a requester submits tasks, **DVE** leverages the KB to interpret the domains for each task, and then DOCS interacts with AMT [1] adaptively. When a worker submits answers, **TI** is run to infer workers' qualities and tasks' truth, by exploring their inherent relations. When a worker requests for new tasks, **OTA** dynamically assigns k tasks with the highest *benefits* to the worker. We conduct experiments to test the effectiveness and efficiency, showing that DOCS outperforms existing state-of-the-art methods on the three modules.

In future work, we will consider the following directions: (1) we will see whether a more diverse sample of workers (e.g., the selected workers follow the expertise distribution) may help in truth inference, as now our model focuses on selecting a biased sample which only contains domain experts; (2) we will study the case if the domain of input data is not in text format (e.g., an image) and also the case if available knowledge base is not appropriate for the crowdsourced domains; (3) we will consider how the design of different interfaces would affect the crowdsourcing answers, e.g., one way is to first classify the tasks in different domains, and each HIT contains all tasks in the same domain; (4) we will consider different evaluation metrics (e.g., *F-score*) and see whether our methods work well in various evaluation metrics.

Chapter 8

Conclusions and Future Work

In this thesis, we address two fundamental components in crowdsourcing framework, i.e., task assignment and truth inference. We first dive deep into these two components, respectively (Chapters 3-4 and Chapter 5). We then discuss the combination of them and apply them to specific crowdsourcing applications (Chapters 6-7). This chapter closes the thesis by first outlining the conclusions (Section 8.1), and then briefly discussing some promising future directions (Section 8.2).

8.1 Conclusions

In this section, we conclude our solutions to task assignment (Chapters 3-4), truth inference (Chapter 5), and how to combine them together and apply them to specific crowdsourcing applications (Chapters 6-7).

• **Task Assignment (Chapters 3-4).** We focus on the task assignment problem under the task-based setting in Chapter 3. Given a set of tasks, when a worker comes, we focus on assigning a subset of tasks to the coming worker. To solve this, we propose a novel task assignment framework by incorporating evaluation metrics into assignment strategies, and formalize the *online task assignment* *problem* under the proposed framework. We generalize the definition of evaluation metrics to be able to quantify the result quality w.r.t a distribution matrix, and devise efficient algorithms to identify the optimal result of each task that can maximize the overall quality. Two respective linear online assignment algorithms are proposed that can efficiently select the best subset of tasks for a coming worker. We develop a system called QASCA, which enables a popular crowdsourcing platform (i.e., AMT) to support our task assignment framework. We evaluate the performance of QASCA on five real applications. Experimental results indicate that QASCA can achieve much better (of more than 8% improvement) result quality compared with five state-of-the-art systems.

We then study the task assignment problem under the worker-based setting in Chapter 4. Given a budget and a set of workers, we focus on selecting a subset of workers which attain the highest aggregated quality within the budget constraint. To solve this, we develop a polynomial-time approximation algorithm, which enables a large number of candidate solutions to be pruned, without a significant loss of accuracy. We further develop a theoretical error bound of this algorithm. Particularly, our approximate computation algorithm is proved to yield an error of not more than 1%. We also leverage a successful heuristic, the simulated annealing heuristic, by designing local neighborhood search functions. To evaluate our solutions, we have performed extensive evaluations on real and synthetic crowdsourced data. Our experimental results show that our algorithms can solve the problem effectively and efficiently. The quality of our solution is also consistently better than the existing works.

Considering the solutions to task assignment problem in Chapters 3 and 4, it might be interesting to study how to address the general task assignment problem by combining these two settings together, that is, *"Given a fixed budget, how to select suitable tasks and assign to the most appropriate workers?"*

• **Truth Inference (Chapter 5).** We perform a thorough analysis and experimental comparison of the solutions to the truth inference problem, i.e., given workers' answers collected for tasks, how to infer the truth of each task? To dive in, we survey 17 existing algorithms, summarize a framework, and provide an in-depth analysis and summary on the 17 algorithms from different perspectives, which can help practitioners to easily grasp existing truth inference algorithms. We also experimentally conduct a thorough comparison of these methods on 5 datasets with varying sizes, publicize our codes and datasets (Section 1.4) and provide experimental findings, which give guidance for selecting appropriate methods under various settings.

• **Multi-Label and Domain-Aware Tasks (Chapters 6-7).** We combine task assignment and truth inference together, and first apply them to multi-label tasks in Chapter 6. For task assignment, we develop an effective algorithm that judiciously selects a subset of tasks with the largest amount of uncertainty reduction for the current worker, in linear time. For truth inference, we propose an effective worker model, and devise a method that jointly infers each task's truth and each worker's model. We further consider how to integrate label correlations into our method. Finally, we develop Comet, and use two real-world datasets to perform experiments on two crowdsourcing platforms. Results show that Comet outperforms existing state-of-the-art methods, and is robust under various settings. It achieves over 20% improvements on the two datasets performed by low-quality workers. We also conduct experiments on simulated data, in order to verify the scalability of Comet.

We then apply task assignment and truth inference to domain-aware tasks in Chapter 7. There are three components: domain vector estimation, task assignment, and truth inference. For domain vector estimation, we propose an algorithm that can reduce the complexity from exponential to polynomial. For task assignment, we have developed an optimal and linear algorithm. For truth inference, we exploit the inherent relations between workers' qualities and tasks' truth, and finally devise an iterative approach that collectively infers those parameters. We also study how to maintain each worker's quality in the long run and devise update policies for the incremental inference algorithms.

8.2 Future Work

In this section, we first provide some research challenges for task assignment and truth inference respectively. We then illustrate other research opportunities in crowdsourced data management.

• **Task Assignment.** (1) As the worker history is maintained and expected to come back to answer another set of tasks, it might be interesting to construct certain scope of active advertisers and actively recruit them to answer tasks. (2) We will further investigate the method of dealing with continuous values and more complicated task types (e.g., cluster-based task [192] and transcription task [93]). (3) More evaluation metrics will be incorporated in the assignment framework. (4) We focus on task assignment over a specific (or homogeneous) set of tasks, then how to incorporate heterogeneous tasks into the assignment framework is another direction. (5) We also plan to consider if a requester has multiple metrics in her mind (say both *Accuracy* and *F-score*), then how can we wisely assign tasks to a coming worker.

• **Truth Inference.** (1) Task Types. In decision-making and single-label tasks, there is no "best" method that beats others, and we recommend D&S [47] and LFC [161], which are relatively more effective and efficient. In numeric tasks, we recommend Mean and LFC_N [161], and there is still room to improve numeric tasks. Moreover, there are other more complicated task types that are merely studied, e.g., translation tasks [93], or tasks that require workers to collect data [190]. (2) Task Design. In order to collect high quality crowdsourced data in an efficient way, it is important to design tasks with friendly User Interface (UI) with a feasible price. Although there are some works that study how to set the prices [75] and acquire answers from crowd more efficiently [82], the design of friendly UI is not studied extensively. It is also interesting to study the relations between the design of UI, price, worker's latency and quality. (3) Data Redundancy. The quality significantly increases with small redundancy, and keeps stable for a large redundancy. Then how to estimate the data redundancy with stable quality? Is it possible to estimate the improvement with more data redundancy? (4) Qualification Test. Not all methods can benefit from qualification test, and the quality of some methods even decrease. So is it possible to estimate the benefit of qualification test for a method? (5) Hidden Test. Although most methods can benefit from them, the improvements vary in different datasets and methods. Thus is it possible to estimate the improvement with hidden test (i.e., a number of golden tasks) for a method on a dataset? (6) Incorporation of More Rich Features. We only consider the collected answers for tasks; however, we do not incorporate more rich information, for example, the contexts in tasks (each task's textual descriptions or the pixels in image tasks). These have been mentioned in existing works [170, 207]. We do not include those into comparisons since most of the datasets do not make the original tasks public. It might be interesting to see how much improvement when such information is considered.

• **Query Optimization.** An SQL query often corresponds to multiple query plans and it relies on a query optimizer to select the best plan. Traditionally, the way a query optimizer works is to estimate the computation cost of each query plan and choose the one with the minimum estimated cost. However, this process turns to be quite challenging in a crowdsourcing environment because (1) there are three optimization objectives (result quality, monetary cost, and latency) that need to be considered; (2) humans are much more unpredictable than machines.

• **Benchmark.** A large variety of TPC benchmarks (e.g., TPC-H for analytic workloads, TPC-DI for data integration) standardize performance comparisons for database systems and promote the development of database research. Although we maintained some open-source public datasets (Section 1.4), there is still lack of standardized benchmarks available. In order to better explore the research topic, it is important to study how to develop evaluation methodologies and benchmarks for crowdsourced data management systems.

• **Big Data.** In the big data era, data volumes are increasing very fast. Compared to machines, humans are much more expensive, and thus it would be increasingly more costly to apply crowdsourcing to emerging big data settings. There are existing works that aim to address this problem, but they only work for some certain data processing tasks, such as data cleaning [40], data labeling [142]. Therefore, it is important to continue this study and to develop new techniques that work for all kinds of data processing tasks.

• **Macro-Tasks.** Most of existing studies focus on micro-tasks, which can be easily assigned to workers and instantly answered by workers. However, many real applications need to use macro-tasks, such as writing a paper. Macro-tasks are hard to be split and accomplished by multiple workers, because they will loose the context information if they are split [81]. Workers are not interested in answering a whole macro-task as each macro-task will take a long time. Thus it is rather challenging to support macro-tasks, including automatically splitting a macro-task, assigning tasks to crowd or machines, and automatically aggregating the answers.

• **Privacy.** There are several types of privacy issues in crowdsourcing. First, the requester wants to protect the privacy of their tasks [203]. The tasks may contain sensitive attributes and could cause privacy leakage. Malicious workers could link them with other public datasets to reveal individual private information. Although the requester can publish anonymity data to the workers using existing privacy techniques, e.g., K-Anonymity, it may lower down the quality as the workers cannot get the precise data. Thus it is challenging to trade-off the accuracy and privacy for requesters. Second, the workers have privacy-preserving requirement. Personal information of workers can be inferred from the answers provided by the workers, such as their locations, professions, hobbies. On the other hand, the requester wants to assign their tasks to appropriate workers that are skilled at their tasks (or close to the tasks).

• **Mobile Crowdsourcing.** With the growing popularity of smartphones, there are emerging mobile crowdsourcing platforms, e.g., gMission [37], ChinaCrowd [3]. These mobile platforms pose new challenges for crowdsourced data management. First, more factors (e.g., spatial distance, mobile user interface) will affect workers' latency and quality. It is more challenging to control quality, latency and cost for mobile platforms. Second, traditional crowdsourcing platforms adopt worker selection model to assign tasks; however, mobile crowdsourcing requires to support server assignment model, which calls for new task assignment techniques.

Bibliography

[1] Amazon mechanical turk. https://www.mturk.com/.

[2] Chi-squared distribution. https://en.wikipedia.org/wiki/Chi-squared_distribution.

[3] Chinacrowd. http://www.chinacrowds.com.

[4] Convnet. https://code.google.com/p/cuda-convnet/.

[5] Crowdflower. http://www.crowdflower.com.

[6] An example in yahoo answers. http://answers.yahoo.com/question/index?qid=20071211155603AAKwtyr.

[7] External hit. http://docs.aws.amazon.com/AWSMechTurk/latest/AWSMturkAPI/ApiReference_ExternalQuestionArticle.html.

[8] Flickr. https://www.flickr.com/.

[9] Random ballot. http://en.wikipedia.org/wiki/Random_ballot.

[10] Upwork. https://www.upwork.com.

[11] Adult Datset. https://github.com/ipeirotis/Get-Another-Label/tree/master/data.

[12] Abdullah Alfarrarjeh, Tobias Emrich, and Cyrus Shahabi. Scalable spatial crowdsourcing: A study of distributed algorithms. In *MDM*, volume 1, pages 134–144. IEEE, 2015.

[13] Sihem Amer-Yahia and Senjuti Basu Roy. Human factors in crowdsourcing. *PVLDB*, 9(13):1615–1618, 2016.

[14] Yael Amsterdamer, Susan Davidson, Anna Kukliansky, Tova Milo, Slava Novgorodov, and Amit Somech. Managing general and individual knowledge in crowd mining applications. In *CIDR*, 2015.

[15] Yael Amsterdamer, Susan B Davidson, Tova Milo, Slava Novgorodov, and Amit Somech. Oassis: query driven crowd mining. In *SIGMOD*, pages 589–600. ACM, 2014.

[16] Yael Amsterdamer, Susan B Davidson, Tova Milo, Slava Novgorodov, and Amit Somech. Ontology assisted crowd mining. *PVLDB*, 7(13):1597–1600, 2014.

[17] Yael Amsterdamer, Yael Grossman, Tova Milo, and Pierre Senellart. Crowd mining. In *SIGMOD*, pages 241–252. ACM, 2013.

[18] Yael Amsterdamer, Yael Grossman, Tova Milo, and Pierre Senellart. Crowdminer: Mining association rules from the crowd. *PVLDB*, 6(12):1250–1253, 2013.

[19] A.P.Dawid and A.M.Skene. Maximum likelihood estimation of observer error-rates using em algorithm. *Appl.Statist.*, 28(1):20–28, 1979.

[20] Ashish Goel and David Lee, Triadic Consensus: A Randomized Algorithm for Voting in a Crowd. `http://arxiv.org/pdf/1210.0664v1.pdf`.

[21] Bahadir Ismail Aydin, Yavuz Selim Yilmaz, Yaliang Li, Qi Li, Jing Gao, and Murat Demirbas. Crowdsourcing for multiple-choice question answering. In *AAAI*, pages 2946–2953, 2014.

[22] Zafer Barutcuoglu, Robert E Schapire, and Olga G Troyanskaya. Hierarchical multi-label prediction of gene function. *Bioinformatics*, 22(7):830–836, 2006.

[23] Bayes' Theorem. `http://en.wikipedia.org/wiki/Bayes'_theorem`.

[24] Wei Bi and James T Kwok. Multilabel classification with label correlations and missing labels. In *AAAI*, 2014.

[25] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *JMLR*, 3(Jan):993–1022, 2003.

[26] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973.

[27] Rubi Boim, Ohad Greenshpan, Tova Milo, Slava Novgorodov, Neoklis Polyzotis, and Wang-Chiew Tan. Asking the right questions in crowd data sourcing. In *ICDE*, pages 1261–1264. IEEE, 2012.

[28] David Bookstaber. Simulated annealing for traveling salesman problem.

[29] Jonathan Bragg, Daniel S Weld, et al. Crowdsourcing multi-label classification for taxonomy creation. In *HCOMP*, 2013.

[30] David Guy Brizan and Abdullah Uz Tansel. A. survey of entity resolution and record linkage methodologies. *Communications of the IIMA*, 6(3):5, 2015.

[31] Chris Buckley, Matthew Lease, and Mark D. Smucker. Overview of the trec 2010 relevance feedback track (notebook). In *The Nineteenth TREC Notebook*, 2010.

[32] Chris Callison-Burch. Fast, cheap, and creative: evaluating translation quality using amazon's mechanical turk. In *EMNLP*, pages 286–295, 2009.

[33] Caleb Chen Cao, Jieying She, Yongxin Tong, and Lei Chen. Whom to ask? jury selection for decision making tasks on micro-blog services. *PVLDB*, 5(11):1495–1506, 2012.

[34] C.G.Small. *Expansions and asymptotics for statistics*. CRC Press, 2010.

[35] Chengliang Chai, Guoliang Li, Jian Li, Dong Deng, and Jianhua Feng. Cost-effective crowdsourced entity resolution: A partial-order approach. In *SIGMOD*, pages 969–984, 2016.

[36] Lei Chen, Dongwon Lee, and Tova Milo. Data-driven crowdsourcing: Management, mining, and applications. In *ICDE*, pages 1527–1529. IEEE, 2015.

[37] Zhao Chen, Rui Fu, Ziyuan Zhao, Zheng Liu, Leihao Xia, Lei Chen, Peng Cheng, Caleb Chen Cao, Yongxin Tong, and Chen Jason Zhang. gmission: a general spatial crowdsourcing platform. *PVLDB*, 7(13):1629–1632, 2014.

[38] X. Cheng and D. Roth. Relational inference for wikification. In *EMNLP*, pages 1787–1796, 2013.

[39] Lydia B Chilton, Greg Little, Darren Edge, Daniel S Weld, and James A Landay. Cascade: Crowdsourcing taxonomy creation. In *SIGCHI*, pages 1999–2008. ACM, 2013.

[40] Xu Chu, John Morcos, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *SIGMOD*, pages 1247–1261, 2015.

[41] Compositions. `http://mathworld.wolfram.com/Composition.html`.

[42] Joana Costa, Catarina Silva, Mário Antunes, and Bernardete Ribeiro. On using crowdsourcing and active learning to improve classification performance. In *ISDA*, 2011.

[43] Crowdsourcing Datasets. `http://dbgroup.cs.tsinghua.edu.cn/ligl/crowddata/`.

[44] Peng Dai, Christopher H. Lin, Mausam, and Daniel S. Weld. Pomdp-based control of workflows for crowdsourcing. *Artif. Intell.*, 202:52–85, 2013.

[45] Sanjib Das, Paul Suganthan G.C., AnHai Doan, Jeffrey F. Naughton, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, Vijay Raghavendra, and Youngchoon Park. Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services. In *SIGMOD*, SIGMOD '17, pages 1431–1446, New York, NY, USA, 2017. ACM.

[46] Susan B. Davidson, Sanjeev Khanna, Tova Milo, and Sudeepa Roy. Using the crowd for top-k and group-by queries. In *ICDT*, pages 225–236, 2013.

[47] Alexander Philip Dawid and Allan M Skene. Maximum likelihood estimation of observer error-rates using the em algorithm. *Applied statistics*, pages 20–28, 1979.

[48] Gianluca Demartini, Djellel Eddine Difallah, and Philippe Cudré-Mauroux. Zencrowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In *WWW*, pages 469–478, 2012.

[49] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *J.R.Statist.Soc.B*, 30(1):1–38, 1977.

[50] Dong Deng, Guoliang Li, Shuang Hao, Jiannan Wang, and Jianhua Feng. Massjoin: A mapreduce-based method for scalable string similarity joins. In *ICDE*, pages 340–351, 2014.

[51] Jia Deng, Olga Russakovsky, Jonathan Krause, Michael S Bernstein, Alex Berg, and Li Fei-Fei. Scalable multi-label annotation. In *SIGCHI*, pages 3099–3102. ACM, 2014.

[52] Werner Dinkelbach. On nonlinear fractional programming. *Management Science*, 13(7):492–498, March,1967.

[53] A Doan, Michael J Franklin, Donald Kossmann, and Tim Kraska. Crowdsourcing applications and platforms: A data management perspective. *PVLDB*, 4(12):1508–1509, 2011.

[54] Xin Luna Dong, Barna Saha, and Divesh Srivastava. Less is more: Selecting sources wisely for integration. *PVLDB*, 6(2):37–48, 2012.

[55] A. Drexl. A simulated annealing approach to the multiconstraint zero-one knapsack problem. *Computing*, 40:1–8, 1988.

[56] Lei Duan, Satoshi Oyama, Haruhiko Sato, and Masahito Kurihara. Separate or joint? estimation of multiple labels from crowdsourced annotations. *Expert Systems with Applications*, 41(13):5723–5732, 2014.

[57] John Duggan and Cesar Martinelli. A bayesian model of voting in juries. *Games and Economic Behavior*, 37(2):259–294, 2001.

[58] Pinar Duygulu, Kobus Barnard, Joao FG de Freitas, and David A Forsyth. Object recognition as machine translation: Learning a lexicon for a fixed image vocabulary. In *ECCV*, pages 97–112. Springer, 2002.

[59] Pavlos Efraimidis and Paul G. Spirakis. Weighted random sampling with a reservoir. *Inf. Process. Lett.*, 97(5):181–185, 2006.

[60] Bradley Efron and Robert J Tibshirani. *An introduction to the bootstrap*. 1994.

[61] Christopher B Eiben, Justin B Siegel, Jacob B Bale, Seth Cooper, Firas Khatib, Betty W Shen, Foldit Players, Barry L Stoddard, Zoran Popovic, and David Baker. Increased diels-alderase activity through backbone remodeling guided by foldit players. *Nature biotechnology*, 30(2):190–192, 2012.

[62] Brian Eriksson. Learning to top-k search using pairwise comparisons. In *AISTATS*, pages 265–273, 2013.

[63] Ju Fan, Guoliang Li, Beng Chin Ooi, Kian-Lee Tan, and Jianhua Feng. icrowd: An adaptive crowdsourcing framework. In *SIGMOD*, pages 1015–1030, 2015.

[64] Ju Fan, Meiyu Lu, Beng Chin Ooi, Wang-Chiew Tan, and Meihui Zhang. A hybrid machine-crowdsourcing system for matching web tables. In *ICDE*, 2014.

[65] Ju Fan, Meihui Zhang, Stanley Kok, Meiyu Lu, and Beng Chin Ooi. Crowdop: Query optimization for declarative crowdsourcing systems. *TKDE*, 27(8):2078–2092, 2015.

[66] Meng Fang, Jie Yin, and Dacheng Tao. Active learning for crowdsourcing using knowledge transfer. In *AAAI*, pages 1809–1815, 2014.

[67] Siamak Faradani, Bjoern Hartmann, and Panagiotis G. Ipeirotis. What's the right price? pricing tasks for finishing on time. In *AAAI Workshop*, 2011.

[68] Farnoush Farhadi, Elham Hoseini, Sattar Hashemi, and Ali Hamzeh. Teamfinder: A co-clustering based framework for finding an effective team of experts in social networks. In *ICDM Workshops*, pages 107–114, 2012.

[69] Amber Feng, Michael J. Franklin, Donald Kossmann, Tim Kraska, Samuel Madden, Sukriti Ramesh, Andrew Wang, and Reynold Xin. Crowddb: Query processing with the vldb crowd. *PVLDB*, 4(12):1387–1390, 2011.

[70] Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. Crowddb: answering queries with crowdsourcing. In *SIGMOD*, pages 61–72, 2011.

[71] Freebase. `https://www.freebase.com/`.

[72] Jing Gao, Qi Li, Bo Zhao, Wei Fan, and Jiawei Han. Truth discovery and crowdsourcing aggregation: A unified perspective. *PVLDB*, 8(12):2048–2049, 2015.

[73] Jing Gao, Qi Li, Bo Zhao, Wei Fan, and Jiawei Han. Mining reliable information from passively and actively crowdsourced data. In *SIGKDD*, pages 2121–2122. ACM, 2016.

[74] Jinyang Gao, Xuan Liu, Beng Chin Ooi, Haixun Wang, and Gang Chen. An online cost sensitive decision-making method in crowdsourcing systems. In *SIGMOD*, 2013.

[75] Yihan Gao and Aditya G. Parameswaran. Finish them!: Pricing algorithms for human computation. *PVLDB*, 7(14):1965–1976, 2014.

[76] Chaitanya Gokhale, Sanjib Das, AnHai Doan, Jeffrey F. Naughton, Narasimhan Rampalli, Jude W. Shavlik, and Xiaojin Zhu. Corleone: hands-off crowdsourcing for entity matching. In *SIGMOD*, pages 601–612, 2014.

[77] Ryan Gomes, Peter Welinder, Andreas Krause, and Pietro Perona. Crowdclustering. In *NIPS*, pages 558–566, 2011.

[78] Benoît Groz and Tova Milo. Skyline queries with noisy comparisons. In *PODS*, pages 185–198, 2015.

[79] Anja Gruenheid, Donald Kossmann, Sukriti Ramesh, and Florian Widmer. Crowdsourcing entity resolution: When is A=B? Technical report, ETH Zürich.

[80] Stephen Guo, Aditya G. Parameswaran, and Hector Garcia-Molina. So who won?: dynamic max discovery with the crowd. In *SIGMOD*, pages 385–396, 2012.

[81] Daniel Haas, Jason Ansel, Lydia Gu, and Adam Marcus. Argonaut: Macrotask crowdsourcing for complex data processing. *PVLDB*, 8(12):1642–1653, 2015.

[82] Daniel Haas, Jiannan Wang, Eugene Wu, and Michael J Franklin. Clamshell: Speeding up crowds for low-latency data labeling. *PVLDB*, 9(4):372–383, 2015.

[83] Kotaro Hara, Vicki Le, and Jon Froehlich. Combining crowdsourcing and google street view to identify street-level accessibility problems. In *SIGCHI*, pages 631–640, 2013.

[84] Hannes Heikinheimo and Antti Ukkonen. The crowd-median algorithm. In *HCOMP*, 2013.

[85] Chien-Ju Ho, Shahin Jabbari, and Jennifer Wortman Vaughan. Adaptive task assignment for crowdsourced classification. In *ICML*, pages 534–542, 2013.

[86] Chien-Ju Ho and Jennifer Wortman Vaughan. Online task assignment in crowdsourcing markets. In *AAAI*, 2012.

[87] John Joseph Horton and Lydia B. Chilton. The labor economics of paid crowdsourcing. In *ACM Conference on Electronic Commerce*, pages 209–218, 2010.

[88] Huiqi Hu, Guoliang Li, Zhifeng Bao, Yan Cui, and Jianhua Feng. Crowdsourcing-based real-time urban traffic speed estimation: From trends to speeds. In *ICDE*, pages 883–894, 2016.

[89] Huiqi Hu, Yudian Zheng, Zhifeng Bao, Guoliang Li, Jianhua Feng, and Reynold Cheng. Crowdsourced POI labelling: Location-aware result inference and task assignment. In *ICDE*, pages 61–72, 2016.

[90] Nguyen Quoc Viet Hung, Nguyen Thanh Tam, Zoltán Miklós, and Karl Aberer. On leveraging crowdsourcing techniques for schema matching networks. In *Database Systems for Advanced Applications*, pages 139–154. Springer, 2013.

[91] Nguyen Quoc Viet Hung, Nguyen Thanh Tam, Lam Ngoc Tran, and Karl Aberer. An evaluation of aggregation techniques in crowdsourcing. In *WISE*, pages 1–15. Springer, 2013.

[92] P. Ipeirotis, F. Provost, and J. Wang. Quality management on amazon mechanical turk. In *SIGKDD workshop*, pages 64–67, 2010.

[93] Panagiotis G. Ipeirotis. Analyzing the amazon mechanical turk marketplace. *ACM Crossroads*, 17(2):16–21, 2010.

[94] Panagiotis G Ipeirotis and Praveen K Paritosh. Managing crowdsourced human computation: a tutorial. pages 287–288, 2011.

[95] Martin Jansche. A maximum expected utility framework for binary sequence labeling. In *ACL*, 2007.

[96] Shawn R. Jeffery, Michael J. Franklin, and Alon Y. Halevy. Pay-as-you-go user feedback for dataspace systems. In *SIGMOD*, pages 847–860, 2008.

[97] Heng Ji, Ralph Grishman, Hoa Trang Dang, Kira Griffitt, and Joe Ellis. Overview of the tac 2010 knowledge base population track. In *TAC*, 2010.

[98] Manas Joglekar, Hector Garcia-Molina, and Aditya G. Parameswaran. Evaluating the crowd with confidence. In *SIGKDD*, pages 686–694, 2013.

[99] Haim Kaplan, Ilia Lotosh, Tova Milo, and Slava Novgorodov. Answering planning queries with the crowd. *PVLDB*, 6(9):697–708, 2013.

[100] David R. Karger, Sewoong Oh, and Devavrat Shah. Iterative learning for reliable crowdsourcing systems. In *NIPS*, pages 1953–1961, 2011.

[101] Leyla Kazemi, Cyrus Shahabi, and Lei Chen. Geotrucrowd: trustworthy query answering with spatial crowdsourcing. In *SIGSPATIAL*, pages 304–313, 2013.

[102] Asif R. Khan and Hector Garcia-Molina. Hybrid strategies for finding the max with the crowd. Technical report, 2014.

[103] Asif R. Khan and Hector Garcia-Molina. Crowddqs: Dynamic question selection in crowdsourcing systems. In *SIGMOD*, SIGMOD '17, pages 1447–1462, New York, NY, USA, 2017. ACM.

[104] Hyun-Chul Kim and Zoubin Ghahramani. Bayesian classifier combination. In *AISTATS*, pages 619–627, 2012.

[105] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, New Series*, 220(4598):671–680, 1983.

[106] Bryan Klimt and Yiming Yang. Introducing the enron corpus. In *Proceedinds of the 1st Conference on Email and Anti-Spam*, 2004.

[107] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models - Principles and Techniques*. MIT Press, 2009.

[108] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[109] Aditya Kurve, David J. Miller, and George Kesidis. Multicategory crowdsourcing accounting for variable task difficulty, worker skill, and worker intention. *TKDE*, 27(3):794–809, 2015.

[110] Alexandre Lacasse, François Laviolette, Mario Marchand, and Francis Turgeon-Boutin. Learning with randomized majority votes. pages 162–177, 2010.

[111] Alberto H.F. Laender, Marcos André Gonçalves, Ricardo G. Cota, Anderson A. Ferreira, Rodrygo L. T. Santos, and Allan J.C. Silva. Keeping a

digital library clean: New solutions to old problems. In *DocEng*, pages 257–262. ACM, 2008.

[112] L.D. Landau and E.M. Lifshitz. *Statistical Physics. Course of Theoretical Physics 5 (3 ed.)*. Oxford: Pergamon Press, 1980.

[113] Theodoros Lappas, Kun Liu, and Evimaria Terzi. Finding a team of experts in social networks. In *KDD*, pages 467–476, 2009.

[114] David D. Lewis. Evaluating and optimizing autonomous text classification systems. In *SIGIR*, pages 246–254, 1995.

[115] Guoliang Li, Chengliang Chai, Ju Fan, Xueping Weng, Jian Li, Yudian Zheng, Yuanbing Li, Xiang Yu, Xiaohang Zhang, and Haitao Yuan. CDB: optimizing queries with crowd-based selections and joins. In *SIGMOD*, pages 1463–1478, 2017.

[116] Guoliang Li, Jiannan Wang, Yudian Zheng, and Michael J. Franklin. Crowdsourced data management: A survey. *TKDE*, 28(9):2296–2319, 2016.

[117] Guoliang Li, Yudian Zheng, Ju Fan, Jiannan Wang, and Reynold Cheng. Crowdsourced data management: Overview and challenges. In *SIGMOD*, pages 1711–1716, 2017.

[118] Qi Li, Yaliang Li, Jing Gao, Lu Su, Bo Zhao, Murat Demirbas, Wei Fan, and Jiawei Han. A confidence-aware approach for truth discovery on long-tail data. *PVLDB*, 8(4):425–436, 2014.

[119] Qi Li, Yaliang Li, Jing Gao, Bo Zhao, Wei Fan, and Jiawei Han. Resolving conflicts in heterogeneous data by truth discovery and source reliability estimation. In *SIGMOD*, pages 1187–1198, 2014.

[120] Qi Li, Fenglong Ma, Jing Gao, Lu Su, and Christopher J Quinn. Crowdsourcing high quality labels with a tight budget. In *WSDM*, 2016.

[121] Xian Li, Xin Luna Dong, Kenneth Lyons, Weiyi Meng, and Divesh Srivastava. Truth finding on the deep web: Is the problem solved? *PVLDB*, 6(2):97–108, 2012.

[122] Yaliang Li, Jing Gao, Chuishi Meng, Qi Li, Lu Su, Bo Zhao, Wei Fan, and Jiawei Han. A survey on truth discovery. *SIGKDD Explorations*, 17(2):1–16, 2015.

[123] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. *Inf. Comput.*, 108(2):212–261, February 1994.

[124] Bing Liu. Sentiment analysis and opinion mining. *Synthesis lectures on human language technologies*, 5(1):1–167, 2012.

[125] Bing Liu and Lei Zhang. A survey of opinion mining and sentiment analysis. In *Mining text data*, pages 415–463. Springer, 2012.

[126] Qiang Liu, Jian Peng, and Alexander T. Ihler. Variational inference for crowdsourcing. In *NIPS*, pages 701–709, 2012.

[127] Xuan Liu, Meiyu Lu, Beng Chin Ooi, Yanyan Shen, Sai Wu, and Meihui Zhang. Cdas: A crowdsourcing data analytics system. *PVLDB*, 5(10):1040–1051, 2012.

[128] Christoph Lofi, Kinda El Maarry, and Wolf-Tilo Balke. Skyline queries in crowd-enabled databases. In *EDBT*, pages 465–476, 2013.

[129] Christoph Lofi, Kinda El Maarry, and Wolf-Tilo Balke. Skyline queries over incomplete data - error models for focused crowd-sourcing. In *ER*, pages 298–312, 2013.

[130] Ilia Lotosh, Tova Milo, and Slava Novgorodov. Crowdplanr: Planning made easy with crowd. In *ICDE*, pages 1344–1347. IEEE, 2013.

[131] Fenglong Ma, Yaliang Li, Qi Li, Minghui Qiu, Jing Gao, Shi Zhi, Lu Su, Bo Zhao, Heng Ji, and Jiawei Han. Faitcrowd: Fine grained truth discovery for crowdsourced data aggregation. In *KDD*, pages 745–754, 2015.

[132] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008.

[133] Christopher D. Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT Press, 2001.

[134] Adam Marcus, David R. Karger, Samuel Madden, Rob Miller, and Sewoong Oh. Counting with the crowd. *PVLDB*, 6(2):109–120, 2012.

[135] Adam Marcus and Aditya Parameswaran. Crowdsourced data management industry and academic perspectives. *Foundations and Trends in Databases*, 6(1-2):1–161, 2015.

[136] Adam Marcus, Eugene Wu, David R. Karger, Samuel Madden, and Robert C. Miller. Human-powered sorts and joins. *PVLDB*, 5(1):13–24, 2011.

[137] Adam Marcus, Eugene Wu, Samuel Madden, and Robert C. Miller. Crowdsourced databases: Query processing with people. In *CIDR*, pages 211–214, 2011.

[138] Panagiotis Mavridis, David Gross-Amblard, and Zoltán Miklós. Using hierarchical skills for optimized task assignment in knowledge-intensive crowdsourcing. In *WWW*, pages 843–853, 2016.

[139] David Menestrina, Steven Euijong Whang, and Hector Garcia-Molina. Evaluating entity resolution results. *PVLDB*, 3(1-2):208–219, 2010.

[140] Rui Meng, Lei Chen, Yongxin Tong, and Chen Zhang. Knowledge base semantic integration using crowdsourcing. *TKDE*, 29(5):1087–1100, May 2017.

[141] Luyi Mo, Reynold Cheng, Ben Kao, Xuan S. Yang, Chenghui Ren, Siyu Lei, David W. Cheung, and Eric Lo. Optimizing plurality for human intelligence tasks. In *CIKM*, pages 1929–1938, 2013.

[142] Barzan Mozafari, Purna Sarkar, Michael Franklin, Michael Jordan, and Samuel Madden. Scaling up crowd-sourcing to very large datasets: a case for active learning. *PVLDB*, 8(2):125–136, 2014.

[143] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience, 1988.

[144] Quoc Viet Hung Nguyen, Thanh Tam Nguyen, Zoltán Miklós, Karl Aberer, Asaf Gal, and Matthias Weidlich. Pay-as-you-go reconciliation in schema matching networks. In *ICDE*, pages 220–231. IEEE, 2014.

[145] Stefanie Nowak and Stefan Rüger. How reliable are annotations via crowdsourcing: a study about inter-annotator agreement for multi-label image annotation. In *MIR*, pages 557–566, 2010.

[146] Sebastian Nowozin. Optimal decisions from probabilistic models: The intersection-over-union case. In *CVPR*, pages 548–555, 2014.

[147] Wentao Robin Ouyang, Lance M. Kaplan, Paul Martin, Alice Toniolo, Mani B. Srivastava, and Timothy J. Norman. Debiasing crowdsourced quantitative characteristics in local businesses and services. In *IPSN*, pages 190–201, 2015.

[148] Aditya G. Parameswaran, Stephen Boyd, Hector Garcia-Molina, Ashish Gupta, Neoklis Polyzotis, and Jennifer Widom. Optimal crowd-powered rating and filtering algorithms. *PVLDB*, 7(9):685–696, 2014.

[149] Aditya G. Parameswaran, Hector Garcia-Molina, Hyunjung Park, Neoklis Polyzotis, Aditya Ramesh, and Jennifer Widom. Crowdscreen: algorithms for filtering data with humans. In *SIGMOD*, pages 361–372, 2012.

[150] Aditya G. Parameswaran, Anish Das Sarma, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. Human-assisted graph search: it's okay to ask questions. *PVLDB*, 4(5):267–278, 2011.

[151] Aditya Ganesh Parameswaran, Hyunjung Park, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. Deco: declarative crowdsourcing. In *CIKM*, pages 1203–1212, 2012.

[152] Hyunjung Park and Jennifer Widom. Crowdfill: collecting structured data from the crowd. In *SIGMOD*, pages 577–588, 2014.

[153] Partition Problem. `http://en.wikipedia.org/wiki/Partition_problem`.

[154] Thomas Pfeiffer, Xi Alice Gao, Yiling Chen, Andrew Mao, and David G. Rand. Adaptive polling for information aggregation. In *AAAI*, 2012.

[155] Ravali Pochampally, Anish Das Sarma, Xin Luna Dong, Alexandra Meliou, and Divesh Srivastava. Fusing data with correlations. In *SIGMOD*, pages 433–444, 2014.

[156] Layla Pournajaf, Li Xiong, Vaidy Sunderam, and Slawomir Goryczka. Spatial task assignment for crowd sensing with cloaked locations. In *MDM*, volume 1, pages 73–82. IEEE, 2014.

[157] Project Page. `http://dbgroup.cs.tsinghua.edu.cn/ligl/crowd_truth_inference/`.

[158] QA. `https://webscope.sandbox.yahoo.com/catalog.php?datatype=l&did=76`.

[159] L. Ratinov, D. Roth, D. Downey, and M. Anderson. Local and global algorithms for disambiguation to wikipedia. In *ACL*, pages 1375–1384, 2011.

[160] Vikas C. Raykar and Shipeng Yu. Eliminating spammers and ranking annotators for crowdsourced labeling tasks. *Journal of Machine Learning Research*, 13:491–518, 2012.

[161] Vikas C Raykar, Shipeng Yu, Linda H Zhao, Gerardo Hermosillo Valadez, Charles Florin, Luca Bogoni, and Linda Moy. Learning from crowds. *JMLR*, 11(Apr):1297–1322, 2010.

[162] Theodoros Rekatsinas, Xin Luna Dong, and Divesh Srivastava. Characterizing and selecting fresh data sources. In *SIGMOD*, pages 919–930. ACM, 2014.

[163] S. H. Rice. A stochastic version of the price equation reveals the interplay of deterministic and stochastic processes in evolution. *BMC evolutionary biology*, 8:262, 2008.

[164] Senjuti Basu Roy, Ioanna Lykourentzou, Saravanan Thirumuruganathan, Sihem Amer-Yahia, and Gautam Das. Task assignment optimization in knowledge-intensive crowdsourcing. *VLDBJ*, 24(4):467–491, 2015.

[165] Anish Das Sarma, Aditya G. Parameswaran, Hector Garcia-Molina, and Alon Y. Halevy. Crowd-powered find algorithms. In *ICDE*, pages 964–975, 2014.

[166] Asad B. Sayeed, Timothy J. Meyer, Hieu C. Nguyen, Olivia Buzek, and Amy Weinberg. Crowdsourcing the evaluation of a domain-adapted named entity recognition system. In *HLT-NAACL*, pages 345–348, 2010.

[167] C. E. Shannon. A mathematical theory of communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 5(1):3–55, January 2001.

[168] Wei Shen, Jianyong Wang, and Jiawei Han. Entity linking with a knowledge base: Issues, techniques, and solutions. *TKDE*, 27(2):443–460, 2015.

[169] Aashish Sheshadri and Matthew Lease. SQUARE: A Benchmark for Research on Computing Crowd Consensus. In *Proceedings of the 1st AAAI Conference on Human Computation (HCOMP)*, pages 156–164, 2013.

[170] Edwin D Simpson, Matteo Venanzi, Steven Reece, Pushmeet Kohli, John Guiver, Stephen J Roberts, and Nicholas R Jennings. Language understanding in the wild: Combining crowdsourcing and machine learning. In *WWW*, pages 992–1002, 2015.

[171] Yaron Singer and Manas Mittal. Pricing mechanisms for crowdsourcing markets. In *WWW*, pages 1157–1166, 2013.

[172] Padhraic Smyth, Usama M. Fayyad, Michael C. Burl, Pietro Perona, and Pierre Baldi. Inferring ground truth from subjective labelling of venus images. In *NIPS*, pages 1085–1092, 1994.

[173] Rion Snow, Brendan O'Connor, Daniel Jurafsky, and Andrew Y Ng. Cheap and fast—but is it good?: evaluating non-expert annotations for natural language tasks. In *EMNLP*, pages 254–263, 2008.

[174] Han Su, Kai Zheng, Jiamin Huang, Hoyoung Jeung, Lei Chen, and Xiaofang Zhou. Crowdplanner: A crowd-based route recommendation system. In *ICDE*, pages 1144–1155. IEEE, 2014.

[175] Han Su, Kai Zheng, Jiamin Huang, Tianyu Liu, Haozhou Wang, and Xiaofang Zhou. A crowd-based route recommendation system-crowdplanner. In *ICDE*, pages 1178–1181, 2014.

[176] John R. Talburt. *Entity Resolution and Information Quality*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010.

[177] Yongxin Tong, Caleb Chen Cao, and Lei Chen. Tcs: Efficient topic discovery over crowd-oriented service data. In *SIGKDD*, KDD '14, pages 861–870, New York, NY, USA, 2014. ACM.

[178] Salvatore Trani, Diego Ceccarelli, Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. Dexter 2.0-an open source tool for semantically enriching data. In *ICWS*, pages 417–420, 2014.

[179] Beth Trushkowsky, Tim Kraska, Michael J. Franklin, and Purnamrita Sarkar. Crowdsourced enumeration queries. In *ICDE*, pages 673–684, 2013.

[180] Twitter Sentiment. `http://www.sananalytics.com/lab/twitter-sentiment/`.

[181] C.J. Van Rijsbergen. *Information retrieval*. Butterworths, 1979.

[182] Matteo Venanzi, John Guiver, Gabriella Kazai, Pushmeet Kohli, and Milad Shokouhi. Community-based bayesian aggregation models for crowdsourcing. In *WWW*, pages 155–164, 2014.

[183] Petros Venetis and Hector Garcia-Molina. Quality control for comparison microtasks. In *Proceedings of the First International Workshop on Crowdsourcing and Data Mining*, pages 15–21. ACM, 2012.

[184] Petros Venetis, Hector Garcia-Molina, Kerui Huang, and Neoklis Polyzotis. Max algorithms in crowdsourcing environments. In *WWW*, pages 989–998, 2012.

[185] Rares Vernica, Michael J Carey, and Chen Li. Efficient parallel set-similarity joins using mapreduce. In *SIGMOD*, pages 495–506. ACM, 2010.

[186] Vasilis Verroios and Hector Garcia-Molina. Entity resolution with crowd errors. In *ICDE*, pages 219–230, 2015.

[187] Vasilis Verroios, Hector Garcia-Molina, and Yannis Papakonstantinou. Waldo: An adaptive human interface for crowd entity resolution. In *SIGMOD*, SIGMOD '17, pages 1133–1148, New York, NY, USA, 2017. ACM.

[188] Vasilis Verroios, Peter Lofgren, and Hector Garcia-Molina. tdp: An optimal-latency budget allocation strategy for crowdsourced MAXIMUM operations. In *SIGMOD*, pages 1047–1062, 2015.

[189] Norases Vesdapunt, Kedar Bellare, and Nilesh N. Dalvi. Crowdsourcing algorithms for entity resolution. *PVLDB*, 7(12):1071–1082, 2014.

[190] Luis Von Ahn, Benjamin Maurer, Colin McMillen, David Abraham, and Manuel Blum. recaptcha: Human-based character recognition via web security measures. *Science*, 321(5895):1465–1468, 2008.

[191] Martin J. Wainwright and Michael I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008.

[192] Jiannan Wang, Tim Kraska, Michael J. Franklin, and Jianhua Feng. CrowdER: crowdsourcing entity resolution. *PVLDB*, 5(11):1483–1494, 2012.

[193] Jiannan Wang, Sanjay Krishnan, Michael J. Franklin, Ken Goldberg, Tim Kraska, and Tova Milo. A sample-and-clean framework for fast and accurate query processing on dirty data. In *SIGMOD*, pages 469–480, 2014.

[194] Jiannan Wang, Guoliang Li, and Jianhua Feng. Can we beat the prefix filtering?: an adaptive framework for similarity join and search. In *SIGMOD*, pages 85–96, 2012.

[195] Jiannan Wang, Guoliang Li, Tim Kraska, Michael J. Franklin, and Jianhua Feng. Leveraging transitive relations for crowdsourced joins. In *SIGMOD*, pages 229–240, 2013.

[196] Sibo Wang, Xiaokui Xiao, and Chun-Hee Lee. Crowd-based deduplication: An adaptive approach. In *SIGMOD*, pages 1263–1277, 2015.

[197] Peter Welinder, Steve Branson, Pietro Perona, and Serge J Belongie. The multidimensional wisdom of crowds. In *NIPS*, pages 2424–2432, 2010.

[198] Peter Welinder and Pietro Perona. Online crowdsourcing: rating annotators and obtaining cost-effective labels. In *CVPR Workshop (ACVHL)*, pages 25–32. IEEE, 2010.

[199] Steven Euijong Whang, Peter Lofgren, and Hector Garcia-Molina. Question selection for crowd entity resolution. *PVLDB*, 6(6):349–360, 2013.

[200] Jacob Whitehill, Paul Ruvolo, Tingfan Wu, Jacob Bergsma, and Javier R. Movellan. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *NIPS*, pages 2035–2043, 2009.

[201] Wikipedia Topic Classification. `https://en.wikipedia.org/wiki/Category:Main_topic_classifications`.

[202] word2vec. `https://code.google.com/p/word2vec/`.

[203] Sai Wu, Xiaoli Wang, Sheng Wang, Zhenjie Zhang, and Anthony K. H. Tung. K-anonymity for crowdsourcing database. *TKDE*, 26(9):2207–2221, 2014.

[204] Yahoo Answers. `https://answers.yahoo.com/dir/index`.

[205] Tingxin Yan, Vikas Kumar, and Deepak Ganesan. Crowdsearch: exploiting crowds for accurate real-time image search on mobile phones. In *MobiSys*, pages 77–90, 2010.

[206] Yan Yan, Glenn M Fung, Rómer Rosales, and Jennifer G Dy. Active learning from crowds. In *ICML*, pages 1161–1168, 2011.

[207] Yan Yan, Rómer Rosales, Glenn Fung, Mark W Schmidt, Gerardo H Valadez, Luca Bogoni, Linda Moy, and Jennifer G Dy. Modeling annotator expertise: Learning when everybody knows a bit of something. In *AISTATS*, pages 932–939, 2010.

[208] Liu Yang, Minghui Qiu, Swapna Gottipati, Feida Zhu, Jing Jiang, Huiping Sun, and Zhong Chen. Cqarank: jointly model topics and expertise in community question answering. In *CIKM*, pages 99–108, 2013.

[209] Peng Ye, UMD EDU, and David Doermann. Combining preference and absolute judgements in a crowd-sourced setting. In *ICML Workshop*, 2013.

[210] Ying Yu, Witold Pedrycz, and Duoqian Miao. Multi-label classification by exploiting label correlations. *Expert Systems with Applications*, 2014.

[211] Chen Jason Zhang, Lei Chen, H. V. Jagadish, and Caleb Chen Cao. Reducing uncertainty of schema matching via crowdsourcing. *PVLDB*, 6(9):757–768, 2013.

[212] Chen Jason Zhang, Yongxin Tong, and Lei Chen. Where to: Crowd-aided path selection. *PVLDB*, 7(14):2005–2016, 2014.

[213] Min-Ling Zhang and Kun Zhang. Multi-label learning by exploiting label dependency. In *KDD*, pages 999–1008. ACM, 2010.

[214] Xiaohang Zhang, Guoliang Li, and Jianhua Feng. Crowdsourced top-k algorithms: An experimental evaluation. *PVLDB*, 9(8):612–623, 2016.

[215] Bo Zhao, Benjamin IP Rubinstein, Jim Gemmell, and Jiawei Han. A bayesian approach to discovering truth from conflicting sources for data integration. *PVLDB*, 5(6):550–561, 2012.

[216] Wayne Xin Zhao, Jing Jiang, Jianshu Weng, Jing He, Ee-Peng Lim, Hongfei Yan, and Xiaoming Li. Comparing twitter and traditional media using topic models. In *ECIR*, pages 338–349. 2011.

[217] Zhou Zhao, Furu Wei, Ming Zhou, Weikeng Chen, and Wilfred Ng. Crowd-selection query processing in crowdsourcing databases: A task-driven approach. In *EDBT*, pages 397–408, 2015.

[218] Zhou Zhao, Da Yan, Wilfred Ng, and Shi Gao. A transfer learning based framework of crowd-selection on twitter. In *SIGKDD*, pages 1514–1517, 2013.

[219] Yudian Zheng, Reynold Cheng, Silviu Maniu, and Luyi Mo. On optimality of jury selection in crowdsourcing. In *EDBT*, pages 193–204, 2015.

[220] Yudian Zheng, Guoliang Li, and Reynold Cheng. DOCS: domain-aware crowdsourcing system. *PVLDB*, 10(4):361–372, 2016.

[221] Yudian Zheng, Guoliang Li, Yuanbing Li, Caihua Shan, and Reynold Cheng. Truth inference in crowdsourcing: Is the problem solved? *PVLDB*, 10(5):541–552, 2017.

[222] Yudian Zheng, Jiannan Wang, Guoliang Li, Reynold Cheng, and Jianhua Feng. Qasca: A quality-aware task assignment system for crowdsourcing applications. In *SIGMOD*, pages 1031–1046, 2015.

[223] Shi Zhi, Bo Zhao, Wenzhu Tong, Jing Gao, Dian Yu, Heng Ji, and Jiawei Han. Modeling truth existence in truth discovery. In *KDD*, pages 1543–1552, 2015.

[224] Jinhong Zhong, Ke Tang, and Zhi-Hua Zhou. Active learning from crowds with unsure option. In *IJCAI*, pages 1061–1068, 2015.

[225] Dengyong Zhou, Qiang Liu, John Platt, and Christopher Meek. Aggregating ordinal labels from crowds by minimax conditional entropy. In *ICML*, pages 262–270, 2014.

[226] Denny Zhou, Sumit Basu, Yi Mao, and John C Platt. Learning from the wisdom of crowds by minimax entropy. In *NIPS*, pages 2195–2203, 2012.

[227] Songchun Zhu, Yu Wu, and David Mumford. Minimax entropy principle and its application to texture modeling. *Neural computation*, 9(8):1627–1660, 1997.

[228] SG Zhukov, VV Chernyshev, EV Babaev, EJ Sonneveld, and H Schenk. Application of simulated annealing approach for structure solution of molecular crystals from x-ray laboratory powder data. *Zeitschrift für Kristallographie/International journal for structural, physical, and chemical aspects of crystalline materials*, 216(1/2001):5–9, 2001.