On Optimality of Jury Selection in Crowdsourcing

Yudian Zheng, Reynold Cheng, Silviu Maniu, and Luyi Mo
Department of Computer Science, University of Hong Kong, Pokfulam Road, Hong Kong
{ydzheng2, ckcheng, smaniu, lymo}@cs.hku.hk

ABSTRACT

Recent advances in crowdsourcing technologies enable computationally challenging tasks (e.g., sentiment analysis and entity resolution) to be performed by Internet workers, driven mainly by monetary incentives. A fundamental question is: how should workers be selected, so that the tasks in hand can be accomplished successfully and economically? In this paper, we study the *Jury Selection Problem* (JSP): Given a monetary budget, and a set of decision-making tasks (e.g., "Is Bill Gates still the CEO of Microsoft now?"), return the set of workers (called *jury*), such that their answers yield the highest "Jury Quality" (or JQ). Existing JSP solutions make use of the *Majority Voting* (MV) strategy, which uses the answer chosen by the largest number of workers. We show that MV does not yield the best solution for JSP. We further prove that among all voting strategies (including deterministic and randomized strategies), *Bayesian Voting* (BV) can optimally solve JSP. We then examine how to solve JSP based on BV. This is technically challenging, since computing the JQ with BV is NP-hard. We solve this problem by proposing an approximate algorithm that is computationally efficient. Our approximate JQ computation algorithm is also highly accurate, and its error is proved to be bounded within 1%. We extend our solution by considering the task owner's "belief" (or *prior*) on the answers of the tasks. Experiments on synthetic and real datasets show that our new approach is consistently better than the best JSP solution known.

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, over 500,000 workers from 190 countries worked on over 250,000 human intelligence tasks (HITs) during August 2012. The large number of workers and HITs have motivated researchers to develop solutions to streamline the crowdsourcing process [6,7,13,22,24,27,37,38].

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 question 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. [7]. 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 expected performance within the budget constraint. The kind of tasks studied in [7] 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 [7,31,35,38] are commonly used in crowdsourcing systems because of their conceptual simplicity. The authors of [7] were the first to propose a system to address JSP for this kind of tasks.

In this paper, we go beyond [7] and perform a comprehensive investigation of this problem. Particularly, we ask the following questions: (1) Is the solution in [7] optimal? (2) If not, what is an optimal solution for JSP? To understand these issues, let us first illustrate how [7] solves JSP.

Figure 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) [7,22,33]. 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. [7] 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 [7].

As illustrated above, MV is used to solve JSP in [7]. In addition

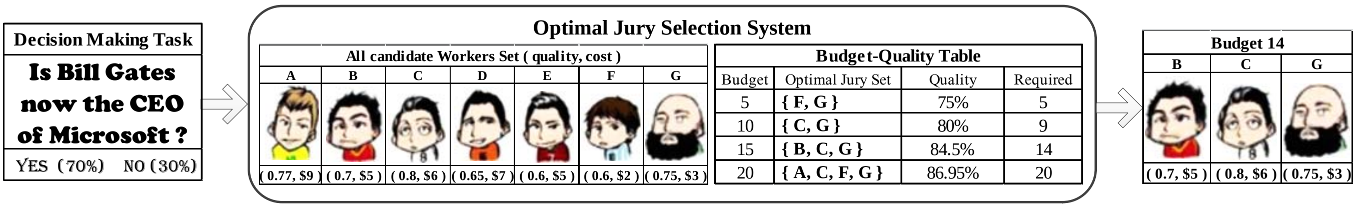


Figure 1: Optimal Jury Selection System.

to MV, researchers have proposed a variety of *voting strategies*, such as Bayesian Voting (BV) [22], Randomized Majority Voting [18], and Random Ballot Voting [29]. 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 [3], based on the workers' votes, and returns the answer having the largest posterior probability.

In this paper, 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 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-based 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 [7].

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 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 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 [7] 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 [22,30,36] are also commonly used in crowdsourcing and several works [1,30,32] 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 paper is arranged as follows. We describe the data model and the problem definition in Section 2. In Section 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 and develop fast solutions to solve JSP in Section 5. In Section 6, we present our experimental results. We discuss how our solutions can be extended for other task types and worker models in Section 7. In Section 8, we review the related works and Section 9 concludes the paper.

2. DATA MODEL & PROBLEM DEFINITION

We now describe our data model in Section 2.1 and define the jury selection problem in Section 2.2.

2.1 Data Model

In this paper, 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 $W = \{j_1, j_2, \ldots, j_N\}$, i.e., $J \subseteq W$, $|J| = n$. Without loss of generality, let $J = \{j_1, j_2, \ldots, j_n\}$. 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 $V = \{v_1, v_2, \ldots, v_n\}$ 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 [7,16,22,30].

We follow the worker model in previous works [7,22,38], 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 [7,22,33] 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 [7], we assume that they are known in advance.

We remark that the optimality of JSP and our solutions can be extended to address other task types and worker models used in [1,22,30,30,32,36]. We will briefly discuss these extensions in Section 7.

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 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 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

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

$$J^* = \operatorname*{arg\,max}_{J \in \mathcal{C}} \max_{S \in \Theta} JQ(J, S, \alpha) \tag{2}$$

Note that existing work [7] only focuses on majority voting strategy (MV) and solves $\arg\max_{J \in \mathcal{C}} JQ(J, MV, 0.5)$, which, as we shall prove later, is sub-optimal for JSP.

In the rest of the paper, 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 3). We then talk about the computation of $JQ(J, S^*, \alpha)$ (Section 4) and finally address of problem of finding J^* (Section 5).

Table 1 summarizes the symbols used in this paper.

3. OPTIMAL VOTING STRATEGY

In this section, we present a detailed description for the voting strategy in Section 3.1. We then formally define JQ in Section 3.2. Finally, we give an optimal voting strategy with respect to JQ in Section 3.3.

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,

Table 1: Table of Symbols

Symbol	Description
$\mathbf{t}$	the ground truth for a task, and $\mathbf{t} \in \{0, 1\}$
α	prior given by the task provider, and $\alpha = P(\mathbf{t} = 0)$
W	a set of all candidate workers $W = \{j_1, j_2, \ldots, j_N\}$
J	a jury, $J \subseteq W$ and $\|J\| = n$, $J = \{j_1, j_2, \ldots, j_n\}$
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

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

DEFINITION 2. *A randomized voting strategy $S(V, J, \alpha)$ returns the result as 0 with probability p and 1 with probability $1 - p$.*

EXAMPLE 1. *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 [21]. And thus, they are widely used when the worst-case performance is the main concern.

Table 2: Classification of Voting Strategies

Deterministic Voting Strategies	Randomized Voting Strategies
Majority Voting (MV) [7]	Randomized Majority Voting (RMV) [18]
Half Voting [25]	Random Ballot Voting [29]
Bayesian Voting [22]	Triadic Consensus [2]
Weighted MV [21]	Randomized Weighted MV [21]
...	...

Table 2 shows a few voting strategies, which are introduced in previous works, and their corresponding category.

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.,

$$JQ(J, S, \alpha) = \Pr(S(\mathbf{V}, J, \alpha) = \mathbf{t}) \tag{3}$$

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 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*

$$\begin{aligned} &\alpha \cdot \textstyle\sum_{V\in\Omega} \Pr(\mathbf{V} = V \mid \mathbf{t} = 0) \cdot \mathbb{E}[\mathbb{1}_{\{S(V)=0\}}] \\ &+ (1-\alpha) \cdot \textstyle\sum_{V\in\Omega} \Pr(\mathbf{V} = V \mid \mathbf{t} = 1) \cdot \mathbb{E}[\mathbb{1}_{\{S(V)=1\}}]. \end{aligned} \tag{4}$$

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 2).

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

EXAMPLE 2. *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 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 $JQ(J, MV, \alpha) = 79.2\%$.*

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, 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 2).

To answer this question, let us reconsider Definition 3 and Equation 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

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

No.	V	P(t = 0)*P(**V** = V | **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 | **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 2: Example of JQ computation for MV and BV ($\alpha = 0.5$, quality of workers $= 0.9, 0.6, 0.6$)

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 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 [10]. 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 1. Next, we give a formal definition for Bayesian Voting (BV) and summarize the above observation in Theorem 1.

DEFINITION 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 1. *BV is optimal with respect to JQ, i.e., $S^* = BV$.*

Note that the BV is also used in [1,16,22]. In the rest of the paper, 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 3) is adopted to measure the goodness of a voting strategy.

EXAMPLE 3. *Let us reconsider Figure 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 2, 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 2 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 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 1, we have $BV(V', J', \alpha) = BV(V, J, \alpha)$. $\square$

A direct consequence of Lemma 1 is as below.

COROLLARY 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 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
$\Pr(\mathbf{V} = V, \mathbf{t} = 1 \mid J) = \Pr(\mathbf{V} = V', \mathbf{t} = 1 \mid J')$. [1] As the mapping $V \to V'$ defines a one-to-one correspondence between Ω and Ω, thus $JQ(J, BV, \alpha) =$

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

$\square$

Corollary 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. 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 4.1). To alleviate this, we propose an efficient approximation algorithm with theoretical bounds to compute JQ for BV in this section.

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

Note that [7] 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 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 [28] (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 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

[1] 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.

$J' = J \cup \{j_{n+1}\}$, 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 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

$$0 \leq g(V) = \sum\nolimits_{i=1}^{n} \left[\, (1-2v_i) \cdot \sigma(q_i) \, \right] < \sigma(q_{n+1}). \quad (5)$$

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

$$G(J) = \min_{d_i = \{-1,+1\}} |\, \sum\nolimits_{i=1}^{n} \; d_i \cdot \sigma(q_i) \,| < \sigma(q_{n+1}). \quad (6)$$

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

$$G(J) = \min_{d_i \in \{-1,+1\}} |\, \sum\nolimits_{i=1}^{n} d_i \cdot w_i \,| < 0.5$$

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., $\min_{d_i \in \{-1,+1\}} |\, \sum_{i=1}^{n} d_i \cdot w_i \,| = 0$, 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.

□

To avoid this hardness result, we propose an approximation algorithm. We first discuss the computation of $JQ(J, BV, 0.5)$ in Section 4.2 and 4.3, and give its approximation error bound in Section 4.4. Finally, we briefly discuss how to adapt the algorithm to $\alpha \in [0, 1]$ in Section 4.5.

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 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

$$\begin{aligned} JQ(J, BV, 0.5) &= \sum\nolimits_{V \in \Omega} [\, A_0(V) + A_1(V) \,] \\ &= \sum\nolimits_{V \in \Omega} [\, A_0(V) + A_1(\overline{V}) \,], \end{aligned} \quad (7)$$

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.

$$u(V) = \ln \Pr(V|\mathbf{t} = 0) = \sum\nolimits_{i=1}^{n} [\, (1 - v_i) \ln q_i + v_i \ln(1 - q_i) \,],$$
$$w(V) = \ln \Pr(V|\mathbf{t} = 1) = \sum\nolimits_{i=1}^{n} [\, v_i \ln q_i + (1 - v_i) \ln(1 - q_i) \,],$$

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

$$R(V) = \sum_{i=1}^{n} [\, (1-2v_i) \cdot \sigma(q_i) \,], \;\; e^{u(V)} = \prod_{i=1}^{n} q_i^{(1-v_i)} \cdot (1-q_i)^{v_i}. \quad (8)$$

As illustrated in Figure 3, we can express $A_0(V) + A_1(\overline{V})$ based

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

Figure 3: Expressing $A_0(V) + A_1(\overline{V})$ using $R(V)$ and $u(V)$

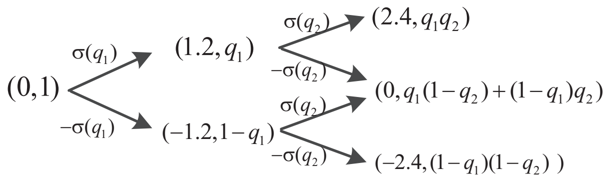


Figure 4: Illustration of the iterative approach { pair $(key, prob)$ }

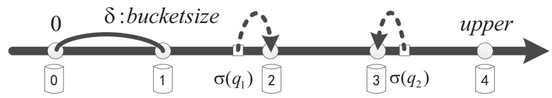


Figure 5: Principle of the bucket array.

on the sign of $R(V)$ and the value of $u(V)$. [2] And therefore, $JQ(J, BV, 0.5) = \sum_{V\in\Omega}[\mathbb{1}_{\{R(V)>0\}} \cdot e^{u(V)} + \mathbb{1}_{\{R(V)=0\}} \cdot \frac{e^{u(V)}}{2}]$.

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

$$prob = \sum\nolimits_{R(V^k)=key\ \wedge\ V^k\in\{0,1\}^k} e^{u(V^k)}. \tag{9}$$

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 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 $prob \cdot (1 - q_{k+1})$ is added to the prob of the new key.

EXAMPLE 4. *We give an example to illustrate the above process in Figure 4, where $n = 2$ and $\sigma(q_1) = \sigma(q_2) = 1.2$. 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.*

4.3 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 bucketsize (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.

We detail this process in Algorithm 2. 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

---

Algorithm 1 `GetBucketArray`

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

$b = [\,0, 0 \cdots 0\,]$; // n elements, all 0
$upper = \max_{i\in[1,n]} \sigma(q_i)$; // compute $upper$
$\delta = \frac{upper}{numBuckets}$; // bucketsize
for $i = 1$ to n **do**
  $b[\,i\,] = \left\lceil \frac{\sigma(q_i)}{\delta} - \frac{1}{2} \right\rceil$; // or b_i in our explanation
return b;

$[0, upper]$ 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 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)[3] 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 3.

Pruning Techniques. We can further improve the running time of the approximation algorithm by applying some pruning techniques in Algorithm 3, 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 further increase the efficiency, in Algorithm 3 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 Approximation Error Bound

Let $\widehat{JQ}$ denotes the estimated value returned by Algorithm 2, and JQ denotes the real Jury Quality. We evaluate the *additive error* bound on $|JQ - \widehat{JQ}|$ and can prove that:

$$\widehat{JQ} \le JQ \quad \text{and} \quad JQ - \widehat{JQ} < e^{\frac{n\delta}{4}} - 1, \tag{10}$$

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 $A_0(V)$, $A_1(V)$, $u(V)$, $w(V)$

---

[2] 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 1, $BV(V) = BV(\overline{V}) = 0$, so $A_0(V) = 0.5 \cdot e^{u(V)}$ and $A_1(\overline{V}) = 0$.

[3] 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 2 EstimateJQ

```
Input:  J = {j_1, j_2 ··· j_n}, numBuckets , n
Output: \widehat{JQ}
b = GetBucketArray(J, numBuckets, n);
b = Sort(b); // sort in decreasing order, for pruning
J = Sort(J); // sort based on worker quality, similar as above
aggregate = AggregateBucket(b, n); // for pruning
\widehat{JQ} = 0; // estimated JQ
SM[ 0 ] = 1; //initialize a map structure
for i = 1 to n do
    M = map(); //initialize an empty map structure
    for (key, prob) ∈ SM do
        flag, value = Prune(key, prob, aggregate[i]);
        if flag = true then
            \widehat{JQ} += value;
            continue // for pruning
        if key + b[i] ∉ M then
            M[ key + b[i] ] = 0;
        M[ key + b[i] ] += prob · q_i; // for v_i = 0
        if key − b[i] ∉ M then
            M[ key − b[i] ] = 0;
        M[ key − b[i] ] += prob · (1 − q_i); // for v_i = 1
    SM = M;
for (key, prob) ∈ SM do
    if key > 0 then
        \widehat{JQ} += prob;
    if key = 0 then
        \widehat{JQ} += 0.5 · prob;
return \widehat{JQ};
```

Algorithm 3 Pruning Techniques

```
def AggregateBucket(b, n):
    aggregate = [ 0, 0 ··· 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];
    return aggregate

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

and $R(V)$ in Section 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 6. It is easy to verify that if we denote $\widetilde{\Omega} = \{0\}\times\{0,1\}^{n-1}$ ($|\widetilde{\Omega}| = 2^{(n-1)}$), then

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

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 6 we can derive that $T(V) = \max\{e^{u(V)}, e^{w(V)}\}$. We can also get

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

Recall Algorithm 2, 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

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

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

Figure 6: $A_0(V) + A_1(V) + A_0(\overline{V}) + A_1(\overline{V})$**.**

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

Figure 7: $\widehat{A_0}(V) + \widehat{A_1}(V) + \widehat{A_0}(\overline{V}) + \widehat{A_1}(\overline{V})$**.**

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 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 ($e^{u(V)}$, $\frac{e^{u(V)}+e^{w(V)}}{2}$, or $e^{w(V)}$) as Figure 7 shows, which is the same as Figure 6, as Algorithm 2 only makes concessions to the key by approximating $R(V)$ as $\widehat{R}(V)$. Then the estimated

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

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 6 shows; while if the estimated $\widehat{R}(V) \le 0$, say $\widehat{R}(V) = 0$, then $\widehat{T}(V) = \frac{e^{u(V)}+e^{w(V)}}{2} < e^{u(V)}$ as Figure 7 shows. Since $T(V) = \max\{e^{u(V)}, e^{w(V)}\}$, then $\widehat{T}(V) \le T(V)$, thus $\widehat{JQ}(J, BV, 0.5) \le JQ(J, BV, 0.5)$.

Generally, we can derive that
(1) $\widehat{T}(V) \le T(V)$ and $\widehat{JQ}(J, BV, 0.5) \le JQ(J, BV, 0.5)$;
(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 $|e^{u(V)} - e^{w(V)}|$.

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

$$\begin{aligned} |R(V) - \widehat{R}(V)| &= |\sum\nolimits_{i=1}^{n} (1-2v_i)\cdot(\sigma(q_i) - \delta b_i)\,| \\ &\le \sum\nolimits_{i=1}^{n} |\,(1-2v_i)\cdot(\sigma(q_i) - \delta b_i)\,| \\ &= \sum\nolimits_{i=1}^{n} |\,\sigma(q_i) - \delta b_i\,| \le \frac{n\delta}{2}, \end{aligned}$$

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 $|R(V)| = |u(V) - w(V)| \le \frac{n\delta}{2}$. Note that we also have $u(V) + w(V) = \sum_{i=1}^{n} [\, \ln q_i + \ln(1-q_i)\,] = \sum_{i=1}^{n} \ln [\, q_i\cdot(1-q_i)\,] \le -n\ln 4$.

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

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

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

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

then we get

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

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

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

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

$$JQ(J, BV, 0.5) - \widehat{JQ}(J, BV, 0.5)$$
$$< \sum\nolimits_{V \in \widetilde{\Omega}} \frac{1}{2^{n-1}} \cdot (e^{\frac{n\delta}{4}} - 1) = e^{\frac{n\delta}{4}} - 1.$$

□

We next show that the bound is very small ($< 1\%$ by setting $d \ge 200$) 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 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 11, we have $JQ - \widehat{JQ} < e^{\frac{5}{4 \cdot d}} - 1$. By setting $d \ge 200$, the bound is $JQ - \widehat{JQ} < 0.627\% < 1\%$.

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 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 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

$$\sum_{V \in \Omega} [\alpha \cdot P(V|\mathbf{t} = 0) \cdot \mathbb{1}_{\{BV(V)=0\}} + (1-\alpha) \cdot P(V|\mathbf{t} = 1) \cdot \mathbb{1}_{\{BV(V)=1\}}]$$
$$= \sum_{V \in \widehat{\Omega}} 0.5 \cdot [\, P(V|\mathbf{t} = 0) \cdot \mathbb{1}_{\{BV(V)=0\}} + P(V|\mathbf{t} = 1) \cdot \mathbb{1}_{\{BV(V)=1\}}]. \quad (11)$$

To prove Equation 11, for a $V \in \Omega$, we define its two mapping counterparts in $\widehat{\Omega}$ as follows:
(1) $\widehat{V} = \{\hat{v}_1, \hat{v}_2, \dots \hat{v}_{n+1}\}$ where $\hat{v}_i = v_i$ for $i \in [1, n]$ and $\hat{v}_{n+1} = 0$;
(2) $\widehat{V}' = \{\hat{v}'_1, \hat{v}'_2, \dots \hat{v}'_{n+1}\}$ 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

$$\alpha \cdot P(V|\mathbf{t} = 0) \cdot \mathbb{1}_{\{BV(V)=0\}} + (1-\alpha) \cdot P(V|\mathbf{t} = 1) \cdot \mathbb{1}_{\{BV(V)=1\}}$$
$$= 0.5 \cdot [\, P(\widehat{V}|\mathbf{t} = 0) \cdot \mathbb{1}_{\{BV(\widehat{V})=0\}} + P(\widehat{V}|\mathbf{t} = 1) \cdot \mathbb{1}_{\{BV(\widehat{V})=1\}} \,] +$$
$$0.5 \cdot [\, P(\widehat{V}'|\mathbf{t} = 0) \cdot \mathbb{1}_{\{BV(\widehat{V}')=0\}} + P(\widehat{V}'|\mathbf{t} = 1) \cdot \mathbb{1}_{\{BV(\widehat{V}')=1\}} \,], \quad (12)$$

then Equation 11 can be proved. As the two sides of Equation 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 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 $\widetilde{V} = \{\tilde{v}_1, \tilde{v}_2, \dots, \tilde{v}_{n+1}\} \in \widehat{\Omega}$: 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 12 is correct.

For a specific $V \in \Omega$, we prove the correctness of Equation 12 by considering two cases:
(1) $\alpha \cdot P(V|\mathbf{t} = 0) \ne (1-\alpha) \cdot P(V|\mathbf{t} = 1)$, 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 12 by proving the following two equations (Equation 13 and 14):

$$\alpha \cdot P(V|\mathbf{t} = 0) \cdot \mathbb{1}_{\{BV(V)=0\}}$$
$$= 0.5 \cdot P(\widehat{V}|\mathbf{t} = 0) \cdot \mathbb{1}_{\{BV(\widehat{V})=0\}} + 0.5 \cdot P(\widehat{V}'|\mathbf{t} = 1) \cdot \mathbb{1}_{\{BV(\widehat{V}')=1\}}, \quad (13)$$

$$(1-\alpha) \cdot P(V|\mathbf{t} = 1) \cdot \mathbb{1}_{\{BV(V)=1\}}$$
$$= 0.5 \cdot P(\widehat{V}|\mathbf{t} = 1) \cdot \mathbb{1}_{\{BV(\widehat{V})=1\}} + 0.5 \cdot P(\widehat{V}'|\mathbf{t} = 0) \cdot \mathbb{1}_{\{BV(\widehat{V}')=0\}}. \quad (14)$$

Here we only prove Equation 13 and Equation 14 can be proved similarly. Equation 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 $P(\widehat{V}'|\mathbf{t} = 1)$

$$= P(\hat{v}'_{n+1} = 1|\mathbf{t} = 1) \cdot \prod\nolimits_{i=1}^{n} q_i^{\hat{v}'_i} \cdot (1-q_i)^{1-\hat{v}'_i}$$
$$= \alpha \cdot \prod\nolimits_{i=1}^{n} q_i^{(1-v_i)} \cdot (1-q_i)^{v_i} = \alpha \cdot P(V|\mathbf{t} = 0)$$

, 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 13, and Equation 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 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 1 we know that all $BV(V) = BV(\widehat{V}) = BV(\widehat{V}') = 0$. Thus we can directly prove Equation 12 by taking the above derived formulas into it.

Then we have proved Equation 12 for the two cases, thus Equation 11, or the theorem can be proved. □

Thus we can use Algorithm 2 for any prior α, by adding to the jury a pseudo-worker of quality α. Moreover, the approximation error bound proved in Section 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 $d \geq 200$.

5. JURY SELECTION PROBLEM (JSP)

Now we focus on addressing $J^* = \arg\max_{J\in\mathcal{C}} JQ(J, BV, \alpha)$, 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 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 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

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

So if we can prove the following inequality

$$\begin{aligned}&\max\{\ \alpha\cdot P(V^{(0)} \mid \mathbf{t}=0)\ ,\ (1-\alpha)\cdot P(V^{(0)} \mid \mathbf{t}=1)\ \} + \\ &\max\{\ \alpha\cdot P(V^{(1)} \mid \mathbf{t}=0)\ ,\ (1-\alpha)\cdot P(V^{(1)} \mid \mathbf{t}=1)\ \} \\ &\geq\ \max\{\ \alpha\cdot P(V \mid \mathbf{t}=0)\ ,\ (1-\alpha)\cdot P(V \mid \mathbf{t}=1)\ \},\end{aligned} \tag{15}$$

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 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 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 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 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 2, recall the definition of b and r, we know that if we can prove the following inequality

$$\begin{aligned}&\max\{\ b\cdot q'_{n+1}, r\cdot(1-q'_{n+1})\ \} + \max\{\ b\cdot(1-q'_{n+1}), r\cdot q'_{n+1}\ \} \\ \geq &\max\{\ b\cdot q_{n+1}, r\cdot(1-q_{n+1})\ \} + \max\{\ b\cdot(1-q_{n+1}), r\cdot q_{n+1}\ \},\end{aligned} \tag{16}$$

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. □

Lemma 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. 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 2, the problem still remains NP-hard, by reduction to a n-th order Knapsack Problem [7].

THEOREM 4. *Solving JSP is NP-hard.*

PROOF. For ease of presentation, we denote $X = [x_1, x_2, \ldots, x_N]$ 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 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.[4]

We next prove that JSP is NP-hard by constructing a special case, that is given α, $J = \{j_1, j_2, \ldots, j_n\}$ 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 $C' = \{c'_1, c'_2, \ldots, c'_{n+1}\}$ 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

$$X^* = \underset{X \in \{X^{(1)},\, X^{(2)}\}}{\arg\max} \; \{\, F(X^{(1)}, W'),\; F(X^{(2)}, W') \,\},$$

where $X^{(1)}$ and $X^{(2)}$ are
(i) $X^{(1)} = [x_1^{(1)}, x_2^{(1)}, \ldots, x_{n+1}^{(1)}]$ where $x_i^{(1)} = 1$ for $i \in [1, n]$ and $x_{n+1}^{(1)} = 0$;
(ii) $X^{(2)} = [x_1^{(2)}, x_2^{(2)}, \ldots, x_{n+1}^{(2)}]$ 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 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 compare between $F(X^{(1)}, W') = JQ(J, BV, \alpha)$ and $F(X^{(2)}, W') = K$.
Then we have
(1) $X^* = X^{(1)} \Rightarrow JQ(J, BV, \alpha) \ge K$;
(2) $X^* = X^{(2)} \Rightarrow K \ge JQ(J, BV, \alpha)$.

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

5.1 Heuristic Solution

To address the computational hardness issue, we use the *simulated annealing heuristic* [17], 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 [5,9,40].

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., $\Delta \ge 0$), 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 distribution [19].

[4]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 shows.

Algorithm 4 JSP

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** $T \ge \epsilon$ **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 $M + c_r \le B$ **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:   $T = T/2$; // cool the temperature
15: **return** $\widehat{J}$;

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 4 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 4: 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 5. 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 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 $JQ(J^*, BV, \alpha)$).

6. EXPERIMENTAL EVALUATION

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

Algorithm 5 `Swap`

Input: $X,\ M,\ \widehat{J},\ H,\ r,\ B,\ N$
Output: $X,\ M,\ \widehat{J},\ H$

if $x_r = 0$ **then**
  randomly pick an index $k \in H$;
  $a = k$; $b = r$; // store the index
else
  randomly pick an index $k \in \{1, 2, \ldots, N\} \backslash H$;
  $a = r$; $b = k$; // store the index
if $M - c_a + c_b \le B$ **then**
  $\Delta = \texttt{EstimateJQ}(\ \widehat{J} \setminus \{j_a\} \cup \{j_b\}\ ) - \texttt{EstimateJQ}(\widehat{J})$;
  if $\Delta \ge 0$ **or** $random(0,1) \le \exp(-\frac{\Delta}{T})$ **then**
    $x_a = 0;\ x_b = 1;\ \ M = M - c_a + c_b$;
    $\widehat{J} = \widehat{J} \setminus \{j_a\} \cup \{j_b\};\ \ H = H \backslash \{a\} \cup \{b\}$;
return $X,\ M,\ \widehat{J},\ H$

Python 2.7 and evaluated on a 16GB memory machine with Windows 7 64bit.

6.1 Synthetic Dataset

6.1.1 Setup

First, we describe our default settings for the experiments. Similar to the settings in [7], 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 [7], 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 4), we set $\epsilon = 10^{-8}$; for JQ computation (Algorithm 2), we set $numBuckets = 50$. To achieve statistical significance of our results, we repeat the results 1,000 times and report the average values.

6.1.2 System Comparison

We first perform the comparison of JSP with previous works, in an end-to-end system experiment. Cao et al. [7] 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 $\arg\max_{J \in \mathcal{C}} JQ(J, MV, 0.5)$. We denote their system as *MVJS* (Majority Voting Jury Selection System) and our system (Figure 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 8. We first evaluate the performance of the two systems by varying $\mu \in [0.5, 1]$ in Figure 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 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 8(b), *OPTJS* on average leads around 3% compared with *MVJS* for different B; in Figure 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 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 variance.

6.1.3 Evaluating OPTJS

Next, we test the approximation error of Algorithm 4 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 9(a). It shows that the two curves almost coincide with each other. As mentioned in Section 6.1.1, 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 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 4. We set $B = 0.5$ and vary $N \in [100, 500]$. The results are shown in Figure 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 ($N = 500$).

Table 3: Counts in different error ranges

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

6.1.4 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[5] and RMV-Randomized Majority Voting). We vary $\mu \in [0.5, 1]$ and illustrate the resulting JQ in Figure 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, performs not better than MV for $\mu \ge 0.5$, as randomized strategies may improve the error bound in the worst case [21]. 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 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 11. We first evaluate the effect of the quality variance σ^2 with varying mean μ in Figure 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 2 for approximating the real JQ. We first evaluate the approximation error in Figure 11(b) by varying $numBuckets \in [10, 200]$. As can be seen, the approximation error drops significantly with $numBuckets$, and is very close to 0 if we have enough buckets. In Figure 11(c) we plot the histogram of differences between the accurate JQ and the

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

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 2 by varying the number of workers $n \in [100, 500]$ in Figure 11(d). The pruning technique is indeed effective, saving more than half the computational cost. Moreover, it 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 the proposed pruning methods.

6.2 Real Dataset

6.2.1 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[6], 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 0.6.

6.2.2 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 further recorded as a point in Figures 12(a)-(c), respectively. It can be seen that Figure 12(a)-(c) has a similar results pattern as Figure 8(b)-(d), i.e., experimental results on the synthetic datasets. Especially, *OPTJS* always outperforms *MVJS* in real-world scenarios.

6.2.3 Is JQ is a good prediction?

Finally, we try to evaluate whether JQ, defined in Definition 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 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.

7. EXTENSIONS TO VARIOUS TASK TYPES AND WORKER MODELS

In Section 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 [22] 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 [26], which decomposes each task into ℓ decision making tasks, and publish these ℓ tasks to workers.

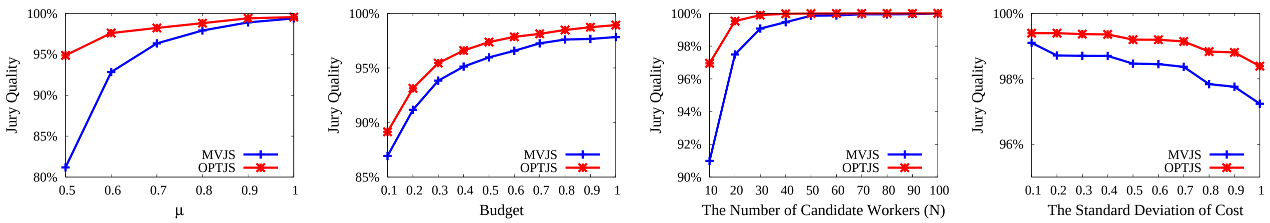
In the case of worker quality model, it can be modeled by measuring the sensitivity and specificity of each worker [39], and it can be generalized to be modeled as a *confusion matrix* (CM) [16]. A confusion matrix C is a matrix of size $\ell \times \ell$ where each element C_{jk} encodes the probability 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 [22], which models each worker as a constant parameter $q \in [0, 1]$ for multiple label tasks. Many studies [1,16,39] 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 7.1, and then extend the JQ and JSP respectively in Section 7.2 and 7.3.

7.1 Optimal Strategy Extension

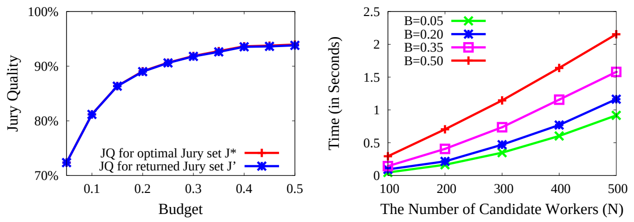
To derive the optimal strategy for multiple-choice tasks, note that here $S(\cdot)$ is a function which takes V ($\in \Omega = \{0, 1, \ldots, \ell-1\}^n$), jury set J (where each worker j_i is modeled as a CM $C^{(i)}$)

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



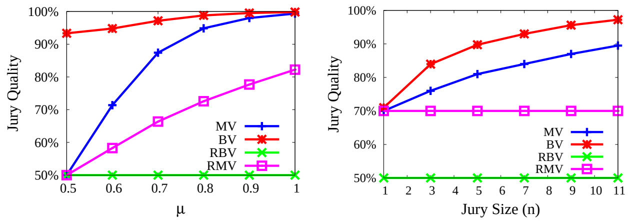
(a) Varying μ (b) Varying B (c) Varying N (d) Varying $\widehat{\sigma}$

Figure 8: End-to-end system comparison



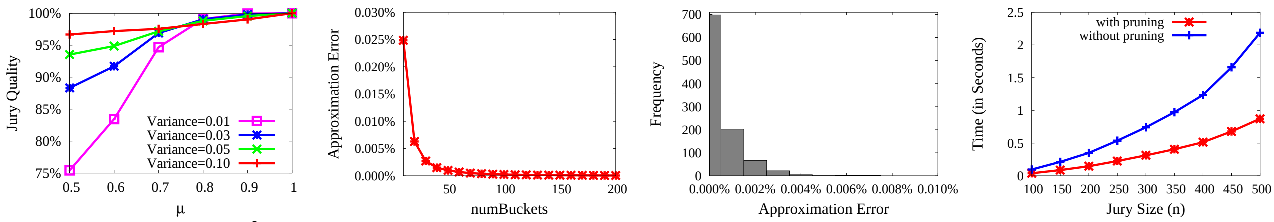
(a) Approximation Error (b) Varying N

Figure 9: Efficiency & Effectiveness of *OPTJS*



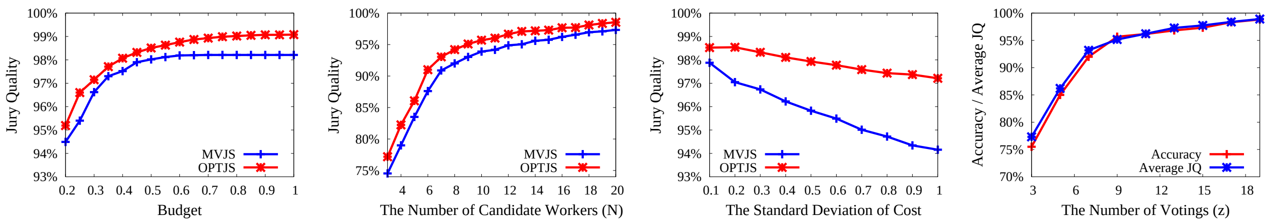
(a) Varying μ (b) Varying n

Figure 10: JQ for different strategies



(a) Varying μ and σ^2 (b) Varying $numBuckets$ (c) Approximation Error (d) Varying n

Figure 11: $JQ(J, BV, 0.5)$ **computation**



(a) Varying B (b) Varying N (c) Varying $\widehat{\sigma}$ (d) Is JQ a good prediction?

Figure 12: Real dataset evaluation

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, 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

$$S^*(V) = \underset{t\in\{0,1,\ldots,\ell-1\}}{\arg\max}\ \alpha_t\cdot\prod_{i=1}^{n} C_{t,v_i}^{(i)}. \tag{17}$$

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 $S^* = BV$.

7.2 JQ Computation Extension

Algorithm 6 `GetBucketSize`

Input: $J = \{j_1, j_2, \ldots, j_n\}$, $numBuckets$, n, ℓ, $\vec{\alpha}$
Output: δ

$upper = 0$;
$tmin = 1$; $tmax = 0$;
for $t = 0$ to $\ell - 1$ **do**
  if $tmin > \alpha_t$ **then** $tmin = \alpha_t$;
  if $tmax < \alpha_t$ **then** $tmax = \alpha_t$;
if $upper < \ln \frac{tmax}{tmin}$ **then** $upper = \ln \frac{tmax}{tmin}$; // prior
for $i = 1$ to n **do**
  $tmin = 1$; $tmax = 0$;
  for $t = 0$ to $\ell - 1$ **do**
    for $j = 0$ to $\ell - 1$ **do**
      if $tmin > C^{(i)}_{t,j}$ **then** $tmin = C^{(i)}_{t,j}$;
      if $tmax < C^{(i)}_{t,j}$ **then** $tmax = C^{(i)}_{t,j}$;
    if $upper < \ln \frac{tmax}{tmin}$ **then** $upper = \ln \frac{tmax}{tmin}$; // CM
$\delta = \frac{upper}{numBuckets}$;
return δ

Compared with Algorithm 2, the case of multiple label tasks where each worker is modeled as a confusion matrix requires several adaptations. First let us illustrate an example below.

EXAMPLE 5. *Take the example in Figure 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 13(a)-(c). Take the voting* $V = \{0, 1\}$ *as an example in Figure 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 13 illustrated above, dealing with multiple label task while modeling each worker as a CM is different from that in Figure 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 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 2, in Figure 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 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

$$BV(V) = \underset{t \in [0, \ell-1]}{\arg\max}\ \alpha_t \cdot \Pr(V \mid \mathbf{t} = t),$$

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

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

Algorithm 7 `EstimateGeneralJQ`

Input: $J = \{j_1, j_2, \ldots, j_n\}$, $numBuckets$, n, ℓ, $\vec{\alpha}$
Output: $\widehat{JQ}$

$\delta =$ `GetBucketSize`$(Q, numBuckets, n, \ell, \vec{\alpha})$;
$\widehat{JQ} = 0$;
for $t = 0$ to $\ell - 1$ **do**
  $SM = map()$;
  $key = (\,0, 0, \ldots, 0\,)$; // initialize an ℓ-tuple with all 0
  for $j = 0$ to $\ell - 1$ **do**
    $key[\,j + 1\,] = \left\lceil \frac{\ln(\alpha_t/\alpha_j)}{\delta} - \frac{1}{2} \right\rceil$; // incorporate prior
  $SM[\,key\,] = 1$; // the key contains prior information
  for $i = 1$ to n **do**
    $M = map()$; // initialize an empty map structure
    for $(key, prob) \in SM$ **do**
      for $v = 0$ to $\ell - 1$ **do**
        $newkey = key$; // copy from key to $newkey$
        for $jj = 0$ to $\ell - 1$ **do**
          $newkey[\,jj + 1\,] += \left\lceil \frac{\ln(C^{(i)}_{t,v}/C^{(i)}_{jj,v})}{\delta} - \frac{1}{2} \right\rceil$;
        if $newkey \notin M$ **then**
          $M[\,newkey\,] = 0$;
        $M[\,newkey\,] = M[\,newkey\,] + prob \cdot C^{(i)}_{t,v}$;
    $SM = M$;
  $\widehat{JQsub} = 0$;
  for $(key, prob) \in SM$ **do**
    $count = 0$;
    $flag =$ **true**;
    for $j = 0$ to $\ell - 1$ **do**
      if $key[\,j + 1\,] < 0$ **then**
        $flag =$ **false**;
        break;
      else if $key[\,j + 1\,] == 0$ **then**
        $count = count + 1$;
    if $flag =$ **true then**
      $\widehat{JQsub} += prob/count$;
  $\widehat{JQ} = \widehat{JQ} + \widehat{JQsub} \cdot \alpha_t$; // prior
return $\widehat{JQ}$;

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 13: Jury Quality Calculation for different t and V ($\ell = 3,\ J = \{j_1, j_2\},\ \vec{\alpha} = \{\alpha_0 = 1/3, \alpha_1 = 1/3, \alpha_2 = 1/3\}$, qualities $C^{(1)}, C^{(2)}$)

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 7 illustrates the JQ estimation for multiple-label tasks. As discussed above, we take ℓ iterations to consider each possible $\mathbf{t}$ respectively, where 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 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 6, which is the same as follows:

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

Note that as we can easily deal with the case where $\alpha_t = 0$ for some $t \in [0, \ell - 1]$ and $C^{(i)}_{t,j} = 0$ 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 2, 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 7 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.

7.3 JSP extension

In this section, we extend JSP to support for CM and multiple label tasks. Recall that in Section 5 we first study two Lemmas (Lemma 2 and 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 3, that is, the monotonicity property on jury size holds even for more general task and worker model:

LEMMA 4 (EXTENSIONS FOR LEMMA 2). *Given $\vec{\alpha}$ and J, $JQ(J, BV, \vec{\alpha}) \le JQ(J', BV, \vec{\alpha})$ where $J' = J \cup \{j_{n+1}\}$.*

PROOF. Similar to the proof in Lemma 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 17, we can prove that

$$\sum\nolimits_{t=0}^{\ell-1} A_t(V) = \max_{t\in[0,\ell-1]} \{\ \alpha_t \cdot P(V \mid \mathbf{t} = t)\ \}. \tag{18}$$

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 18 becomes

$$\begin{aligned} &\sum\nolimits_{j=0}^{\ell-1} \sum\nolimits_{t=0}^{\ell-1} A_t(V^{(j)}) \\ =& \sum\nolimits_{j=0}^{\ell-1} \max_{t\in[0,\ell-1]} \{\ \alpha_t \cdot P(V^{(j)} \mid \mathbf{t} = t)\ \} \\ =& \sum\nolimits_{j=0}^{\ell-1} \max_{t\in[0,\ell-1]} \{\ \alpha_t \cdot C^{(n+1)}_{t,j} \cdot P(V \mid \mathbf{t} = t)\ \}. \end{aligned} \tag{19}$$

By denoting $b_t = \alpha_t \cdot P(V|\mathbf{t} = t)$ for $t \in [0, \ell - 1]$, if we can prove Equation 19 is not lower than Equation 18, i.e.,

$$\sum\nolimits_{j=0}^{\ell-1} \max_{t\in[0,\ell-1]} \{\ b_t \cdot C^{(n+1)}_{t,j}\ \} \ge \max_{t\in[0,\ell-1]} \{\ b_t\ \}, \tag{20}$$

, 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

$$\sum\nolimits_{j=0}^{\ell-1} \max_{t\in[0,\ell-1]} \{\ b_t \cdot C^{(n+1)}_{t,j}\ \} \ge \sum\nolimits_{j=0}^{\ell-1} b_{t'} \cdot C^{(n+1)}_{t',j} = b_{t'}.$$

By extending to any $t' \in [0, \ell - 1]$, we can finally prove Equation 20, thus proving the Lemma. □

Lemma 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., $B \ge \sum_{i=1}^{N} c_i$).

Even though the extension for Lemma 2 holds, the extension for Lemma 3, i.e., detecting what kind of CM will contribute more to

JQ remains an open question. Previous research [16,30] has addressed how to rank workers (or to detect spammers in all workers) based on their associated Confusion Matrices. The basic idea of [16,30] is to see whether a worker's vote is indicative on the true label or not.

To implement the idea, [16] transforms the "hard" label by a worker to a "soft" label distribution. To be precise, if a worker having CM C votes a label $j \in \{0, 1, \dots, \ell-1\}$, 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 [3], as follows:

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

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, [16] 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 $\vec{\alpha}$:

$$score(\vec{\alpha}, C) = -\sum_{j=0}^{\ell-1} \alpha_j \cdot penalty(p^{(j)}). \tag{21}$$

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 [30] 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:

$$score(C) = \sum_{j=0}^{\ell-1} diff(C, j). \tag{22}$$

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 4 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 4.

8. RELATED WORKS

Crowdsourcing. Nowadays, crowdsourcing has evolved as a problem solving paradigm [6] to address computer-hard tasks. To incorporate the crowd into query processing, crowdsourced databases (e.g., CrowdDB [13], Deco [27], Qurk [24] and CDAS [22]) are built, compared with traditional database systems, they do not hold the closed-world assumption. As a novel paradigm, the power of crowdsourcing has also been leveraged in other applications. For example, in Optical Character Recognition [34], Entity Resolution [35], Collaborative Tagging [37], Schema Matching [15,38], Web Table Understanding [11] and so on.

Voting Strategy. In order to aggregate the collective wisdom of a jury, given some specific voting of a task from the jury, voting strategies are widely used to return a result, which is an estimation of the ground truth for the task. For example, Majority Voting strategy [7] strictly returns the answer corresponding to higher votes, and Random Ballot Voting [29] randomly selects the returned result. Similarly other strategies [2,21,22,25] are also talked about in a great deal. Different from their works, here we give a systematic way to classify all the strategies into two categories, and try to observe the optimal strategy in all these strategies under the Jury Selection Problem. Note that different from our problem, people may evaluate strategies under different purposes. For example, [23] analyzes the optimal Bayesian manipulation strategies by assessing the expected loss in social welfare, and [10] applies Bayesian model to take a game-theoretic approach in characterizing the symmetric equilibrium of the game with juries.

Worker Model. To model a worker's quality in crowdsourcing, most existing works [7,22,25,38] define it as a constant parameter indicating the probability that the worker correctly answers a question, while other work [16] defines it as a Confusion Matrix, which tries to capture relations between labels in questions and is specific to choices in tasks. For the methods to derive worker's quality, a normal way is to leverage the answering history. If they are not sufficient, [22] hides golden questions (questions with known ground truth) and derive the quality based on the worker's answers for them, while other work [16] applies Expectation Maximization [8] algorithm to iteratively updates worker's quality until convergence. For micro-blog services especially in Twitter, the retweet actions are usually explored to derive the error rate for each worker [7]. In our work we define worker's quality by a constant parameter (commonly used by other works) and assume that they are known in advance. Moreover, we also extend our method to address the Confusion Matrix mentioned in [16].

Online Processing. There are also some online processing systems [4,14,22] in crowdsourcing, which addresses how to assign tasks to workers and process the workers' answers. For example, [22] proposes quality-sensitive answering model and terminate assigning questions which has got confident answers; [4] proposes an entropy-like approach to define the uncertainty of each question and assigns questions with highest uncertainty; [14] proposes cost-sensitive model to address which questions are better answered by humans or machines. Different from them, we especially evaluate how to estimate the JQ before the workers are selected to answer the questions, and the quality estimation can provide statistics and guidance for the task publisher to wisely invest budget. Even though existing work [22,25] tried to estimate the quality, their settings not robust (assuming that each worker is of the same quality). And given a budget constraint, we can leverage it well to select the optimal jury set.

Expert Team Formation. In social network, several works [12,20] studied the problem of expert team formation, that is, given the aggregated skill requirements for a task, how to finding 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 [12,20], 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 a similar problem, called Jury Selection Problem, which is firstly proposed by [7]. But we find that the solution is sub-optimal in [7], which cannot leverage the quality for low quality workers. We formally address the optimal JSP problem in the paper.

9. CONCLUSIONS

In this paper, we have studied Jury Selection Problem (JSP) for 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.

10. REFERENCES

[1] A.P.Dawid and A.M.Skene. Maximum likelihood estimation of observer error-rates using em algorithm. *Appl.Statist.*, 28(1):20–28, 1979.

[2] Ashish Goel and David Lee, Triadic Consensus: A Randomized Algorithm for Voting in a Crowd. `http://arxiv.org/pdf/1210.0664v1.pdf`.

[3] Bayes' Theorem. `http://en.wikipedia.org/wiki/Bayes'_theorem`.

[4] R. Boim, O. Greenshpan, T. Milo, S. Novgorodov, N. Polyzotis, and W. C. Tan. Asking the right questions in crowd data sourcing. In *ICDE*, pages 1261–1264, 2012.

[5] D. Bookstaber. Simulated annealing for traveling salesman problem.

[6] D. Braham. Crowdsourcing as a model for problem solving: An introduction and cases. *Convergence*, 14(1):75–90, 2008.

[7] C. C. Cao, J. She, Y. Tong, and L. Chen. Whom to ask? jury selection for decision making tasks on micro-blog services. *PVLDB*, 5(11):1495–1506, 2012.

[8] 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.

[9] A. Drexl. A simulated annealing approach to the multiconstraint zero-one knapsack problem. *Computing*, 40:1–8, 1988.

[10] J. Duggan and C. Martinelli. A bayesian model of voting in juries. *Games and Economic Behavior*, 37(2):259–294, 2001.

[11] J. Fan, M. Lu, B. C. Ooi, W.-C. Tan, and M. Zhang. A hybrid machine-crowdsourcing system for matching web tables. In *ICDE*, pages 976–987, 2014.

[12] F. Farhadi, E. Hoseini, S. Hashemi, and A. Hamzeh. Teamfinder: A co-clustering based framework for finding an effective team of experts in social networks. In *ICDM Workshops*, pages 107–114, 2012.

[13] A. Feng, M. J. Franklin, D. Kossmann, T. Kraska, S. Madden, S. Ramesh, A. Wang, and R. Xin. Crowddb: Query processing with the vldb crowd. *PVLDB*, 4(12):1387–1390, 2011.

[14] J. Gao, X. Liu, B. C. Ooi, H. Wang, and G. Chen. An online cost sensitive decision-making method in crowdsourcing systems. In *SIGMOD Conference*, pages 217–228, 2013.

[15] N. Q. V. Hung, N. T. Tam, Z. Miklós, and K. Aberer. On leveraging crowdsourcing techniques for schema matching networks. In *Database Systems for Advanced Applications*, pages 139–154. Springer, 2013.

[16] P. Ipeirotis, F. Provost, and J. Wang. Quality management on amazon mechanical turk. In *SIGKDD workshop*, pages 64–67, 2010.

[17] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, New Series*, 220(4598):671–680, 1983.

[18] A. Lacasse, F. Laviolette, M. Marchand, and F. Turgeon-Boutin. Learning with randomized majority votes. pages 162–177, 2010.

[19] L. Landau and E. Lifshitz. *Statistical Physics. Course of Theoretical Physics 5 (3 ed.).* Oxford: Pergamon Press, 1980.

[20] T. Lappas, K. Liu, and E. Terzi. Finding a team of experts in social networks. In *KDD*, pages 467–476, 2009.

[21] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Inf. Comput.*, 108(2):212–261, Feb. 1994.

[22] X. Liu, M. Lu, B. C. Ooi, Y. Shen, S. Wu, and M. Zhang. Cdas: A crowdsourcing data analytics system. *PVLDB*, 5(10):1040–1051, 2012.

[23] T. Lu, P. Tang, A. D. Procaccia, and C. Boutilier. Bayesian vote manipulation: Optimal strategies and impact on welfare. In *UAI*, pages 543–553, 2012.

[24] A. Marcus, E. Wu, S. Madden, and R. C. Miller. Crowdsourced databases: Query processing with people. In *CIDR*, pages 211–214, 2011.

[25] L. Mo, R. Cheng, B. Kao, X. S. Yang, C. Ren, S. Lei, D. W. Cheung, and E. Lo. Optimizing plurality for human intelligence tasks. In *CIKM*, pages 1929–1938, 2013.

[26] A. G. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom. Crowdscreen: algorithms for filtering data with humans. In *SIGMOD*, pages 361–372, 2012.

[27] H. Park, R. Pang, A. G. Parameswaran, H. Garcia-Molina, N. Polyzotis, and J. Widom. Deco: A system for declarative crowdsourcing. *PVLDB*, 5(12):1990–1993, 2012.

[28] Partition Problem. `http://en.wikipedia.org/wiki/Partition_problem`.

[29] Random Ballot Voting. `http://en.wikipedia.org/wiki/Random_ballot`.

[30] V. C. Raykar and S. Yu. Eliminating spammers and ranking annotators for crowdsourced labeling tasks. *Journal of Machine Learning Research*, 13:491–518, 2012.

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

[32] M. Venanzi, J. Guiver, G. Kazai, P. Kohli, and M. Shokouhi. Community-based bayesian aggregation models for crowdsourcing. In *Proceedings of the 23rd international conference on World wide web*, pages 155–164. International World Wide Web Conferences Steering Committee, 2014.

[33] P. Venetis and H. Garcia-Molina. Quality control for comparison microtasks. In *CrowdKDD, KDD 2012 Workshop*, 2012.

[34] L. von Ahn, B. Maurer, C. McMilen, D. Abraham, and M. Blum. Recaptcha: Human-based character recognition via web security measures. *Science*, 321(5895):1465–1468, 2008.

[35] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *PVLDB*, 5(11):1483–1494, 2012.

[36] J. Whitehill, P. Ruvolo, T. Wu, J. Bergsma, and J. R. Movellan. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *NIPS*, pages 2035–2043, 2009.

[37] X. S. Yang, R. Cheng, L. Mo, B. Kao, and D. W. Cheung. On incentive-based tagging. In *ICDE*, pages 685–696, 2013.

[38] C. J. Zhang, L. Chen, H. V. Jagadish, and C. C. Cao. Reducing uncertainty of schema matching via crowdsourcing. *PVLDB*, 6(9):757–768, 2013.

[39] B. Zhao, B. I. P. Rubinstein, J. Gemmell, and J. Han. A bayesian approach to discovering truth from conflicting sources for data integration. *PVLDB*, 5(6):550–561, 2012.

[40] S. Zhukov, V. Chernyshev, E. Babaev, E. 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.