Cleaning Uncertain Data with a Noisy Crowd

Chen Jason Zhang Lei Chen Yongxin Tong Zheng Liu
Hong Kong University of Science and Technology, Hong Kong, China
{czhangad, leichen, yxtong, zliual}@cse.ust.hk

***Abstract*—Uncertain data has been emerged as an important problem in database systems due to the imprecise nature of many applications. To handle the uncertainty, probabilistic databases can be used to store uncertain data, and querying facilities are provided to yield answers with confidence. However, the uncertainty may propagate, hence the returned results from a query or mining process may not be useful. In this paper, we leverage the power of crowdsourcing for cleaning uncertain data. Specifically, we will design a set of Human Intelligence Tasks (HIT)s to ask a crowd to improve the quality of uncertain data.**

Each HIT is associated with a cost, thus, we need to design solutions to maximize the data quality with minimal number of HITs. There are two obstacles for this non-trivial optimization - first, the crowd has a probability to return incorrect answers; second, the HITs decomposed from uncertain data are often correlated. These two obstacles lead to very high computational cost for selecting the optimal set of HITs. Thus, in this paper, we have addressed these challenges by designing an effective approximation algorithm and an efficient heuristic solution. To further improve the efficiency, we derive tight lower and upper bounds, which are used for effective filtering and estimation. We have verified the solutions with extensive experiments on both a simulated crowd and a real crowdsourcing platform.

I. Introduction

Uncertain data has been emerged as an important problem in database systems due to the imprecise nature of many applications. In the field of information extraction, some learning-based model (e.g. Conditional Random Field) usually produces a ranked list of extractions, each of which is associated with a probability of correctness [15]. In entity resolution and schema matching, automatic tools may generate multiple results, each of which is associated with a confidence value indicating the matching quality [30], [10]. When data from multiple data sources are integrated, conflicting information may be handled by keeping conflicted data associated with probabilities, a.k.a uncertain/probabilistic data.

Due to the existence of an exponential number of possible worlds in an uncertain database (a possible world is one possible combination of real instances of uncertain data), mining uncertain data is often computationally expensive. Moreover, the uncertainty may propagate, so the returned results from a query may not be useful. Thus, it is desirable to improve the data quality by reducing the uncertainty. In many applications, data becomes uncertain because computers have difficulties to recognize human-intuitive semantics via the given data representation (e.g. images, natural language) or handle human-intrinsic tasks (e.g. extracting the correct information or language translation). Therefore, intuitively, human perception should be very effective in cleaning uncertain data.

Existing works of uncertain data cleaning [5], [23] assume there is a cleaning agent (e.g. an expert) available, which may access and return the 'ground truth' of the uncertain data. However, in many circumstances, such availability does not hold. Along with the emerging of uncertain data, crowdsourcing has become a hot research topic due to the success of several crowdsourcing platforms such as Amazon Mechanical Turk and CrowdFlower. These platforms provide human computational power, which is not only always available, but is at least fairly affordable. In this paper, we leverage the power of crowdsourcing for cleaning uncertain data.

A. Uncertainty Model and Quality Measurement

Among various models of uncertainty, we consider the *x-relation*, which is widely adopted in the field of uncertain database [1], [5], [23], [34], [33]. In the x-relation model, an x-relation contains a set of independent x-tuples. Each x-tuple includes a set of mutually exclusive tuples (or called alternatives), associated with probabilities, which represent a discrete probability distribution of these tuples being correct.

Table I illustrates a probabilistic database, where the uncertainty of each entry is captured by an x-tuple. In particular, each x-tuple represents an address of a facility location, provided by an information extraction model (e.g. CRF or HMM, and the creation of such probabilistic database from the extraction models is detailed in [15]). For instance, the x-tuple T_1 indicates four possible addresses for a location, i.e. $\{t_1,t_2,t_3,t_4\}$, for textual data "51A Hayward East New York"; and the probability of t_1 being correct is 0.4. Such a probabilistic data model well captures the correlation among the attributes (labels). In the example of Table I, the labels 'House no', 'Area' and 'City' are correlated: the marginal probability $Pr(House\ no = 51A) = Pr(t_2) + Pr(t_3) = 0.5$, while conditional probability $Pr(House\ no = 51|Area = East) = 1$ and $Pr(House\ no = 51|Area = Hayward\ East) = 0$.

To measure the data quality of an x-tuple, we follow the proposal in [5], [23], [4] and use the ***negative*** *value of the Shannon entropy*, i.e. $\sum_p p \log p$. For instance, the quality of T_1 in Table I, denoted by $Q(T_1)$, is computed as $Q(T_1) = 0.4 * \log 0.4 + 0.3 * \log 0.3 + 0.2 * \log 0.2 + 0.1 * \log 0.1 = -0.55$. This measure quantifies the ambiguity of an x-tuple: the more uncertain the data, the lower the quality. Our core objective is to improve the quality of x-tuples with the help of crowdsourcing.

B. Cleaning with Crowd

In Table I, we demonstrate an example of uncertain data, in which the uncertainty is essential because data extraction

Location address of T_1: 51A Hayward East New York

Xid	Tid	House no	Area	City	$Pr(t_i)$
T_1	t_1	51	Hayward	New York	0.4
T_1	t_2	51A	Hayward East	New York	0.3
T_1	t_3	51A	Hayward East	York	0.2
T_1	t_4	51	East	York	0.1

TABLE I. RUNNING EXAMPLE - UNCERTAIN DATA GENERATED BY AN INFORMATION EXTRACTION TOOL

ID	HIT content	ground truth $Pr(h_c)$	crowdsourced answer $Pr(A_c)$
HIT1	Is the House no 51?	y(0.5) n(0.5)	y(0.5) n (0.5)
HIT2	Is the House no 51A?	y(0.5) n (0.5)	y(0.5) n(0.5)
HIT3	Is the Area Hayward?	y(0.4) n (0.6)	y(0.45) n (0.55)
HIT4	Is the Area Hayward East?	y(0.5) n (0.5)	y(0.5) n (0.5)
HIT5	Is the Area East?	y(0.1) n (0.9)	y(0.3) n (0.7)
HIT6	Is the City New York?	y(0.7) n (0.3)	y(0.6) n (0.4)
HIT7	Is the City York?	y(0.1) n(0.9)	y(0.3) n (0.7)

TABLE II. RUNNING EXAMPLE - CANDIDATE HITS FOR CLEANING WITH CROWD, AND DISTRIBUTIONS OF THE GROUND TRUTH AND CROWDSOURCED ANSWER (CROWD ACCURACY = 0.75)

techniques cannot entirely 'understand' the location address written in English - *a natural language*. Therefore, it becomes critical to investigate the following question: where can we find the human perceptions concerning the given uncertain data? Fortunately, we have crowdsourcing as a promising solution. With the recent development of crowdsourcing in both academic and industrial communities, there are sophisticated online crowdsourcing platforms (e.g. Mechanical Turk, CrowdFlower) serving affordable sources of human perceptions. The expected information concerning the given uncertain data can be queried via publishing questions, named Human Intelligent Tasks (a.k.a. HITs), each of which would generate a monetary cost. Additionally, the work-flow of publishing HITs can be automated with available APIs (e.g. REST API). Facilitated with these, we can significantly improve the data quality by automatically issuing a number of HITs.

It is well-known that crowdsourcing works best when tasks can be broken down into very simple pieces [27], [26], [19]. An entire tuple (containing values of many attributes) may be too large a grain for a crowd - each individual worker may have small quibbles with a proposed tuple, so that a simple question on the correctness of tuples may get mostly negative answers, with each user declaring it less than perfect. In other words, determining the correctness of an entire tuple is too difficult for crowdsourcing workers. On the other hand, asking open-ended questions is not recommended for a crowd, because it may be difficult to integrate multiple suggestions of heterogeneous semantics. As a result, we propose to ask the crowd questions regarding individual cells. Each HIT is a question of form "Is the *c.att c*?" , where *c* and *c.att* represent a cell value and its corresponding attribute, respectively. Given an example in Table I, there are seven candidate HITs available to be selected for crowdsourcing, as shown in Table II. This kind of much simpler questions, in most circumstances, can be easily answered with a yes or no.

With the running example in Table I, we demonstrate how the crowd is going to clean the uncertain data by answering HITs, and improve the data quality as a consequence. Assuming that "I: City = New York" (i.e. HIT6) is confirmed to be true by the crowd. Then we have

$$Pr(t_1|I) = \frac{Pr(t_1)Pr(I|t_1)}{Pr(I) = Pr(t_1) + Pr(t_2)} = \frac{0.4 * 1}{0.4 + 0.3} = 0.57 > 0.5 \quad (1)$$

and similarly $Pr(t_2|I) = 0.43, Pr(t_3|I) = 0$ and $Pr(t_4|I) = 0$. As a result, the quality $Q(T_1)$ is improved from -0.55 to -0.3.

In the above example, we assume that obtaining information I from crowds with 100% confidence, which is unrealistic if we consider the error rate of crowdsourcing workers. As pointed out in paper [22], even with the simple tasks such as labelling, the quality of normal workers (not sloppy workers or spammers) is around 75%. Clearly, this quality varies depending on many factors, including the format and domain of the HIT, the time of publication, the HIT interface etc. Sometimes the quality may be even worse than 75%. Nevertheless, such noisy answers may still be conductive as far as the uncertainty is well managed. Continued with the running example, we make a more realistic assumption: "I: City = New York" is answered by a crowd C with general confidence 60%. Assuming the HIT is answered independently by the worker, we have

$$Pr(t_1|e : I\ is\ obtained\ from\ crowd) = Pr(t_1)Pr(e|t_1)/Pr(e)$$
$$= \frac{Pr(t_1)Pr(crowd\ is\ correct)}{Pr(I)Pr(C\ is\ correct) + (1 - Pr(I))Pr(C\ is\ incorrect)}$$
$$= \frac{0.4 * 0.6}{0.7 * 0.6 + 0.3 * 0.4} = 0.44$$

Similarly, $Pr(t_2|e) = 0.34$, $Pr(t_3|e) = 0.15$ and $Pr(t_4|e) = 0.07$. The quality $Q(T_1)$ is hereby improved to -0.52. From the above results, we can see that, even when the crowd is imperfect, the quality of uncertain data is still improved after cleaning. After crowdsourcing a few more HITs, the confidence of the correct tuple is increased to be close to 1, while the others approach to 0.

C. Challenges and Contributions

The above analysis shows that crowdsourcing provides a new opportunity for uncertain data cleaning. However, since each HIT is associated with a cost, we need to design solutions to maximize the data quality by selecting an optimal minimum set of HITs. This selection is not trivial due to the following two obstacles: first, the crowd has a probability to return incorrect answers; second, the HITs decomposed from an x-tuple are naturally correlated. These two obstacles lead to the NP-hardness for selecting the optimal set of HITs. To address this challenge, firstly, we design an approximation algorithm with approximation guarantee $(1 + 1/e)$; and secondly, we propose a heuristic solution, which is much more efficient than the approximation one, but has comparable effectiveness.

We summarize our contributions as follows:

First, in Section III, we indicate how to utilize noisy crowdsourced answers to improve the data quality of an x-tuple, and analyze the functional relations among the crowd's accuracy, data quality and a candidate HIT.

T	an x-tuple
$t_1...t_{\|T\|}$	tuples of T
$Pr(t_1)...Pr(t_{\|T\|})$	probabilities of tuples
$Q(T)$	data quality of T
$ATT = \{att_1...att_{\|ATT\|}\}$	semantic attributes of T
c_{ij} (c for simplicity)	value of the cell intersecting t_i and att_j
$h_{c_{ij}}(h_c)$	the cleaning HIT verifying $c_{ij}(c)$
$Pr(h_{c_{ij}})$	probability of c_{ij} being the correct value of att_j, i.e. the probability that the ground truth of $h_{c_{ij}}$ is 'yes'
m	the total number of candidate HITs
A_c	the crowdsourced answer of h_c
P_{cr}	the accuracy of the crowd
$Pr(A_c)$	probability of crowd answering 'yes' to h_c
$H(cr)$	the entropy of the crowd
$H(.)$	the entropy of a random variable
$t \models c$	c contains the correct(incorrect) value if t is correct(incorrect)

TABLE III. MEANINGS OF SYMBOLS

Second, in Section IV and V, we address the core optimization problem - how to select the most profitable questions with a cardinality constraint. We prove that the finding the exact solution is NP-hard, and an approximation algorithm as well as a heuristic solution are proposed. In addition, we derive tight lower and upper bounds, which are used for effective filtering and estimation.

Third, in Section VI, we have conducted extensive experiments on both synthetic and real data sets. The experimental results show that the data quality is significantly improved via the power of crowdsourcing.

II. PROBLEM FORMULATION

In this section, we first formally introduce x-tuple together with its quality metric. Then, we define crowd-based cleaning operations, and finally formulate our problem of utilizing crowdsourcing to clean x-tuples.

A. Uncertain Data and Quality Metric

Definition 2.1 (x-tuple with semantic attributes): For a given table under schema Σ, which consists of a set of **semantic** attributes $ATT = \{att_1, att_2, ..., att_{|ATT|}\}$. Each tuple t_i consists of three components: a unique identifier Tid, a confidence $P(t_i)$ that is the probability of t being correct, and the set of semantic attributes ATT. An x-tuple T consists of one or more tuples i.e. $T = \{t_1, t_2, ..., t_{|T|}\}$. $|ATT|$ and $|T|$ are the size of ATT and T, respectively. Each of the tuples is associated with a probability of being correct. Let $Pr(t_i)$ denote the probability of t_i being correct, then $\sum_{i=1}^{|T|} Pr(t_i) \leq 1$.

Please note the *semantic* attributes indicate the attributes describing human-intrinsic information of an entity. Besides the semantic attributes, schema Σ may also consist other attributes (e.g. IDs). In the example of Table I, "House no", "Area" and "City" are semantic attributes, while others are not. Intuitively, only data under semantic attributes worth being crowdsourced.

Definition 2.2 (Data Quality [5], [23]): Given an x-tuple T, the quality of T, denoted by $Q(T)$, is the negative value of the Shannon entropy, i.e.

$$Q(T) = \sum_{i=1}^{|T|} Pr(t_i) \log Pr(t_i) \tag{2}$$

where $\sum_{i=1}^{|T|} Pr(t_i) = 1$. If $\sum_{i=1}^{|T|} Pr(t_i) < 1$, we conceptually insert into T a tuple t_{null}, with null values for all attributes and probability $Pr(t_{null}) = 1 - \sum_{i=1}^{|T|} Pr(t_i)$.
We are using the same mathematical metric as proposed in [5], [23], which is originally called *PWS-quality*. Shannon entropy quantifies the randomness of a random variable, and low randomness indicates high quality of uncertain data. However, we adopt the metric to evaluate the quality of an individual x-tuple, rather than the answer of a query. By improving the quality of x-tuples (i.e. the data), the quality of any query answers would be naturally improved.

B. Crowdsourcing Model

Definition 2.3 (cleaning HIT): Given x-tuple T, let c_{ij} be the cell intersecting tuple $t_i \in T$ and attribute att_j. A cleaning HIT (or HIT in short) corresponding to c_{ij}, denoted $h_{c_{ij}}$, is a human intelligent task asking the crowd to verify whether c_{ij} is the correct value for att_j in T. Moreover, let $Pr(h_{c_{ij}})$ be the probability that the correct answer of $h_{c_{ij}}$ is 'yes', then

$$Pr(h_{c_{ij}}) = \sum_{t_x \models c_{ij}} Pr(t_x) \tag{3}$$

where $t_x \models c_{ij}$ denotes c_{ij} is the value of att_j in tuple t_x, i.e. t_x entails c_{ij}. In other words, $t_x \models c_{ij}$ means that if t_x is correct (incorrect), then c_{ij} is correct (incorrect).

From Definition 2.3, we can see that the candidate HITs can be easily constructed for a given x-tuple T. Since different tuples may have the same value for an attribute, the probability of c_{ij} being the correct value of att_j (i.e. the correct answer of $h_{c_{ij}}$ is yes) can be computed by summing up the probabilities of tuples in which c_{ij} is the value of att_j. The total number of HITs constructed for t, denoted by m, is upper bounded by $|Att| * |T|$, i.e. the number of attributes multiply the number of tuples. Each HIT can be considered as a random variable following a Bernoulli distribution. The distributions of HITs in Table I are demonstrated. Please note that the HITs derived from the same x-tuple are correlated in general. In the example of Table I, given that "the House no is 51A" (HIT2), we can easily infer that "the Area is Hayward East"(HIT4), and the probability of "the City is York"(HIT7) becomes 0.4 (similar to Equation 1).

Due to differences in skill levels, or the amount of time and effort spent, the answers returned by the crowd may be imperfect. We model the potential noisy answers by a general error model for the crowd, illustrated as follows.

Definition 2.4 (Crowd's Accuracy): Given a crowd cr, the crowd's accuracy (or accuracy for short), denoted by $P_{cr} \in [0.5, 1]$, is the probability that cr correctly answers each HIT. In addition, each HIT is assumed to be answered independently. How to model the crowds is application-specific. While some works assume that the crowdsourced answers are 100% accurate [37], [36], [38], [26], we adopt a more general error model, which ensures that the answer returned by the crowd is always correct with a probability no lower than $1/2$, as shown in Definition 2.4. This is a classical crowdsourcing model widely used by a stream of works [14], [7], [29], [24], [20]. A crowd may have different accuracies for different domains. The accuracy for a domain can be easily estimated with a set of sample HITs with ground truth.

Definition 2.5 (Entropy of Crowd): Given a crowd cr and its accuracy P_{cr}, the entropy of cr is

$$H(cr) = -P_{cr} \log P_{cr} - (1 - P_{cr}) \log (1 - P_{cr}) \quad (4)$$

Given the crowd's accuracy, $H(cr)$ is a **positive** constant measuring the randomness of the crowd's behaviour.

C. Problem Statement

Now, we formally define the problem of data cleaning via crowdsourcing as follows.

Definition 2.6 (Problem Statement): Given an x-tuple T, the budget B, a crowd with accuracy P_{cr}, our goal is to maximize the quality $Q(T)$ by selecting HITs and receiving at most B crowdsourced answers.

From the above statement, the crowd is considered as a tool for cleaning uncertain data. However, since crowds may be noisy and unreliable, the first concern is *how to use the noisy crowdsourced answers to improve data quality*. In Section III, we present a series of theoretical analyses, proving that the expected improvement of data quality is always non-negative. Then, in Section IV, we optimize the crowdsourcing process by selecting the most profitable set of HITs.

III. Utilization of Crowdsourced Answers

A crowd can be noisy, hence each crowdsourced answer is inherently uncertain. In this section, we discuss how to use uncertain crowdsourced answers to improve the data quality of x-tuples.

A. Merging Crowdsourced Answers into X-tuples

Since both crowdsourced answers and x-tuple are essentially uncertain, the effect of crowdsourced answers on data quality can be treated as posterior probabilities conditioning on answers, which is properly addressed with Bayes' theorem.

First, for a given x-tuple T, we clarify the functional relations among the crowd accuracy P_{cr}, data quality $Q(T)$ and a candidate HIT h_c. Let A_c be the answer of h_c provided by the crowd, and $Pr(A_c)$ and $1-Pr(A_c)$ be the probabilities of 'yes' and 'no' being answered, respectively. Please note the difference between h_c and A_c: h_c denotes the 'ground truth' of the HIT, i.e.

$$Pr(the\ ground\ truth\ of\ h_c\ is\ yes) = Pr(h_c)$$

while A_c denotes the 'crowdsourced answer' of the HIT, and

$$Pr(h_c\ is\ answered\ yes\ by\ the\ crowd) = Pr(A_c)$$

Because the crowd may make mistakes, they follow different distributions. Since the HITs are assumed to be answered independently, the relation between $Pr(h_c)$ and $Pr(A_c)$ can be expressed as the following equation.

$$Pr(A_c) = Pr(h_c)P_{cr} + (1 - Pr(h_c))(1 - P_{cr}) \quad (5)$$

In table II, the values of $Pr(h_c)$ and $Pr(A_c)$ are demonstrated, which reveal the differences discussed above.

Then, we are interested in how this answer A_c would affect the quality, i.e. how to compute $Q(T|A_c = yes)$ and $Q(T|A_c = no)$. In fact, the probability of each $t \in T$ is updated when the A_c is received. Intuitively, if t indicates contradicting content as A_c, then $Pr(t)$ should decrease; otherwise $Pr(t)$ should increase since it is further confirmed by the crowd. Explicitly, for each crowdsourced answer A_c, the probability of any $t \in T$ is modified as

$$\begin{aligned} Pr(t|A_c = yes) &= Pr(t)Pr(A_c = yes|t)/Pr(A_c) \\ Pr(t|A_c = no) &= Pr(t)Pr(A_c = no|t)/(1 - Pr(A_c)) \end{aligned} \quad (6)$$

where $Pr(A_c|t) = P_{cr}$ when $t \models c$ (i.e. the crowd confirms t), and $Pr(A_c|t) = 1 - P_{cr}$ when $t \models \neg c$ (i.e. the crowd disagrees with t). By recursively applying Equation 6, conflicting crowdsourced answers are integrated into the x-tuple t.

It is easy to perform the algebraic manipulations to show that, for any two answers A_{c_0} and A_{c_1}, we have

$$Pr(t|A_{c_0}, A_{c_1}) = Pr(t|A_{c_1}, A_{c_0}) \quad (7)$$

Equation 7 indicates that the final result of adjustment is independent of the sequence of the crowdsourced answers. In other words, when we have a deterministic set of HITs, it does not matter in what sequence the answers are used for adjusting the distribution in the x-tuple. In contrast, what matters is to determine the set of HITs to be asked, which is the core challenge addressed in Section IV.

B. Crowd's Accuracy vs. Data Quality

According to the Equations 5 and 6, we introduce two theorems to conclude the relation between the crowd accuracy and the data quality.

Theorem 3.1 (Harmless Random): If a crowd randomly answers a HIT, i.e. $P_{cr} = 0.5$, then it does not affect the data quality.

Proof: When a crowd randomly answers HITs, i.e. $P_{cr} = 0.5$. By Equation 5, we have $Pr(A_c) = 0.5$. In addition, by Equation 6, we have $Pr(t|A_c = yes) = Pr(t|A_c = no) = Pr(t)$. Therefore, the distribution of the tuples remains unchanged, so is the data quality. This completes the proof of Theorem 3.1. ■

Now we compute the expectation of data quality after receiving the answer A_c, denoted by $\mathbb{E}Q(T|A_c)$. For simplicity, we summarize the result of the computation with lemma 3.2 as follows, and include the process of deduction as the proof,

which is detailed in the appendix. Then, we achieve the proof of Theorem 3.3 based on the result of Lemma 3.2.

Lemma 3.2:

$$\mathbb{E}Q(T|A_c) = Q(X) + P_{cr}\log P_{cr} + (1-P_{cr})\log(1-P_{cr}) - Pr(A_c)\log Pr(A_c) - (1-Pr(A_c))\log(1-Pr(A_c)) \quad (8)$$

Proof: (Sketch) Facilitated by Equation 5, we have

$$\mathbb{E}Q(T|A_c) = Pr(A_c)Q(T|A_c = yes) + (1-Pr(A_c))Q(T|A_c = no)$$
$$= Pr(A_c)\sum_{t\in T} Pr(t|A_c = yes)\log Pr(t|A_c = yes) + (1-Pr(A_c))\sum_{t\in T} Pr(t|A_c = no)\log Pr(t|A_c = no)$$

By substituting Equation 6 and Equation 3, we have

$$\mathbb{E}Q(T|A_c) = \sum_{t_i \models c} Pr(t_i)\log Pr(t_i) + Pr(h_c)P_{cr}\log\frac{P_{cr}}{Pr(A_c)} + \sum_{t_i \models \neg c} Pr(t_i)\log Pr(t_i) + (1-Pr(h_c))(1-P_{cr})\log\frac{1-P_{cr}}{Pr(A_c)} + Pr(h_c)(1-P_{cr})\log\frac{1-P_{cr}}{1-Pr(A_c)} + (1-Pr(h_c))P_{cr}\log\frac{P_{cr}}{1-Pr(A_c)}$$
$$= Q(X) + P_{cr}\log P_{cr} + (1-P_{cr})\log(1-P_{cr}) - Pr(A_c)\log Pr(A_c) - (1-Pr(A_c))\log(1-Pr(A_c))$$

■

Theorem 3.3 (Non-negative Expectation): Given an x-tuple T, if a HIT h_c is answered by a crowd with accuracy P_{cr}, the expected improvement of data quality of T is mathematically equivalent to the difference between 'the entropy of the crowdsourced answer of h_c' and the 'entropy of the crowd (Definition 2.5)', which is non-negative.

Proof: From the result of Lemma 3.2,we derive the expected improvement of data quality:

$$\Delta Q(T) = \mathbb{E}Q(T|A_c) - Q(T) = P_{cr}\log P_{cr} + (1-P_{cr})\log(1-P_{cr}) - Pr(A_c)\log Pr(A_c) - (1-Pr(A_c))\log(1-Pr(A_c)) \quad (9)$$

Descriptively, the expected quality improvement is equivalent to the 'entropy of the answer minus the entropy of the crowd', which is always non-negative as shown as follows.

By substituting Equation 5 into Equation 9, we take the derivative of $\Delta Q(T)$

$$\frac{\partial \Delta Q(T)}{\partial P_{cr}} = (2Pr(h_c)-1)\log\frac{1-(1-Pr(h_c))(1-P_{cr})-Pr(h_c)P_{cr}}{2Pr(h_c)P_{cr}-Pr(h_c)-P_{cr}+1} + \log\frac{P_{cr}}{1-P_{cr}} \quad (10)$$

It is easy to see that $\Delta Q(T)$ achieves minimum when $P_{cr} = 0.5$. Together with the result that $\Delta Q(T) = 0$ if $P_{cr} = 0.5$ (Theorem 3.1), we complete the proof of Theorem 3.3. ■

From the above analysis, we always have non-negative expectation of quality improvement for asking each HIT. However, the quality of an x-tuple does not necessarily monotonically increase as receiving crowdsourced answers. It is possible for the quality to decrease, when a 'surprising' answer is received - a 'no' is answered by the crowd for a HIT with high probability to be correct, or vice versa. However, one can see that the data quality obviously converges to zero (i.e. the upper bound of data quality), since each HIT has a non-negative expectation on the change of quality. To achieve fast convergence, the key issue is how to select the HITs wisely.

IV. K-HIT SELECTION

The core computational challenge we need to address is to select a collection of HITs with a cardinality constraint k. For the given budget B, we are interested in asking k HITs at each iteration, so there are totally $\lceil \frac{B}{k} \rceil$ iterations for budget to run out. When $k > 1$, different workers can then pick up these tasks and solve them in parallel, cutting down wall-clock time. Therefore, for a given budget of HITs B, larger k value leads to lower latency. However, we pay for this by having some questions answered that are not at the top of the list - we are issuing k good questions rather than only the very best one. At each iteration, we select k distinct HITs; but the same HIT may be repetitively selected in different iterations. Please note that the selection of HITs depends on the crowdsourced answers received from previous iterations. So the overall selection is an adaptive process, and how many times each HIT is asked during the whole process is determined by the crowdsourced answers and the proposed algorithms discussed in the rest of this section.

For a given x-tuple with m candidate distinct HITs, there are potentially C_m^k possible selections. In this section, we discuss how to select k HITs in order to maximize the expected quality improvement.

A. Deriving Objective Function

For a given set of k HITs - $S_h = \{h_{c_1}, h_{c_2}, ..., h_{c_k}\}$, we derive the expectation of quality improvement caused by the aggregation of crowdsourced answers of these k HITs. Let A_{S_h} denote the aggregated answer of S_h, then A_{S_h} is a discrete random variable with 2^k possible outcomes, since each h_{c_j} is either answered 'yes' or 'no'. Let D_A and p_A be the domain and probability distribution of A_{S_h}, respectively, i.e.

$$\begin{aligned} &D_A = \{a_i | a_i \subseteq 2^{\{h_{c_1}, h_{c_2}, ..., h_{c_k}\}} \text{ and} \\ &\forall h_{c_j} \in a_i \text{ , } h_{c_j} \text{ is answered yes by crowd;} \\ &\text{ and } \forall h_{c_j} \notin a_i \text{ , } h_{c_j} \text{ is answered no by crowd}\} \\ &p_A = (Pr(a_1), Pr(a_2), ..., Pr(a_{2^k})) \end{aligned} \quad (11)$$

We aim to find a set of k HITs such that their answers would lead to the maximal (expected) improvement of data

quality. So we investigate expected data quality by receiving the crowdsourced answer A_{S_h} - we have

$$\begin{aligned}\mathbb{E}Q(T|A_{S_h}) &= \sum_{a_i \in D_A} Pr(a_i) \sum_{t \in T} Pr(t|a_i) \log Pr(t|a_i) \\ &= \sum_{a_i \in D_A} \sum_{t \in T} Pr(t) Pr(a_i|t) \log \frac{Pr(t)Pr(a_i|t)}{Pr(a_i)} \\ &= \sum_{t \in T} Pr(t) \log Pr(t) + \sum_{a_i, t} Pr(t) Pr(a_i|t) \log Pr(a_i|t) \\ &\quad - \sum_{a_i \in D_A} Pr(a_i) \log Pr(a_i)\end{aligned}$$

Then we have the expected improvement

$$\begin{aligned}&\Delta Q_{S_h}(T) = \mathbb{E}Q(T|A_{S_h}) - Q(T) \\ &= \sum_{a_i, t} Pr(t) Pr(a_i|t) \log Pr(a_i|t) - \sum_{a_i \in D_A} Pr(a_i) \log Pr(a_i)\end{aligned} \tag{12}$$

Note $\sum_{a_i,t} Pr(t)Pr(a_i|t) \log Pr(a_i|t) = -H(A_{S_h}|T)$, where $H(A_{S_h}|T)$ is the entropy of A_{S_h} conditioned on T. Recall that A_{S_h} is a set of k HITs, each of which is answered by the crowd independently. This reflects that answers in A_{S_h} are independent conditioned on T. So

$$\begin{aligned}&H(A_{S_h}|T) = -\sum_{i=1}^{k} H(A_{c_i}|T) \\ &= -\sum_{i=1}^{k} \{P_{cr} \log P_{cr} + (1 - P_{cr}) \log (1 - P_{cr})\} \\ &= -k\{P_{cr} \log P_{cr} + (1 - P_{cr}) \log (1 - P_{cr})\} = kH(cr)\end{aligned} \tag{13}$$

Therefore, given fixed k and crowd accuracy P_{cr}, $H(A_{S_h}|T)$ is equivalent k times the entropy of the crowd $H(cr)$, i.e. a constant. So, we have

$$\begin{aligned}\Delta Q_{S_h}(T) &= -\sum_{a_i \in D_A} Pr(a_i) \log Pr(a_i) - kH(cr) \\ &= H(A_{S_h}) - kH(cr)\end{aligned} \tag{14}$$

We highlight the conclusion of Equation 14 with the following theorem.

Theorem 4.1 (Objective Function): Given an x-tuple, a set of k cleaning HITs S_h, and the crowd accuracy, the expected quality improvement by asking these HITs is equivalent to difference between the 'entropy of the answer' and 'k times the entropy of crowd'.

Since $kH(cr)$ is a constant, we only need to select HITs to maximize $H(A_{S_h})$. Formally, we have the following optimization goal:

$$S_h := \arg\max_{S_h} \ H(A_{S_h}) \tag{15}$$

B. Lower and Upper bounds

According to Theorem 4.1, we expect to select a set of k HITs, with highest $H(A_{S_h})$. For a given set HITs S_h, computing the exact value of $H(A_{S_h})$ encounters exponential complexity, due to the correlation among HITs and the crowd imperfection. Explicitly, we present the closed-form formula for the computation of $H(A_{S_h})$

$$\begin{aligned}&H(A_{S_h}) = \\ &-\sum_{a \in D_A} [\sum_{t \in T} Pr(t) \prod_{h_{c_i} \in a_i, t \models c_i} P_{cr} \prod_{h_{c_i} \in a_i, t \models \neg c_i} (1 - P_{cr}) \\ &+ \log \sum_{t \in T} Pr(t) \prod_{h_{c_i} \in a_i, t \models c_i} P_{cr} \prod_{h_{c_i} \in a_i, t \models \neg c_i} (1 - P_{cr})]\end{aligned} \tag{16}$$

Clearly, the exact computation yields exponential complexity w.r.t k, which is inefficient when k is large. Thus, we are interested in reducing or avoiding the exact computation pertaining to $H(A_{S_h})$. In this subsection, we include an upper bound and a lower bound, both of which can be computed in linear time. The bounds can be used to enable effective filtering (in Section V-A) and heuristic estimation (in Section V-B).

1) Upper Bound: We use $H(S_h)$ to denote the joint entropy of HITs in S_h. Please note the difference between $H(S_h)$ and $H(A_{S_h})$ - $H(S_h)$ measures the randomness of the HITs, or more precisely, 'the ground truth of the HITs'; while $H(A_{S_h})$ measures the randomness of the 'crowdsourced answer of the HITs'. Intuitively, this difference is caused by the fact that the crowd makes mistakes. Then by the chain rule of Shannon entropy, we have $H(S_h|A_{S_h}) = H(A_{S_h}, S_h) - H(A_{S_h})$, therefore

$$\begin{aligned}H(A_{S_h}) &= H(A_{S_h}, S_h) - H(S_h|A_{S_h}) \\ &= H(A_{S_h}|S_h) + H(S_h) - H(S_h|A_{S_h})\end{aligned} \tag{17}$$

Note all the HITs are independently answered, so

$$H(A_{S_h}|S_h) = \sum_{i=1}^{k} H(A_{h_i}|S_h) = \sum_{i=1}^{k} H(cr) = kH(cr) \tag{18}$$

Therefore, we have the following upper bound -

$$\begin{aligned}H(A_{S_h}) &= H(S_h) + kH(cr) - H(S_h|A_{S_h}) \\ &\leq H(S_h) + kH(cr)\end{aligned} \tag{19}$$

2) Lower Bound: Recall that $H(S_h)$ and $H(A_{S_h})$ are different because the crowd makes mistakes. Inspired by this, we define an indicator function:

$$Y = \begin{cases} 0 \ crowd \ correctly \ answers \ all \ the \ HITs \ in \ S_h \\ 1 \ crowd \ answers \ at \ least \ one \ HIT \ incorrectly \end{cases} \tag{20}$$

Naturally, Y determines whether the crowd makes mistakes on the HITs in S_h, and we have the distribution of Y - $Pr(Y = 0) = P_{cr}^k$ and $Pr(Y = 1) = 1 - P_{cr}^k$.

Now, we rewrite $H(S_h|A_{S_h})$

$$\begin{aligned}
&H(S_h|A_{S_h}) = H(S_h|A_{S_h}) - H(S_h|A_{S_h}, Y) + H(S_h|A_{S_h}, Y) \\
&= H(Y|A_{S_h}) - H(Y|A_{S_h}, S_h) + H(S_h|A_{S_h}, Y) \\
&= H(Y|A_{S_h}) + H(S_h|A_{S_h}, Y) \\
&\leq H(Y) + H(S_h|A_{S_h}, Y) \\
&= H(Y) + \sum_{a_i \in D_A} [Pr(A_{S_h} = a_i, Y = 0) H(S_h|A_{S_h} = a_i, Y = 0) \\
&\quad + Pr(A_{S_h} = a_i, Y = 1) H(S_h|A_{S_h} = a_i, Y = 1)]
\end{aligned} \tag{21}$$

Please note that A_{S_h} is equivalent to the ground truth of S_h when the crowd makes no error, i.e. $Y = 0$. So the first term in the summation becomes 0:

$$H(S_h|A_{S_h} = a_i, Y = 0) = 0 \tag{22}$$

The entropy of any random variable is maximized when each possible outcome has the same probability. Since A_{S_h} includes k HITs, the size of the domain of A_{S_h} is 2^k. When $Y = 1$, we know that the crowd makes errors, so the number of possible outcome becomes $2^k - 1$, i.e. the original minus the value that A_{S_h} has taken. Therefore, we have

$$H(S_h|A_{S_h} = a_i, Y = 1) \leq \log(2^k - 1) \tag{23}$$

Substituting formula 22 and 23 into formula 21, we can show

$$\begin{aligned}
H(S_h|A_{S_h}) &\leq H(Y) + \log(2^k - 1) \sum_{a_i \in D_A} Pr(A_{S_h} = a_i, Y = 1) \\
&\leq H(Y) + Pr(Y = 1) \log(2^k - 1) \\
&\leq -P_{cr}^k \log P_{cr}^k - (1 - P_{cr}^k) \log(1 - P_{cr}^k) \\
&\quad + (1 - P_{cr}^k) \log(2^k - 1)
\end{aligned} \tag{24}$$

Finally, we substitute formula 24 into formula 19, we have

$$\begin{aligned}
H(A_{S_h}) =& H(S_h) + kH(cr) - H(S_h|A_{S_h}) \\
\geq& H(S_h) + kH(cr) + P_{cr}^k \log P_{cr}^k + \\
& (1 - P_{cr}^k) \log(1 - P_{cr}^k) - (1 - P_{cr}^k) \log(2^k - 1)
\end{aligned} \tag{25}$$

C. Computing the Bounds: X-partition Algorithm

From formula 19 and 25, one can see that we only need to compute the joint entropy $H(S_h)$, since the other components are all constant for a given x-tuple and a set of HITs. A naive way of computing the joint entropy is to compute all the marginal probabilities, hence yields to $O(2^k)$ complexity. However, with creating a small $O(|T|)$ in-memory index, we can achieve overall linear complexity - $O(k|T|)$.

We propose a novel algorithm named "X-partition". The intuition of the algorithm is, for each HIT h_c, the x-tuple T can be divided in to two parts, with $t_i \models c$ and $t_j \models \neg c$ respectively. As a consequence, k HITs can partition T into at most 2^k parts, and the aggregated probability of each part (i.e. the sum of probabilities of entries within a part) constitutes the marginal distribution of S_h. Note each part includes at least one tuple, so the total number of parts is at most $|T|$. Now we illustrate the algorithm in detail as following steps.

Step 1: Remove a HIT h_c from S_h.

Step 2: Partition the table into two parts T_0 and T_1, where $\forall t \in T_0, t \models c$, and $\forall t \in T_1, t \models \neg c$.

Step 3: Remove a HIT $h_{c'}$ from S_h

Step 4: For each of the current parts T_l, further divide it into two parts, T_{l0} and T_{l1}.

Step 5: repeat Step 3 and Step 4 until all k HITs are removed, then return $-\sum p \log p$, where p is the sum of probabilities of tuples within each part.

Correctness and Complexity: The correctness of the algorithm is straightforward. Each part T_l corresponds to a point of the marginal distribution of the S_h, so the entropy of these parts are equivalent to the joint entropy. In addition, the order of HITs would not affect the final partitioning result. At each iteration, for a given HIT, partitioning requires all the tuples, hence takes $O(|T|)$ time. As a result, the overall complexity becomes $O(k|T|)$, which is essentially linear w.r.t the size of input.

Running example: Given the example in Table I and II, we consider a set of HITs $S_h = \{h_2, h_3\}$. Assume the accuracy of the crowd is 0.75, then as shown in Table II, we have - $Pr(h_2) = 0.5$,$Pr(h_3) = 0.4$, i.e. the probability of h_2 (h_3) has ground truth 'yes' are 0.5 (0.4); moreover, the probability of h_2 (h_3) being answered 'yes' by the crowd is 0.5 (0.45). Now we present the distributions of S_h and A_{S_h} as follows.

outcomes	prob (S_h)	prob (A_{S_h})
$h_2 = y$,$h_3 = y$	0	0.175
$h_2 = y$, $h_3 = n$	0.5	0.325
$h_2 = n$,$h_3 = y$	0.4	0.275
$h_2 = n$,$h_3 = n$	0.1	0.225

Note the distribution of S_h indicates the ground truth of the HITs, which is derived from the x-tuple in Table I. Clearly, this distribution is irrelevant with the accuracy of the crowd. On the other hand, A_{S_h} denotes the distribution of the crowdsourced answer, which is computed according to formula 16. Having these two distributions, we can easily compute $H(S_h) = 1.36$, and $H(A_{S_h}) = 1.96$. We also have $H(cr) = 0.81$, then the upper bound is $H(S_h) + kH(cr) = 1.36 + 2 * 0.81 = 2.11 \geq 1.96$ and lower bound $H(S_h) + kH(cr) + P_{cr}^k \log P_{cr}^k + (1 - P_{cr}^k) \log(1 - P_{cr}^k) - \log(2^k - 1) = 1.36 + 2 * 0.81 - 0.337 - 1.58 = 0.193 \leq 1.96$

V. Hardness and Algorithms

One can see that the optimization problem derived in formula 15 is non-linear, due to the nature of Shannon entropy. In fact, the optimization problem is to find k HITs from the m candidate HITs, so the searching space is of complexity $O(m!)$. However, $H(A_{S_h})$ is the entropy of the crowdsourced answer of a set of HITs, and it is known that entropy is a sub-modular function [2]. Our optimization problem is essentially a sub-problem of the maximization of a general submodular function. We prove that finding the optimal solution for our optimization problem is NP-hard.

Theorem 5.1 (NP-hardness): It is NP-hard to find a set of k HITs such that the expected improvement of data quality is maximized.

Proof: It is sufficient to prove the NP-completeness of its decision version, Decision k-HIT Selection (DkHIT) , i.e. given an x-tuple and an integer k and a value H_0, decide whether one can find a set of k HITs such that $H(A_{S_h}) \geq H_0$. To reach the NP-completeness of DkHIT, it is sufficient to prove a special case of DkHIT is NPC. Now we state the special case of DkHIT by adding the following constraint on T: for each way of partitioning T into two subsets S_1 and S_2, there exists a cell c such that $(\forall t_i \in T_1, t_i \models c) \wedge (\forall t_j \in T_2, t_i \models \neg c)$. With this constraint, we reduce this sub-problem of DkHIT to the *set partition problem*. The partition problem is the task of deciding whether a given multiset of positive integers can be partitioned into two subsets S_1 and S_2 such that the sum of the numbers in S_1 equals the sum of the numbers in S_2. Given a set partition problem with input multiset S, let $Sum = \sum_{x \in S} x$. We create a tuple t_i for each positive integer $x_i \in S$, and assign its possibility $Pr(t_i) = x_i/Sum$. Let the cells satisfy the constraint, and we set $k = 1, \Delta H = -\log(0.5) = 1$ for DkHIT. Assume there is yes-certificate for the special case of DkHIT when $k = 1, \Delta Q = \log(0.5) = -1$. Since $k = 1$, $H_{A_{S_h}}$ is actually equivalent to $H(A_c)$. Then there exists a cell c such that $Pr(h_c) = 0.5$. Therefore, by the constraint, there is a way to partition T into two subsets, each with aggregate probability 0.5. Since the mapping from the positive integers to the tuples is one-to-one, we obtain a yes-certificate for the special case of DkHIT. If there is a yes-certificate for the set partition problem, then the T can be partitioned into two subsets, each with aggregate probability 0.5. According to the constraint, there exists a cell c with $Pr(h_c) = 0.5$. Then, selecting $S_Q = \{h_c\}$ would achieve quality improvement $H_{A_h} = -0.5 \log 0.5 - 0.5 \log 0.5 = 1$. Therefore,$\{h_c\}$ serves as yes-certificate for the special case of DkHIT ■

A. Approximation Algorithm

Although finding the optimal solution is intractable, it is known that the maximization of submodular functions can be approximated with a performance guarantee of $(1-1/e)$, by a greedy algorithm [18] - we iteratively select the best one HIT, given the ones selected so far.

Formally, we have the optimization function at the k^{th} iteration:

$$x := \arg\max_x H(A_{(S_{k-1} \cup \{h_x\})}) \tag{26}$$

where S_{k-1} is the set of HITs selected from previous iterations.

At each iteration, we need to determine one HIT h_x out of m candidates, so $H(A_{(S_{k-1} \cup \{h_x\})})$ would be computed at most m times to find the maximum. Therefore, this approximation algorithm entails time complexity $O(2^k m)$, i.e. linear w.r.t the size of the x-tuple, but exponential w.r.t k. In order to further improve the efficiency, we propose two pruning methods, illustrated as follows.

Instance-level pruning:

We first adopt the lower upper bounds derived from Section IV-B. For a pair of HITs h_0 and h_1, if we have

$$H(A_{(S_{k-1} \cup \{h_0\})}).lb \geq H(A_{(S_{k-1} \cup \{h_1\})}).ub$$

then h_1 can be safely pruned. Please note that $A_{(S_{k-1} \cup \{h_0\})}.lb$ and $A_{(S_{k-1} \cup \{h_1\})}.ub$ denote the lower bound of $A_{(S_{k-1} \cup \{h_0\})}$ and the upper bound of $A_{(S_{k-1} \cup \{h_1\})}$, respectively. Both of them can be efficiently computed within $O(k)$ time by the X-partition algorithm in Section IV-C.

Algorithm-level pruning:

Submodularity can be exploited algorithmically to implement an accelerated variant of the greedy algorithm. In each of the k iterations, the greedy algorithm must identify the HIT with maximum marginal gain, given the HITs selected in the previous iterations. The key insight is that, as a consequence of submodularity of the objective function, the marginal benefit of any HIT is monotonically non-increasing during the iterations of the algorithm. In other words, for all h and k, the submodularity guarantees that $H(A_{(S_k \cup \{h\})}) \leq H(A_{(S_{k-1} \cup \{h\})})$. Therefore, we maintain a list of upper bounds (i.e.$A_{(S_{k-1} \cup \{h\})}$) on the marginal gains sorted in decreasing order. In each iteration, the algorithm extracts the HIT h_0 with the highest value from the ordered list. As a result, for all h_1, if we have

$$A_{(S_k \cup \{h_0\})} \geq A_{(S_{k-1} \cup \{h_1\})} \tag{27}$$

then, h_1 can be safely pruned.

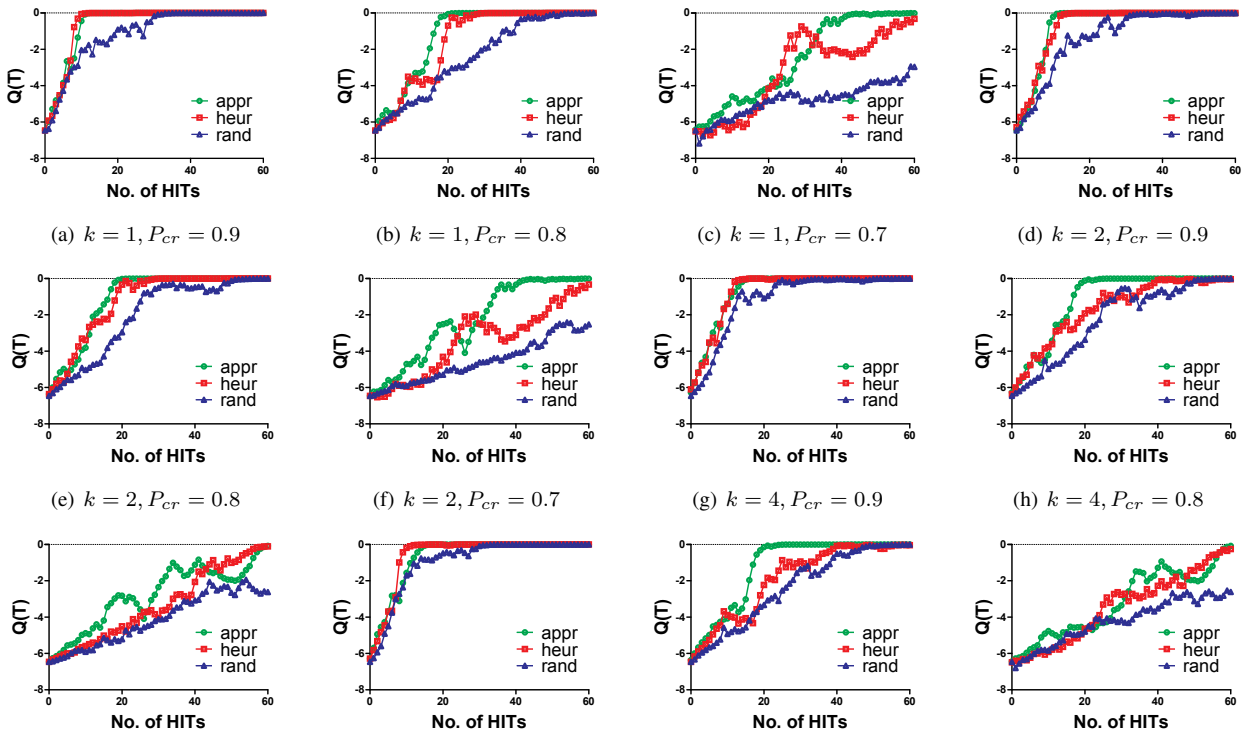
B. Heuristic Algorithm

In the approximation algorithm above, after applying the pruning methods, we may still have to compute the exact value of $H(A_{(S_{k-1} \cup \{h_x\})})$ for several times, each of which entails exponential complexity w.r.t k. This can be inefficient when k is large. In this subsection, we use the average of upper bound and lower bound to estimate the value of $H(A_{(S_{k-1} \cup \{h_x\})})$ for HIT selection. Explicitly, we apply the following estimator:

$$\begin{aligned} H(A_{(S_{k-1} \cup \{h_x\})}) &\approx \frac{A_{(S_{k-1} \cup \{h_x\})}.lb + A_{(S_{k-1} \cup \{h_x\})}.ub}{2} \\ &= H(S_h) + kH(cr) \\ &\quad + \frac{P_{cr}^k \log P_{cr}^k + (1 - P_{cr}^k) \log (1 - P_{cr}^k) - \log (2^k - 1)}{2} \end{aligned} \tag{28}$$

We compute the estimator with formula 28, and select the HIT with the highest estimated value. Therefore, considering the entire heuristic algorithm, we do not need to compute any exact value of $H(A_{(S_{k-1} \cup \{h_x\})})$. In the experiments, we show that the effectiveness of this heuristic algorithm is comparable with the approximation algorithm.

The heuristic algorithm is particular useful when k is large - it entails very short (linear w.r.t k) execution time for the program to select HITs. On the other hand, recall that larger k would lead to less overall time cost of crowdsourcing (i.e. the time waiting for the crowd to return answers), since multiple workers can take HITs in parallel. So the heuristic algorithm is recommended over the approximation algorithm when the user expects short overall running time (program execution time + crowdsourcing). In other words, when the budget is the main constraint, small k and the approximation algorithm should be adopted, whereas the heuristic algorithm with a large k value is suggested if the time-efficiency is the primary constraint.



(a) $k = 1, P_{cr} = 0.9$ (b) $k = 1, P_{cr} = 0.8$ (c) $k = 1, P_{cr} = 0.7$ (d) $k = 2, P_{cr} = 0.9$

(e) $k = 2, P_{cr} = 0.8$ (f) $k = 2, P_{cr} = 0.7$ (g) $k = 4, P_{cr} = 0.9$ (h) $k = 4, P_{cr} = 0.8$

(i) $k = 4, P_{cr} = 0.7$ (j) $k = 8, P_{cr} = 0.9$ (k) $k = 8, P_{cr} = 0.8$ (l) $k = 8, P_{cr} = 0.7$

Fig. 1. $P_{cr} = 0.9, 0.8, 0.7$ and $k = 1, 2, 4, 8$

VI. EXPERIMENTAL EVALUATION

The goal of our experiments is twofold: first, we study the effect of different parameters for quality improvement with respect to our approximation algorithm and heuristic algorithm; second, we compare the two proposed algorithms with a baseline algorithm, which randomly selects HITs. With simulation on synthetic data, we explore wide ranges of values for our parameters P_{cr} and k. In addition, we test our methods with Amazon Mechanical Turk, which is a well-known public crowdsourcing platform.

A. Simulation on Synthetic Data

In this subsection, we conduct a series of experiments on synthetic data with a simulated crowd. In particular, a crowd answers each HIT independently, and each HIT has probability P_{cr} to be correctly answered. We set the total budget of HITs $B = 60$ for an x-tuple, and at each iteration we select k HITs, so there are totally $\lceil 60/k \rceil$ iterations (there are maybe less than k HITs in the last iteration). In the experiments, we demonstrate the performances of the approximation algorithm (**appr**), the heuristic algorithm (**heur**) and a naive algorithm (**rand**) - each HIT is selected randomly. In this subsection, we use a total of $10,000$ x-tuples, each of which contains 50 tuples and 20 attributes, and report the average performance. Since each x-tuple usually contains a small number of tuples (less than 15 in real data sets used in Section VI-C), the synthetic data is fairly large enough to test the scalability of the proposed algorithms. Please note that the proposed algorithms are both **linear** w.r.t the size of x-tuples. The main computational challenge is that the time cost of **appr** is exponential w.r.t k. This is thoroughly evaluated in the experiments.

We first present the results on varying the crowd accuracy P_{cr}. As shown in Figures 1(a)-1(l), regardless the setting of k and the option of k-HIT selection algorithm, $Q(T)$ is gradually improved with the reception of crowdsourced answers, when P_{cr} is set to 0.7, 0.8 and 0.9. This is consistent with our theoretical analysis, that is any HIT would positively affect the quality of the x-tuple when P_{cr} is larger than 50%. Moreover, for all three competing algorithms, we find that the higher P_{cr} is, the faster the convergence. This finding is supported by observing Figures 1(a)-1(c), Figures 1(d)-1(f), Figures 1(g)-1(i), and Figures 1(j)-1(l). In other words, we would need less HITs for a crowd with higher accuracy.

Second, we discuss the effectiveness of the algorithm with varying k, which is the number of questions selected for each iteration. For a given budget $B = 60$, larger k leads to less iterations. Having more than one (k) HITs on the crowd, different workers can take them on parallel. We set k to 1,2,4,8, and test the performances of *appr* and *heur*. As shown in Figure 1, smaller k tends to be more effective on quality improvement. In fact, the larger k is, the less advantage *appr* and *heur* have comparing to a random selection. This is because each HIT is selected based on crowdsourced answers of HITs selected in previous iterations. Recall that we select

k out of m candidates at each iteration, so when $k = m$, *appr* and *heur* are the same as random selection, i.e. select all of the HITs we have.

Third, we compare the performances of the approximation algorithm (*appr*), the heuristic algorithm (*heur*) and a naive algorithm (*rand*). As indicated in Figure 1, *heur* and *appr* significantly outperform rand in all cases. It's worth noting that *appr* and *heur* have the same performance when $k = 1$. This is because, at each iteration, both algorithms would always select the HIT with probability closest to 0.5, which happens to achieve their respective local optima. When $k > 1$, *heur* and *appr* occasionally have similar performance (e.g. in Figures 1(d),1(h),1(j)); whereas *appr* performs slightly better than *heur* in most cases (e.g. Figures 1(e),1(f),1(g),1(i),1(k),1(l)) in terms of the number of HITs.

B. Efficiency

In Section V-A and V-B, the approximation algorithm (*appr*) and the heuristic algorithm (*heur*) are proposed as solutions for k-HIT selection. Recall that *appr* generates near-optimal solutions with approximation guarantee, but it encounters high computational cost when k is large. Therefore, two filtering techniques are proposed. In this subsection, we demonstrate the computational cost of the proposed algorithms in Figure 2. In particular, the curves labelled with *appr_i_pruning* and *appr_a_pruning* represent the approximation algorithm facilitated by the instance-level pruning and algorithm-level pruning, respectively. In addition, the curve *appr_both_prunings* indicates that both pruning methods are used. As a result, the pruning techniques significantly improve the efficiency of the *appr*, and the time cost of *heur* is linear w.r.t k. We can observe that the instance-level pruning is more effective than the algorithm-level one. In addition, combining the pruning methods is better than any single one of them. We also run the exact solution of k-HIT selection, which enumerates all the possible size-k sets of HITs. However, due to the NP-hardness of the exact solution, we cut off the experiments when $k > 7$, in which it takes more than half an hour.

C. Testing on Amazon Mechanical Turk

We implement our two approaches on Amazon Mechanical Turk (AMT), which is a widely used crowdsourcing marketplace. We tested two real-world data sets, namely *Address* and *Call-for-paper*. In particular, *Address* includes 30 x-tuples, each of which has 5 attributes and 3-10 tuples. It contains data of facility addresses, which are extracted from plaintext by a CRF-based extraction tool. On the other hand, *Call-for-paper* includes 40 x-tuples, each of which has 7 attributes and 3-15 tuples. *Call-for-paper* is aggregated from a number of websites containing general information of computer-science conferences, including the location city, submission deadlines etc. These websites include wikiCFP (wikicfp.com), confSearch (www.confsearch.org) and some personal web-pages manually maintained by computer scientists. We designed a crawler to fetch data from each website. Since different data sources may

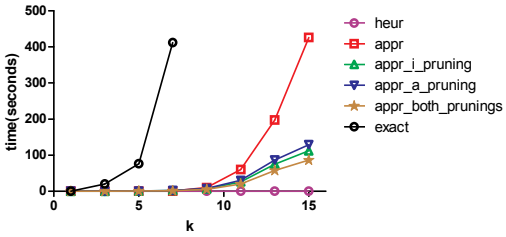


Fig. 2. Efficiency of k-HIT Selection Algorithms

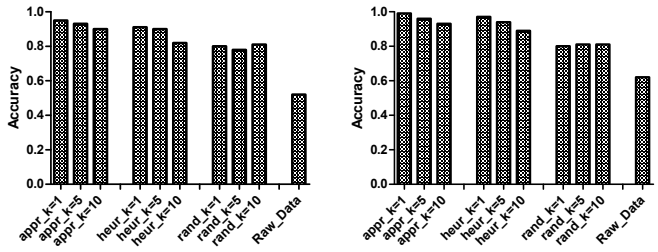
provide conflicting information, we consider the information of each conference as an x-tuple. More details can be found in [32].

For each x-tuple in these two data sets, we manually label the ground truth. In *Address*, we manually determine the best tuple according to the raw data (i.e. plain text); in *Call-for-paper*, we use the data from the official websites of the conferences as ground truth.

For each data set, each worker is required to take an unpaid *Qualification Test* containing 10 sample questions. A worker is accepted only if he/she correctly answers at least 6 questions in the qualification test. Each worker is only allowed to answer one HIT from each x-tuple, but they may take multiple HITs from different x-tuples. In order to estimate the crowd's accuracies, we first publish 30 testing HITs for each data set. Then we estimate the crowd's accuracies with these testing HITs as well as the questions answered in the qualification tests. Initially, after the 30 testing HITs being finished, the crowd's accuracies on *Address* and *Call-for-paper* are 0.86 and 0.73, respectively. We use these two values to set up the crowd's accuracies for the data sets. At the end of experiments, crowd's accuracies on *Address* and *Call-for-paper* are 0.85 and 0.76, respectively. This suggests that members of the crowd, though far from perfect, were pretty good at completing their tasks correctly. We set the budget $B = 60$, and each HIT is awarded 0.06 USD for both data sets. We compare *appr*, *heur* and *rand* with $k = 1, 5, 10$. The average performances are demonstrated in Figure 4. In terms of the improvement of data quality, one can see that the performance is basically consistent with the simulation - *appr* and *heur* outperform *rand* in all cases in terms of the number of HITs. Please note a budget is set up for each x-tuple, so users are allowed to set different budgets for different data instances. For the scalability concerning the number of x-tuples, if the same budget is assumed to be set for all x-tuples, the total number of HITs is simply equal to the 'number of x-tuples' multiplies the 'budget', i.e. linear w.r.t the number of x-tuples.

Data Accuracy Improvement:

Lastly, we verify the correctness of our approaches, by evaluating the accuracy of the best tuple in each x-tuple, i.e. the tuple with the highest possibility after cleaning. The accuracy is the ratio between the number of best tuples that are the same as the ground truth and the total number of x-tuples. The data accuracy improvement is illustrated in Figure 3. Please note that "Raw_Data" denotes the data accuracy before cleaning. One can see that the data accuracy is significantly improved after cleaning. Moreover, for *appr* and *heur*, the lower value of k tend to have higher accuracy. This is because smaller k



(a) Address (b) Call-for-paper

Fig. 3. Data Accuracy Improvement

indicates better HIT selections, but longer latency.

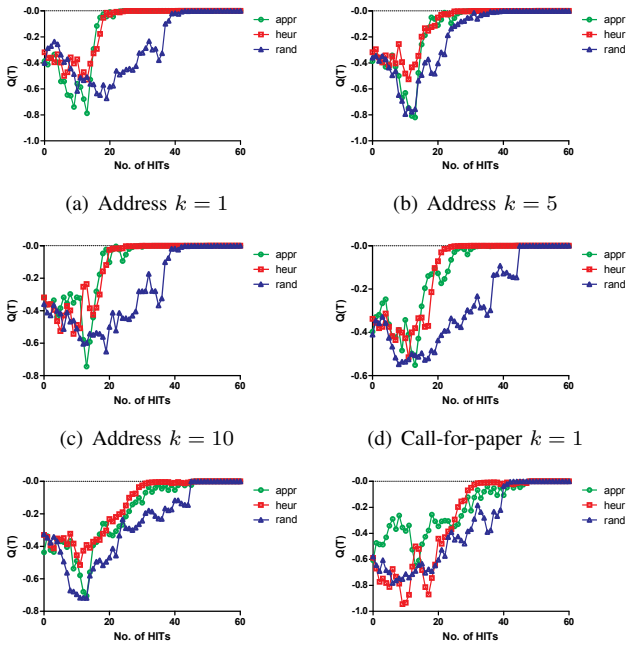
VII. Related Works

A. Cleaning Uncertain Data

Data cleaning techniques have been developed for decades by the database community [11], [28]. In particular, there is a number of works studying the issues of cleaning uncertain data [35], [16], [5], [23]. Given that a limited amount of resources to clean the database, [5] describes a technique for choosing the set of uncertain objects to be cleaned, in order to achieve the best improvement in the quality of query answers. They develop a quality metric, namely PWS-quality, for a probabilistic database, and they investigate how such a metric can be used for data cleaning purposes. Please note PWS-quality is essentially the negative value of Shannon entropy, which is the same metric we adopt in this paper. In [23], the authors extend the approach of [5] to support top-k queries. [16] provides an analysis on the sensitivity of a probabilistic query answer to the input data. [35] applies user feedbacks to improve the quality of an uncertain database integrated from different sources. This paper distinguishes itself from the existing works in two perspectives as follows. First, we consider the crowd as a special resource for cleaning uncertain data. The speciality includes 1) the crowd works best when the tasks are broken down into small pieces; and 2) crowd may provide wrong information. Second, we focus on cleaning individual data instances (i.e. x-tuples), not query answers. By improving the quality of data instances, any related queries can be benefited.

B. Crowdsourcing

The recent development of crowdsourcing brings us a new opportunity to engage human intelligence into the process of answering queries (see [8] as a survey). Crowdsourcing provides a new problem-solving paradigm [3], [21], which has been blended into several research communities, including database and data mining. In the practical viewpoint, [9] proposed and develop a query processing system using microtask-based crowdsourcing to answer queries. Moreover, in [25], a declarative query model is proposed to cooperate with standard relational database operators. In addition, in the viewpoint of theocratical study, many fundamental queries have been extensively studied, including filtering [24], max [14], sorting [22], join [37], etc. Besides, crowdsourcing-based solutions of many complex algorithms are developed, such



(a) Address $k = 1$ (b) Address $k = 5$

(c) Address $k = 10$ (d) Call-for-paper $k = 1$

(e) Call-for-paper $k = 5$ (f) Call-for-paper $k = 10$

Fig. 4. Testing on Amazon Mechanical Turk

as categorization based on graph search [26], clustering [13], entity resolution [36], [38], tagging in social networks [6], trip planning [17], topic discovery [31] etc.

In the recent work [12], the authors use crowdsourcing techniques to improve the results of information extraction systems. Our work is essentially different from [12] discussed as follows. First, during the selection of HITs, [12] uses the 'token entropy' and 'mutual information' to select HITs, but does not consider the error rate of the crowd, which is one of the core challenges addressed in our model. Second, [12] only focuses on data from information extraction systems, and their methods cannot be used when the uncertainty is caused by other reasons (e.g. information conflicts among different websites in the *call-for-paper* data set in Section VI-C).

VIII. Conclusion

In this paper, we propose utilizing the power of crowdsourcing on cleaning uncertain data. In many circumstances, the uncertainty is caused by the fact that machines cannot fully capture the human-intuitive semantics (e.g. images, human languages). Therefore, we are motivated to use crowdsourcing to eliminate the uncertainty of the data. In particular, we first prove that the crowd, although noisy, is a profitable approach for cleaning uncertain data. Then, we optimize the approach by efficiently identifying the most valuable set of questions (HITs).

Uncertainty is inherited in many modern applications, such as information extraction, entity resolution, schema matching, and truth discovery. We believe that adopting crowdsourcing as a new component of a DBMS would be extremely conductive for the improving the quality of uncertain data,

hence effectively improve the overall performance. Our work represents an initial solution towards cleaning uncertain data with crowdsourcing.

ACKNOWLEDGEMENT

This work is supported in part by the Hong Kong RGC Project N_HKUST637/13, National Grand Fundamental Research 973 Program of China under Grant 2014CB340303, NSFC Grant No. 61328202, Microsoft Research Asia Gift Grant and Google Faculty Award 2013.

REFERENCES

[1] Parag Agrawal, Omar Benjelloun, Anish Das Sarma, Chris Hayworth, Shubha U. Nabar, Tomoe Sugihara, and Jennifer Widom. Trio: A system for data, uncertainty, and lineage. In *VLDB*, pages 1151–1154, 2006.

[2] Carlos Guestrin Andreas Krause. A note on the budgeted maximization of submodular functions. Technical report, School of Computer Science, Carnegie Mellon University, March 2005.

[3] Daren C. Brabham. Crowdsourcing as a model for problem solving an introduction and cases. *Convergence February 2008 vol. 14 no. 1 75-90*, 2008.

[4] Reynold Cheng. Querying and cleaning uncertain data. In Kurt Rothermel, Dieter Fritsch, Wolfgang Blochinger, and Frank Drr, editors, *Quality of Context*, volume 5786 of *Lecture Notes in Computer Science*, pages 41–52. Springer Berlin Heidelberg, 2009.

[5] Reynold Cheng, Jinchuan Chen, and Xike Xie. Cleaning uncertain data with quality guarantees. *PVLDB*, 1(1):722–735, 2008.

[6] Mahashweta Das, Saravanan Thirumuruganathan, Sihem Amer-Yahia, Gautam Das, and Cong Yu. Who tags what? an analysis framework. *PVLDB*, 5(11):1567–1578, 2012.

[7] Susan B. Davidson, Sanjeev Khanna, Tova Milo, and Sudeepa Roy. Using the crowd for top-k and group-by queries. In *ICDT*, pages 225–236, 2013.

[8] AnHai Doan, Raghu Ramakrishnan, and Alon Y. Halevy. Crowdsourcing systems on the world-wide web. *Commun. ACM*, 54(4):86–96, 2011.

[9] Amber Feng, Michael J. Franklin, Donald Kossmann, Tim Kraska, Samuel Madden, Sukriti Ramesh, Andrew Wang, and Reynold Xin. Crowddb: Query processing with the vldb crowd. *PVLDB*, 4(12):1387–1390, 2011.

[10] Avigdor Gal. Managing uncertainty in schema matching with top-k schema mappings. pages 90–114, 2006.

[11] V. Ganti. *Data Cleaning*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[12] Sean Goldberg, Daisy Zhe Wang, and Tim Kraska. Castle: Crowd-assisted system for textual labeling & extraction. In *HCOMP*, 2013.

[13] Ryan Gomes, Peter Welinder, Andreas Krause, and Pietro Perona. Crowdclustering. In *NIPS*, pages 558–566, 2011.

[14] Stephen Guo, Aditya G. Parameswaran, and Hector Garcia-Molina. So who won?: dynamic max discovery with the crowd. In *SIGMOD Conference*, pages 385–396, 2012.

[15] Rahul Gupta and Sunita Sarawagi. Creating probabilistic databases from information extraction models. In *VLDB*, pages 965–976, 2006.

[16] Bhargav Kanagal, Jian Li, and Amol Deshpande. Sensitivity analysis and explanations for robust query evaluation in probabilistic databases. In *SIGMOD Conference*, pages 841–852, 2011.

[17] Haim Kaplan, Ilia Lotosh, Tova Milo, and Slava Novgorodov. Answering planning queries with the crowd. *PVLDB*, 6(9):697–708, 2013.

[18] Samir Khuller, Anna Moss, and Joseph Naor. The budgeted maximum coverage problem. *Inf. Process. Lett.*, 70(1):39–45, 1999.

[19] Aniket Kittur, Boris Smus, Susheel Khamkar, and Robert E. Kraut. Crowdforge: Crowdsourcing complex work. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 43–52, New York, NY, USA, 2011. ACM.

[20] Xuan Liu, Meiyu Lu, Beng Chin Ooi, Yanyan Shen, Sai Wu, and Meihui Zhang. Cdas: A crowdsourcing data analytics system. *PVLDB*, 5(10):1040–1051, 2012.

[21] T.W. Malone, R. Laubacher, and C. Dellarocas. Harnessing crowds: Mapping the genome of collective intelligence. Research Paper No. 4732-09, MIT, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA, USA, February 2009. Sloan Research Paper No. 4732-09.

[22] Adam Marcus, Eugene Wu, David R. Karger, Samuel Madden, and Robert C. Miller. Human-powered sorts and joins. *PVLDB*, 5(1):13–24, 2011.

[23] Luyi Mo, Reynold Cheng, Xiang Li, David W. Cheung, and Xuan S. Yang. Cleaning uncertain data for top-k queries. In *ICDE*, pages 134–145, 2013.

[24] Aditya G. Parameswaran, Hector Garcia-Molina, Hyunjung Park, Neoklis Polyzotis, Aditya Ramesh, and Jennifer Widom. Crowdscreen: algorithms for filtering data with humans. In *SIGMOD Conference*, pages 361–372, 2012.

[25] Aditya G. Parameswaran and Neoklis Polyzotis. Answering queries using humans, algorithms and databases. In *CIDR*, pages 160–166, 2011.

[26] Aditya G. Parameswaran, Anish Das Sarma, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. Human-assisted graph search: it's okay to ask questions. *PVLDB*, 4(5):267–278, 2011.

[27] Hyunjung Park and Jennifer Widom. Query optimization over crowdsourced data. *PVLDB*, 6(10):781–792, 2013.

[28] Erhard Rahm and Hong Hai Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

[29] Anish Das Sarma, Aditya Parameswaran, Hector Garcia-Molina, and Alon Halevy. Finding with the crowd. Technical report, Stanford University.

[30] Parag Singla and Pedro Domingos. Entity resolution with markov logic. In *ICDM*, pages 572–582, 2006.

[31] Yongxin Tong, Caleb Chen Cao, and Lei Chen. Tcs: Efficient topic discovery over crowd-oriented service data. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 861–870, New York, NY, USA, 2014. ACM.

[32] Yongxin Tong, Caleb Chen Cao, Chen Jason Zhang, Yatao Li, and Lei Chen. Crowdcleaner: Data cleaning for multi-version data on the web via crowdsourcing. In *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*, pages 1182–1185, 2014.

[33] Yongxin Tong, Lei Chen, Yurong Cheng, and Philip S. Yu. Mining frequent itemsets over uncertain databases. *PVLDB*, 5(11):1650–1661, 2012.

[34] Yongxin Tong, Lei Chen, and Bolin Ding. Discovering threshold-based frequent closed itemsets over probabilistic data. In *IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012*, pages 270–281, 2012.

[35] Maurice van Keulen and Ander de Keijzer. Qualitative effects of knowledge rules and user feedback in probabilistic data integration. *VLDB J.*, 18(5):1191–1217, 2009.

[36] Jiannan Wang, Tim Kraska, Michael J. Franklin, and Jianhua Feng. Crowder: Crowdsourcing entity resolution. *PVLDB*, 5(11):1483–1494, 2012.

[37] Jiannan Wang, Guoliang Li, Tim Kraska, Michael J. Franklin, and Jianhua Feng. Leveraging transitive relations for crowdsourced joins. In *SIGMOD Conference*, pages 229–240, 2013.

[38] Steven Euijong Whang, Peter Lofgren, and Hector Garcia-Molina. Question selection for crowd entity resolution. *PVLDB*, 6(6):349–360, 2013.