On-Demand Query Result Cleaning

Ying Yang
Supervised by Oliver Kennedy and Jan Chomicki
State University of New York at Buffalo, Buffalo, NY, USA
{yyang25,okennedy,chomicki}@buffalo.edu

ABSTRACT

Incomplete data is ubiquitous. When a user issues a query over incomplete data, the results may contain incomplete data as well. If a user requires high precision query results, or current estimation algorithms fail to make accurate estimates on incomplete data, data collection by humans may instead be used to find values for, or to *confirm* this incomplete data. We propose an approach that incrementally confirms incomplete data: First, queries on incomplete data are processed by a probabilistic database system. Incomplete data in the query results is represented in a form called *candidate questions*. Second, we incrementally solicit user feedback to confirm candidate questions.

The challenge of this approach is to determine in what order to confirm candidate questions with the user. To solve this, we design a framework for ranking candidate questions for user confirmation using a concept that we call *cost of perfect information* (CPI). The core component of CPI is a *penalty function* that is based on *entropy*. We compare each candidate question's CPI and choose an optimal candidate question to solicit user feedback. Our approach achieves accurate query results with low confirmation and computation costs. Experiments on a real dataset show that our approach outperforms other strategies.

1. INTRODUCTION

RDBMSs assume data to be correct, complete and unambiguous. However, in reality, this is not always the case. Incomplete data appears in many domains including statistical models, expert systems, sensor data, and web data extraction. Without user interaction, incomplete data can be queried in several ways: (1) by treating incomplete data as NULLs, or (2) by estimating incomplete data through heuristic methods. In case 1, query results will not be complete and in case 2, automatic processes may not be able to achieve an accurate result. Human operators, however, can often accurately find the ground truth for incomplete data. People have started to realize the power of human computation through crowdsourcing services such as Mechanical Turk, oDesk, and SamaSource, which use human operators to confirm incomplete information. Examples include finding the book's authors for a picture of the book or matching earthquake survivor pictures with missing persons in Haiti [10, 11, 13]. For such tasks, reliable data collection by humans may be required to achieve accurate query results. In some domains, reliable measurement and expert knowledge are also needed to collect reliable data.

Proceedings of the VLDB 2014 PhD Workshop.

EXAMPLE 1. *A patient (Alice) comes to the doctor (Dave) and describes her symptoms. Since Alice has just come to hospital, her BloodSugar, BloodPressures and heart rate may not be known. Dave issues a query Q querying whether Alice has one or more diseases. The system then interacts with Dave to obtain the ground truths of the unknown values and eventually obtains an accurate query result. Below we show the dialogue between the system and Dave.*

System: Please input query:
Doctor: select Name,'HEART DISEASE'
from Table
where BloodPressureA >= 180 and BloodSugar >= 90
or HeartRate > 100
UNION select Name,'HIGH BLOOD PRESSURE'
from Table
where BloodSugar > 150 and BloodPressureB > 90
or KidneyDisease=true
UNION ···
(System calculating which variable to confirm by our algorithm...)
System: What is the value of BloodPressureA?
Doctor: 121.
System: Thank you. And what is the value of HeartRate?
Doctor: 110.
System: Alert! The patient has HEART DISEASE
What is the value of...

Traditionally, incomplete data is cleaned before query processing [15]. However, cleaning all incomplete data may require upfront time and cost when some of the data may not even be utilized in the query results. Therefore, our primary challenge is to achieve accurate query results with minimal confirmation cost through reliable data collection. To address this challenge, we propose to postpone uncertainty management until *after* query processing has completed. After query processing, the importance of each incomplete datum to a query result becomes clear. Incomplete data may disappear and contribute little or nothing to the incompleteness of the query results. On the other hand, some incomplete data are detrimental to the query result

accuracy. It is clear that we should put more effort and resources towards the latter case.

Current probabilistic database systems (PDBMS) provide query processing solutions without making completeness assumptions on data in the ways RDBMSs do [5, 8]. During query processing, many PDBMSs represent incomplete data as *boolean formulas* over atoms. Here, an atom stands for a property of incomplete data that can be either true or false. Boolean atoms in Example 1 include: "Is HeartRate > 100?", or "Does the patient have kidney disease family history?". Boolean atoms can be connected by logic connectives $\{\neg, \wedge, \vee\}$ into boolean formulas. In this paper, boolean atoms reference the incomplete data in the input, and contribute to the incompleteness of the query results. Confirming a boolean atom means obtaining the ground truth of the atom, which usually comes with a cost. The challenge is how to confirm boolean atoms to achieve accurate query results with minimal total confirmation cost. Confirming all boolean atoms will achieve accurate query results but at maximal confirmation cost. Since boolean atoms can be connected by logic connectives, the ground truth of a boolean atom can affect the utility of others. Confirming a fraction of boolean atoms can often achieve accurate query results with a lower confirmation cost and latency. For instance, consider the formula $(A > 180 \wedge B < 50) \vee C > 30$. If we know $A > 180$ is false, then there is no need to confirm B since confirming $\{A > 180, C > 30\}$ will be sufficient. Therefore, we want a form of incremental cleaning, where the system incrementally solicits the ground truths of boolean atoms as the user provides answers. One of the main challenges for soliciting user feedback is to decide in what order to confirm the boolean atoms.

We consider the problem of confirming boolean atoms to achieve an accurate query result with a low confirmation cost and a quick response time. We develop algorithms for ranking boolean atoms for confirmation using a novel decision theoretic concept that we call *cost of perfect information* (CPI). CPI provides a method to estimate the benefit obtained from a user confirming the ground truth of a boolean atom. At the core of CPI is a *penalty function* which quantifies the uncertainty of a query result. To decide which boolean atom to confirm, we need to compare each boolean atom's penalty function and choose the smallest one. Our approach has several benefits. In addition to an accuracy guarantee, we provide an incremental query result cleaning method with low confirmation cost and quick response time. In a large database or a database with frequently changing data, some incomplete data may never be used in queries or may be overwritten before use. Therefore, using our method saves confirmation cost compared to naively cleaning all data. For small and medium size databases, with low data volatility, in the extreme case all incomplete data will be confirmed by our method. Even in this case, compared to the naive total cleaning approach, we provide an accurate query result with low latency by confirming incomplete data related to the query result first. We strike a good trade-off between query result confirmation cost and computation cost. Furthermore, the proposed method gives users the flexibility to control the trade-off between computation time and confirmation cost. We have done experiments on real data and the results show that the rankings of incomplete data proposed by our algorithms yield lower confirmation costs than other approaches.

Q(D)	global conditions: (BloodPressureA >= 70 and BloodPressureA <= 190, BloodPressureB >= 40 and BloodPressureB <= 110)
Name	ϕ_{CQ}
Alice	(BloodPressureA >= 180 and BloodSugar >= 90 or HeartRate> 100) OR (BloodSugar > 150 and BloodPressureB > 90 or KidneyDisease=true)
Sally	BloodPressureB > 90

Table 1: Query result in Example 1

1.1 Research Questions

We have a probabilistic database D, defined over a set of *possible worlds*. Each possible world is a deterministic database instance. A user issues a query Q on D, and since D consists of a set of possible database instances, Q is evaluated in parallel on all instances in D. The query results Q(D) will be composed of a set of possible result instances as well. The goal is to reduce the number of result instances in Q(D). We model incomplete data in Q(D) as one or more candidate question formulas (ϕ_{CQ}). To obtain a deterministic Q(D), we need to confirm ϕ_{CQ}. A *candidate question formula* is formed by candidate questions connected by logic connectives. A *candidate question* (cq) is a boolean atom of the form x$\odot$y, x$\odot\{c\}$, where x and y are incomplete data, c is a constant and $\odot$ is a relational operator $\{>, <, \geq, \leq, =\}$[1]. *cq* has a ground truth value which is unknown. We are given a probability distribution for its possible values. Confirming a candidate question involves posing a natural language question that can be answered with "yes" or "no". All candidate questions in ϕ_{CQ} form a *candidate question set* (CQ). Candidate question formulas consist of all incomplete data in Q(D) denoted as CQ=$\{CQ_1, \cdots, CQ_n\}$ where n is the number of tuples in Q(D). Table 1 shows part of the query result by PDBMS in Example 1. Atoms in ϕ_{CQ} are candidate questions cq. All cq in ϕ_{CQ} constitute CQ, CQ_k is composed of cq in ϕ_{CQ} in row k. There is a cost associated with obtaining a ground truth, and confirming all CQ provides us with an upper bound on the total confirmation cost to achieve any desired accuracy for the query results. However, confirming a subset of CQ may be sufficient to achieve a desired accuracy. Hence, our approach is to confirm CQ incrementally. This approach makes use of the fact that some candidate questions provide more information (reducing uncertainty) to the query results than others. The challenge is to decide which cq_i provides the most benefit to the query results when confirmed. Hence, the problem we consider in this paper is ranking CQ to reduce the uncertainty of query results with minimal confirmation cost. We propose a decision-theoretic concept CPI, which determines the optimal candidate question oq:

$$oq = argmin_{cq_i}\ CPI(cq_i)$$

oq is utilized to solicit the ground truth from the user. D will be updated to D' based on oq's ground truth. This interaction terminates when a user determined precision of Q(D) is achieved.

In conclusion, the contributions presented in this work include:

- A mechanism to incrementally clean query results with low confirmation cost.

[1]More general formulas are possible, but beyond the scope of this paper.

- A penalty function that quantifies uncertainty of query results using the information theoretic concept of entropy.
- A decision-theoretic concept that we call the *cost of perfect information* (CPI). CPI measures the penalty of a query result, with confirmation cost as a bias.
- An approximation of CPI that allows a user to trade off between computation cost and confirmation cost.

2. BACKGROUND AND RELATED WORK

2.1 C-tables

C-table is short for conditional table [6]. A condition is a conjunct of equality atoms of the form $x = y, x = c$ and of inequality atoms of the form $x \neq y$, $x \neq c$, where x and y are variables and c is a constant. Boolean true and false can be respectively encoded as boolean atoms $x = x$ and $x \neq x$. Let ϕ be a boolean formula of atoms. A valuation v is a mapping of variables to constants. A valuation v satisfies ϕ if its assignment of constants to variables makes the formula true. A global condition ψ_T is associated with the entire table T and limits the set of valid valuations to those that satisfy ψ_T. A local condition ϕ_t is associated with tuple t of T. Table 1 shows part of the C-table obtained in example 1. An example of a global condition is in Table 1, and local conditions are the ϕ_{CQ} cell in each row. A tuple t exists in the query result if ϕ_t is evaluated to be true and does not exist if false. PSUJ-queries are closed over C-tables.

The probabilistic database component of our system is implemented based on C-tables with some generalization: local conditions are extended to arithmetic formulas over atoms and constants, or boolean formulas of the same.

2.2 Uncertain applications

[17] is an overview of the representation formalisms and query processing techniques for probabilistic data. Trio [1] introduced the concept of an Uncertainty-Lineage Database, a database with both uncertainty and lineage. A recent work is MayBMS [5], which presents a confidence estimation for tuples in query results. A general probabilistic database system, PIP [8] uses symbolic representations of probabilistic data to defer computation of expectations, moments and other statistical measures until the expression to be measured is fully known.

Jeffery et al. [7] present a decision-theoretic framework to determine the order in which to confirm schema matching and reference reconciliation candidates to provide more benefit to a dataspace. Our work is potentially more general and can confirm any incomplete data that can be expressed in our version of C-tables. We could treat a schema matching candidate as one form of candidate question and apply our mechanism to rank these candidate questions together with other candidate questions for user confirmation. In this way, our framework measures the benefit of confirming different types of incomplete data in a uniform way. Our framework is more general because, besides schema matching, we can also confirm missing data values to obtain accurate query results. A recent work [3] presents several cleaning algorithms without assuming cleaning is perfect. Our work is orthogonal to this work and it is query-result oriented. We can provide accurate query results with shorter response time than confirming all incomplete data before query processing. Raman and Hellerstein [16] presented an interactive data cleaning system that integrates transformation and discrepancy detection. Bahar and Mirek [14] considered uncertainty in query and data, and proposed a probability-based framework that reduces the uncertainty. Our work differs in that we consider budgets when cleaning query results. Given a limited budget, our system can optimize its utilization. Our work can be applied to cost-sensitive systems, for example crowdsourcing platform such as Mechanical Turk.

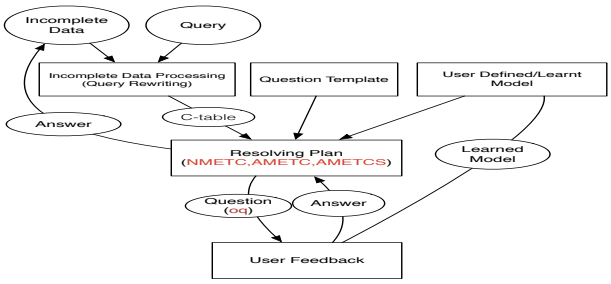

Figure 1: Data flow in our system.

2.3 Cost-sensitive decision theory

Enlightened by the idea that in decision trees, we tend to greedily choose the feature that yields the highest information gain, we studied research on inducing decision trees that considers the cost of acquiring features. A survey by Lomax et al. [9] presented over 50 algorithms on cost-sensitive decision tree learning. ID3 provides a measure to define the expected entropy for an attribute. C4.5 uses an information gain ratio to reduce the influence by the bias towards attributes that have more values. CS-ID3 [18] and CS-C4.5 [4] utilize the same information gain formulas as ID3 and C4.5 respectively, however, they include cost of attributes to bias the measure towards selecting attributes that cost less and considering the information grain at the same time. EG2 [12] adopted a user-defined parameter ω that varies the extent of the bias. In our approach, entropy and cost are also considered as factors to determine the ranking of incomplete data for user confirmation. However, the above solutions cannot be directly adopted to solve our problem. In the above solutions, there is a training data set and each training data consists of a set of attributes and a corresponding decision. A decision tree is trained using training data and used for decision-making on the data that has the same data distribution as training data. Choosing an attribute as current node in the tree do not affect the existence of other attributes. In our work, the incomplete data in query results is represented as boolean formulas and the ground truth of one incomplete data can affect the existence of other incomplete data.

3. FRAMEWORK

Figure 1 shows the system workflow.

- **Incomplete Data Processing (IDP)**:
  When a user issues a query on incomplete data, the system processes the query as a normal PDBMS and returns a C-table as output.
- **User Defined/Learned Model (UDLM)**:
  In our version of C-tables, incomplete data is represented as candidate questions. We want to use any existing models on the incomplete data to confirm the

ground truths of the candidate questions with minimal confirmation cost. If a user has rules for confirming candidate questions without extra confirmation cost, they can be predefined in the UDLM. For example: "candidate question A always returns true if B is false". We can thus obtain the ground truth of A from B if B false. These models can be treated as global conditions ψ_T and utilized to reduce the number of query result instances to achieve accurate query results.

- **Question Template (QT)**: The QT component manages question templates that translate candidate questions into natural language forms. For some candidate questions, the total cost of confirming them together is smaller than confirming them separately. For instance, the total cost of confirming systolic and diastolic blood pressure is half the cost of confirming them separately, whereas the total cost of confirming blood sugar and urine are the same as the cost of confirming them separately. A user could group candidate questions into one question, denoted as $\langle(cq_1, cq_2, \cdots, cq_n)$,C$\rangle$, where $(cq_1, cq_2, \cdots, cq_n)$ are a group of candidate questions and C is the total cost of confirming them together. In future work, we will consider how users could provide this type of templates and the group questions could be treated as one candidate question to minimize total confirmation cost.

- **Resolving Plan (RP)**: After obtaining C-tables from the IDP, the RP component fetches input from the UDLM and the QT for pre-confirming candidate questions at no cost. Then the RP calculates optimal candidate questions oq and formulates natural language questions using question templates in the QT. After obtaining the ground truth of oq, RP updates the initial incomplete data, and iterates the whole precess until the desired query result precision is achieved.

- **User Feedback Interface (UF)**: After a candidate question is translated into a natural language question, the UF interface presents the question to the user and receives the true answer. Then incomplete data is updated given the ground truth. As future work, we will consider to learn models from the user-system interaction and store them in the UDLM for future use.

3.1 Ordering Candidate Questions

In this paper we focus on the RP phase, where it is necessary to select an optimal question. We consider several algorithms: First, naive minimum expected total cost (NMETC) method. This serves as a lower bound for confirmation cost. Then we introduce an information-theoretic approach to ranking candidate questions for user feedback which is called *approximate minimum expected total cost (AMETC)*. Also, we devise a Monte-Carlo version of AMETC which we call *approximate minimum expected total cost with sampling (AMETCS)*. For simplicity, we initially confirm the candidate question formula ϕ_{CQ} in each tuple independently. We assume formulas do not share common candidate questions and each tuple contributes the same value to the query results. In section 3.2, we generalize our method to avoid making such assumptions.

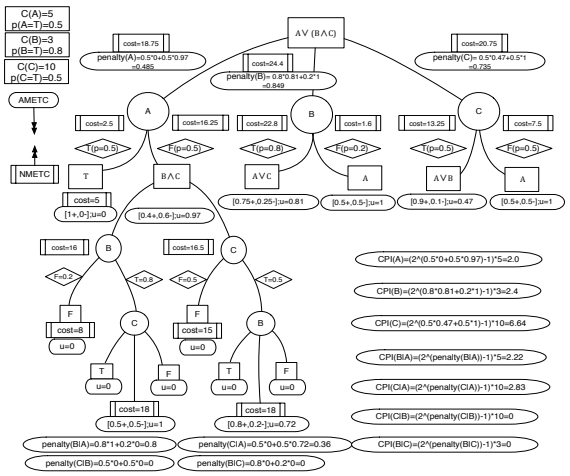

Figure 2: NMETC and AMETC example, rectangles represent boolean formulas; circles represent candidate questions; diamonds represent probability distributions of the ground truths of the candidate questions; squashed rectangles represent functions in AMETC, subprocesses represent expected total confirmation cost in NMETC.

3.1.1 NMETC-Naive Minimum Expected Total Cost

A *valid candidate question path* is an ordered candidate question set CQ with a given ground truth for each candidate question. We can obtain the deterministic query results if evaluating based on the ground truths given by the path. One naive solution to ranking candidate questions is to calculate the expected total cost for all possible valid candidate question paths and choose the candidate question set in the path with minimal expected total cost. As shown in Figure 2, we construct a tree structure to illustrate the process. A formula node lists all its candidate questions as child nodes. Each circle node of the tree represents a candidate question, which has two branches denoting two sets of possible worlds from the two possible ground truths of the candidate question. We start from the leaves of the tree, then climb all the way up to the root. During this process, the expected total cost at each node is calculated as weighted sum of its children's confirmation cost. After visiting all nodes in the tree, we start from the root and choose the circle node that returns the minimum expected total cost.

The time complexity of this method is exponential in the number of candidate questions. Although the method has high complexity, it gives us a lower bound of the confirmation cost we can reasonably expect. We use this method for a confirmation cost comparison with other methods.

3.1.2 AMETC-approximate expected total cost

We develop an information-theoretic concept, we call the *the cost of perfect information* (CPI), which quantifies the penalty of a query result after a candidate question is confirmed, with confirmation cost as a bias. In what follows, we illustrate how CPI can be utilized to measure the benefit of confirming a candidate question.

Suppose we are given a probabilistic database D, and a query Q, a candidate question formulas in Q(D) is denoted as ϕ_{CQ}. We assume that there is some means of measuring the uncertainty of a query result, denoted as U(Q(D)), which

will be introduced shortly. Given a candidate question cq_i, if the system solicits the ground truth of cq_i, there are two possible results, each with their possible world: a possible world in which cq_i is confirmed and the other possible world in which cq_i is disconfirmed. We denote the uncertainty in two possible worlds by $U(Q(D\backslash cq_i^+))$, $U(Q(D\backslash cq_i^-))$, where $D\backslash cq_i^{+/-}$ represents D after cq_i is confirmed or disconfirmed. Furthermore, we assume the probability of cq_i confirmed to be true is p_i, the penalty of a question result after cq_i is confirmed can be expressed by the penalty function Pen:

$$Pen(cq_i) = U(Q(D\backslash cq_i^+)) \cdot p_i + U(Q(D\backslash cq_i^-)) \cdot (1 - p_i) \quad (1)$$

Pen measures the expected uncertainty of query results if we confirm cq_i. A lower penalty indicates the query result is more certain.

The key to computing the penalty function is to determine $U(Q(D\backslash cq_i^+))$ and $U(Q(D\backslash cq_i^-))$. $U(Q(D\backslash cq_i^{+/-}))$ quantifies the uncertainty of an updated query result and we want them to be 0 if confirmation requires no further questions, that is, the query result achieves 100% accuracy. They approach 1 if the query result is very uncertain. As an uncertainty metric, we use entropy:

$$U(Q(D)) = P \cdot log(P) + (1 - P) \cdot log(1 - P) \quad (2)$$

Here we assume that we can approximate the probability of a possible world P, where P denotes p(ϕ_{CQ}=true).

This penalty function presents us the uncertainty of the query results and we want to bias the measurement towards achieving less uncertainty with less confirmation cost. Hence, inspired by the algorithm EG2 [12], we define the cost of perfect information as follows:

$$CPI(cq_i) = (2^{Pen(cq_i)} - 1) \cdot C_{(cq_i)} \quad (3)$$

where $C_{(cq_i)}$ is the cost of confirming cq_i. This approach is also illustrated in Figure 2. We start from the root and greedily compute CPI for candidate questions in the root's children. As seen in Figure 2, candidate question A has the smallest CPI, A is the optimal question. Therefore, we consider the branch where A is the root. First, we confirm A for the ground truth and update Q(D). If Q(D) is still not sufficiently accurate, we keep greedily calculating CPI for all the root's children until achieving the desired accuracy.

3.1.3 AMETCS-approximate minimum expected total cost with sampling

When calculating U(Q(D)) in AMETC, the most time consuming part is to calculate the probability distribution of the question results P. Sampling according to the distribution of candidate questions reduces the computation time. Furthermore, the users can have more control of the computation time and the query result accuracy. More samples lead to higher precision of the query results, but cause longer computation time.

3.2 Ordering K-Candidate Question Sets

Previously, we assumed that candidate question formulas for each tuple t do not share candidate questions in common and each tuple has the same importance to the query results. We now generalize CPI to work across multiple tuples by a weighted sum of the independent penalties.

$$Pen'(cq_i) = \frac{U(Q(D\backslash cq_i^+)) \cdot p_i + U(Q(D\backslash cq_i^-)) \cdot (1 - p_i)}{W} \quad (4)$$

Min Cost	METC	AMETC	AMETCS	Random
112.1729	151.2639	151.2639	174.3010	226.2738

Table 2: Average cost of querying diseases together.

$$Pen(cq_i) = \sum_{j}^{|f_{cq_i}|} Pen'(cq_i) \quad (5)$$

where f_{cq_i} denotes the formulas that contain cq_i, $|f_{cq_i}|$ denotes the number of these formulas, p_i is cq_i' s probability distribution in f_{cq_i}. f_{cq_i} is in a tuple t, and W is the importance of the tuple t to the query results.

4. EXPERIMENTS

We have conducted experiments using the UCI machine learning Dermatology Data Set [2]. It contains 34 attributes which are used to diagnose erythematosquamous diseases. The data set was split into training and testing data. We have used the training data to construct queries representing diagnosis of diseases and testing data to run experiments. We have constructed a classification tree object using training data. Each node in the tree is an attribute. A set of attributes together classify the training data into corresponding diseases. We have obtained the set of attributes for each disease and used them to construct a query for each disease. Then the queries were issued on the test data. An example query might be:

```
select 'lichen planus' from Testdata
where f20 < 0.5 and f27 < 0.5 and f15 < 0.5 and
f5 < 0.5 and f7 < 1.5 and f26 < 0.5 and
f22 < 1.5 and f30 < 2.
```

where f_i is the i_{th} attribute. We have selected eight attributes that are involved in the five diseases and randomly replaced 80% of their data values with variables representing incomplete data. Incomplete data is 19% of the whole training data set. The probability distribution of a candidate question has been estimated by the distribution that the ground truth takes of the whole answer set. There are two kinds of attributes: clinical attributes and histopathological attributes. The values of clinical attributes are easy to obtain, therefore, we set the confirming cost of candidate questions formed by those attributes to 1 while histopathological attributes are comparatively difficult to obtain and we set their confirmation cost to 30 based on the laboratory examination cost.

We have compared the average cost obtained by each method. In Figure 3, the min cost method assumes we know the ground truths of all candidate questions and chooses the optimal candidate question path to solicit user feedback. This is impossible in practice, but it serves as an optimal lower bound on confirmation cost. NMETC measures the best possible performance we can achieve without knowing the ground truths of candidate questions, which is a practical lower bound. From the result we can see the performance of AMETC and NMETC are very close. For AMETCS, the number of samples is half of the number of valid candidate question paths. The average cost of AMETCS increases compared to AMETC. However, as we show in Figure 4, the computation time decreases to almost half of AMETC. Thus AMETCS is suitable for big data cleaning.

Since diseases may share common attributes and we want to query if a patient has any of five diseases to save diagnosis

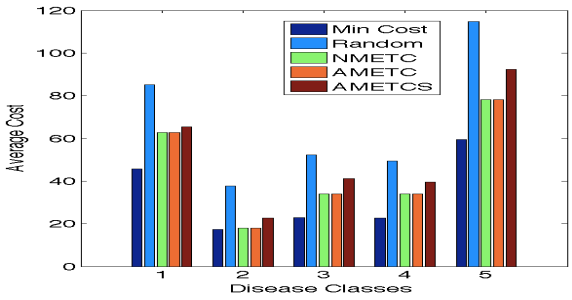

Figure 3: Average cost of each formula

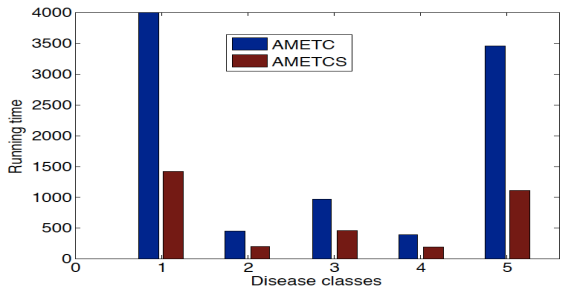

Figure 4: Algorithms running time

time. We have issued a query that considers all five diseases together. We have applied the modified penalty function (5) in CPI. As the result shows in Table 2, each cost is smaller than the sum of querying each disease separately and the patient and doctor's time are saved.

5. RESEARCH PLAN

To overcome the assumptions and limitations of the current framework we propose the following research plan: (1) We want to generalize candidate questions to include non-boolean atoms. In this way, same incomplete data in different candidate questions can be confirmed in one confirmation round. (2) The current framework assumes the probability distributions for incomplete data are given. However, this is not always feasible. We want to make use of the existing reliable data to estimate the probability distribution of incomplete data. (3) In many domains, there exist models that can be utilized to increase query result accuracy without consulting users. Currently, in the UDLM component, we mainly utilized the user-defined models to preprocess incomplete data at no cost. We want to explore how to learn models from system-user interactions. For instance, in data integration, possible schema matches are candidate models, one of which is a correct match. We can treat each candidate model as probabilistically related to the real model. We can integrate these candidate models with our current framework and adjust the confidence of a model on confirmation. The probability will converge on true model. This learned model will be utilized adaptively in subsequent queries. (4) Our current framework considers user feedback is always correct. However, humans may make mistakes. In crowdsourcing, humans may provide incorrect answers to tasks accidentally or on purpose. In medical expert systems, user feedback is obtained based on test results by machines. The test results may not be 100% accurate. We want to build a user profile to record the credibility of user feedback. We need to decide how to integrate this profile with our framework and based on the correctness of the decisions made based on the feedback, update the credibility of the profile.

6. CONCLUSION

This paper proposes a mechanism to obtain accurate query results from incomplete data with low confirmation cost and quick response time. We presented a framework based on incremental soliciting of user feedback. As part of the framework, we developed a penalty function that measures the uncertainty of the query results. We introduced a concept called CPI which considers confirmation cost to bias the penalty measurement towards smaller-cost candidate questions. We described experiments on a medical dataset and showed our approach outperformed other strategies.

7. REFERENCES

[1] P. Agrawal, O. Benjelloun, A. D. Sarma, C. Hayworth, S. Nabar, T. Sugihara, and J. Widom. Trio: A system for data, uncertainty, and lineage. In *VLDB*, 2006.

[2] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[3] R. Cheng, E. Lo, X. S. Yang, M.-H. Luk, X. Li, and X. Xie. Explore or exploit?: Effective strategies for disambiguating large databases. 2010.

[4] A. Freitas, A. Costa-Pereira, and P. Brazdil. Cost-sensitive decision trees applied to medical data. In *Data Warehousing and Knowledge Discovery*. 2007.

[5] J. Huang, L. Antova, C. Koch, and D. Olteanu. Maybms: A probabilistic database management system. In *SIGMOD*, 2009.

[6] T. Imieliński and W. Lipski, Jr. Incomplete information in relational databases. *J. ACM*, 1984.

[7] S. R. Jeffery, M. J. Franklin, and A. Y. Halevy. Pay-as-you-go user feedback for dataspace systems. In *SIGMOD*, 2008.

[8] O. Kennedy and C. Koch. Pip: A database system for great and small expectations. In *ICDE*, 2010.

[9] S. Lomax and S. Vadera. A survey of cost-sensitive decision tree induction algorithms. *ACM Comput. Surv.*, 2013.

[10] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller. Demonstration of qurk: a query processor for humanoperators. In *SIGMOD*, 2011.

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

[12] M. Núñez. The use of background knowledge in decision tree induction. *Mach. Learn.*, 1991.

[13] A. G. Parameswaran and N. Polyzotis. Answering queries using humans, algorithms and databases. 2011.

[14] B. Qarabaqi and M. Riedewald. User-driven refinement of imprecise queries. In *ICDE*, 2014.

[15] E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. 2000.

[16] V. Raman and J. M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, 2001.

[17] D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases*. 2011.

[18] M. Tan and J. C. Schlimmer. Cost-sensitive concept learning of sensor use in approach and recognition. In *MLSB12*, 1989.