UGuide – User-Guided Discovery of FD-Detectable Errors

Saravanan Thirumuruganathan Laure Berti-Equille Mourad Ouzzani
Jorge-Arnulfo Quiane-Ruiz Nan Tang
Qatar Computing Research Institute
HBKU, Research Complex, P.O. Box 5825, Doha, Qatar
{sthirumuruganathan, lberti, mouzzani, jquianeruiz, ntang}@hbku.edu.qa

ABSTRACT

Error detection is the process of identifying problematic data cells that are different from their ground truth. Functional dependencies (FDs) have been widely studied in support of this process. Oftentimes, it is assumed that FDs are given by experts. Unfortunately, it is usually hard and expensive for the experts to define such FDs. In addition, automatic data profiling over dirty data in order to find correct FDs is known to be a hard problem. In this paper, we propose an *end-to-end* solution to detect FD-detectable errors from dirty data. The broad intuition is that given a dirty dataset, it is feasible to automatically find approximate FDs, as well as data that is possibly erroneous. Arguably, at this point, only experts can confirm true FDs or true errors. However, in practice, experts never have enough budget to find all errors. Hence, our problem is, given a limited budget of expert's time, which questions we should ask, either FDs, cells, or tuples, such that we can find as many data errors as possible. We present efficient algorithms to interact with the user. Extensive experiments demonstrate that our proposed framework is effective in detecting errors from dirty data.

1. INTRODUCTION

High quality data is important for any data intensive application. However, data often contains a number of inaccuracies and other discrepancies that have significant impact on decision making. Hence, there has been extensive research [1, 12, 13, 20] on identifying and repairing data errors. Functional dependencies (FDs) [2] and its extension conditional functional dependencies (CFDs) [14] have been widely used to detect data errors.

However, in a number of real-world scenarios, a comprehensive set of FDs that can detect all errors does not really exist. Hence, the problem of finding *all FD-related errors* of a dirty dataset T is hard. There are three intuitive solutions: (1) Automatically profile T to find FDs [26]; (2) Automatically profile a clean sample T' of T to find FDs; and (3) Discover approximate FDs on T and then ask the expert to manually pick the valid FDs. In this case, the expert is kept in the loop. Evidently, solution (1) is deemed to fail as one cannot find FDs from a dirty data T and then use them to clean T. Solution (2) is likely to fail as well, because in practice, getting such a sample is expensive and worse, it is hard – if not impossible – to ensure that the provided sample is representative. Finally, solution (3) shifts the burden to the expert, which is also not practical since the number of approximate FDs can be very large and certainly beyond the capacity of the expert.

SIGMOD'17, May 14-19, 2017, Chicago, IL, USA
 ISBN 978-1-4503-4197-4/17/05…frm[o]–5.00
DOI: http://dx.doi.org/10.1145/3035918.3064024

1.1 Problem

An interesting yet practical requirement is: Given a dirty dataset T, identify a small set of FDs that can detect most of the FD errors in T. However, we are not provided with a comprehensive set of FDs and there is no automatic solution that is able to detect all FD-related errors. Hence, we could instead try a hybrid approach where an expert is involved to remove the ambiguity of decisions that an automatic algorithm will face. However, in practice, experts are expensive and should be used judiciously. More concretely, the problem we address in this paper is: Given a dirty dataset T and a budget B of questions to submit to an expert, *how to detect as many FD-related errors as possible?*

1.2 Our Methodology

Obviously, detecting data errors using FDs is a chicken and egg problem as FDs discovered from erroneous data can be as erroneous as the data themselves. Our approach to solve this problem consists in smartly leveraging the feedback of the user with limited involvement. As illustrated in Figure 1, given a dirty dataset with its corresponding set of discovered FDs and a fixed budget of questions that can be submitted to the user, we propose effective algorithms that maximize the number of errors that are FD-detectable. Overall, our algorithms rely on two key observations:

(a) We are mostly interested in rules that detect actual errors in the data rather than the myriad of rules that just hold on the data. That is, we are interested in correct FDs, but if no error exist on those FDs, then they are not of interest to us.

(b) There exists a heavy overlap between the errors identified by various FDs. In other words, the same data error could be identified by multiple data quality rules [1]. As the key objective is to detect data errors, it is not necessary to validate all data quality rules that (almost) hold on the data as they do not contribute much to error detection. Also, even when considering the subset of FDs that is vio-

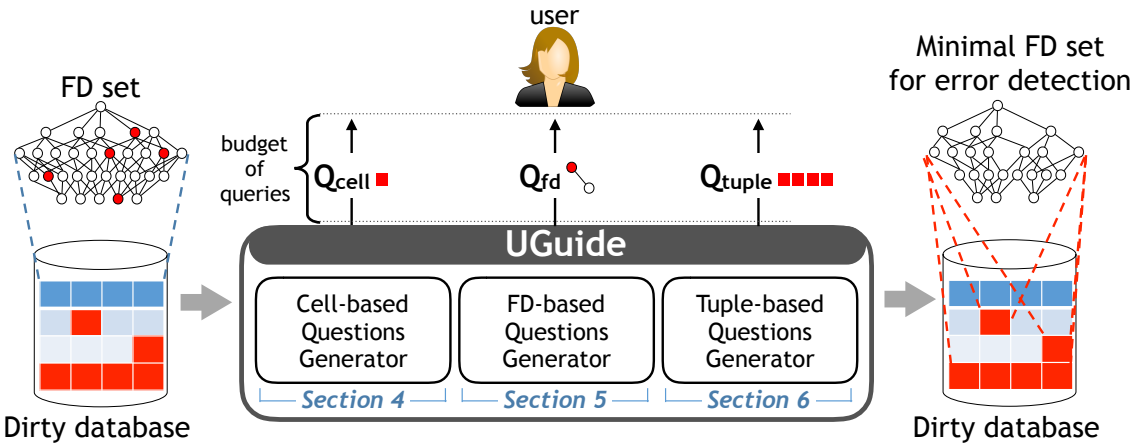


Figure 1: Overview of our approach.

lated by a number of tuples, it is not necessary for the expert to validate each of them. Instead, it is sufficient to validate a smaller fraction of the FDs that jointly "cover" the set of all violations.

We identify three fundamental types of questions that could be posed to the user: cell-based, tuple-based, and FD-based questions. For each of these three types of questions, the set of FDs that can capture the maximal number of errors under a certain budget of queries submitted to the user is computed.

Cell-based questions. The expert is shown a cell (an attribute and its value) and asked if it is erroneous or not. We formulate a weighted hitting set-based solution where the confidence of the FDs that can help in detecting erroneous cells is computed and recursively updated each time the expert gives a new label for a cell (true, false, or unknown).

FD-based questions. The expert is shown an FD and asked if it is a valid FD. We formulate the problem as a classical combinatorial problem called "budgeted max coverage" where the objective is to find a small number of sets with maximum cumulative weight that covers a set of elements under a fixed budget. In our context, each FD corresponds to a set where the set of violations it can identify constitutes its elements. The weight of the set is the cost of soliciting the user for validating that FD.

Tuple-based questions. The expert is shown an individual tuple and asked if the tuple has any erroneous cells. Our algorithms leverage the dual relationship between a set of FDs and Armstrong relations. We seek to obtain from the expert, a small set of clean tuples ($T_S \subset T$) with an interesting property – the set of exact FDs that hold over T_S have a substantial overlap with the set of FDs discovered over the clean version of T (say T_{clean}) and has a minimal number of false positive FDs (those that hold over T_S but not T_{clean}). We develop an efficient sampling-based algorithm for identifying T_S.

1.3 Contributions

We summarize our major contributions hereafter:

- We study the problem of detecting the set of FD-detectable errors under a fixed budget (Section 2). We advocate for a solution strategy that focuses on identifying the subset of FDs that can detect errors in contrast to identifying all the FDs (Section 3).
- We consider three fundamental type of questions that could be asked to the expert (Sections 4, 5 and 6) and propose algorithms that provide optimal solutions under the budget constraint for each type of questions.
- We conduct extensive experiments over multiple real-world and synthetic datasets that show the efficiency and efficacy of our solutions (Section 7).

2. PROBLEM FORMULATION

2.1 Preliminaries

Let T represent a database relation with schema $R = \{A_1, A_2, \ldots, A_m\}$. Let $X \subseteq R$ be a subset of attributes. We represent the projection of T to X as $T[X]$ while the projection of a tuple t to a set of attributes X as $t[X]$. If X and Y are two sets of attributes, XY denotes the set $X \cup Y$.

Functional Dependencies: A functional dependency (FD) $X \rightarrow Y$ over a set of attributes $X, Y \subseteq R$ states that X functionally determines Y. An FD is satisfied by T if $\forall t_1, t_2 \in T$, if $t_1[X] = t_2[X]$ then $t_1[Y] = t_2[Y]$. X is the LHS (determinant) of the FD while Y is the RHS (dependent). An FD is violated if there exists at least one pair of tuples (t_1, t_2) such that $t_1[X] = t_2[X]$ but $t_1[Y] \neq t_2[Y]$. Armstrong axioms are a set of sound and complete axioms covering the implications of FDs. The set of all errors that can be determined through FDs over T is denoted by $\mathcal{E}_{\mathcal{F}}$. T_C is the cleaned version of T after all the errors in $\mathcal{E}_{\mathcal{F}}$ are fixed. $\mathcal{F}_T$ is the set of FDs discovered from the relation T.

Types of FDs: An FD $X \rightarrow A$ is *minimal* if no subset of X determines A, *i.e.,* removing any attribute from X makes the FD invalid. In this paper, we restrict ourselves to FDs that are *non trivial*, *i.e.,* $X \cap Y = \varnothing$, and *normalized*, *i.e.,* the RHS is a single attribute. By applying Armstrong axioms, we can demonstrate that any relation that satisfies an FD $X \rightarrow A_1A_2$ also satisfies $X \rightarrow A_1$ and $X \rightarrow A_2$. Given a set of minimal, non-trivial FDs, we can infer all the other FDs that hold on T through the Armstrong axioms. Specifically, we can see that given an FD $X \rightarrow A$, any subset of X makes the FD invalid while a superset of X is a non-

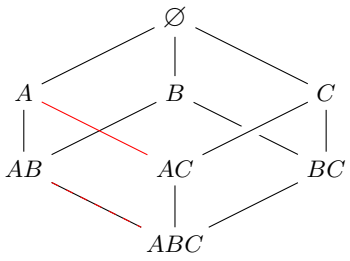


Figure 2: Attribute Lattice for FD Discovery

minimal FD that can be inferred. Specifically, if $X \rightarrow A$ holds then $XY \rightarrow A$ also holds.

Approximate Functional Dependencies: An approximate FD (AFD) $X \rightarrow A$ is a dependency that holds on most tuples of T and is violated by a small fraction of T. To better define the concept of approximation, violating tuples are used to calculate a satisfaction error [23]. In this paper, we consider the g_3 satisfaction error metric that corresponds to the smallest number of violating tuples that need to be deleted from the table so that the AFD exactly holds and has no violations. If the g_3 score of an AFD is less than a given threshold ϵ, we consider the AFD to be approximately satisfied. Please refer to [23] for a detailed discussion about FDs and various algorithms for discovering them.

Representing FDs as a Lattice: The set of all possible FDs can be modeled as a power set lattice over attributes. The lattice [23] is a directed graph where each node represents a unique set of attributes. Nodes at level i contains all sets of attributes of size exactly i. The root node, level-0, corresponds to an empty attribute set denoted by $\varnothing$. The first level has m nodes corresponding to each of the m attributes $A_1, A_2, \ldots, A_m$, and so on. Two nodes in adjacent levels are connected if their corresponding sets of attributes are subset or superset of each other. Each edge in the lattice corresponds to a potential FD. Figure 2 shows an example of an attribute lattice with three attributes. Given the edge $AB - ABC$, AB is the parent node of ABC and the this edge represents one possible FD $AB \rightarrow C$.

Combinatorics of Functional Dependencies: Consider the attribute lattice in Figure 2. We can make the following observations. The i-th level of the lattice consists of $\binom{m}{i}$ nodes. The total number of nodes in the lattice is $\sum_{i=0}^{m} \binom{m}{i} = 2^m$. Each node has on average $m/2$ edges. Since each edge corresponds to a candidate FD, there are $2^m \cdot m/2$ candidate FDs in total. Given an attribute A_i, the number of candidate FDs with A_i as RHS is 2^{m-1}. This can be obtained by observing that the power set of the rest of the $m-1$ attributes is 2^{m-1} and for each of the set X in the powerset, we can construct an FD of the form $X \rightarrow A_i$. Using a similar argument, the number of nodes in the lattice containing attribute A_i is 2^{m-1}.

Types of Questions: We consider the following three types of questions to be submitted to the user who will ultimately validate the correctness of the data or/and the correctness of the dependencies by providing a "yes", "no" or "I don't know" response. In the case of a "no" answer, the user does not have to make any correction. To help the user in the validation, we also provide some context around the data or the dependencies.

1. *Cell-based questions:* "Is a particular value (or cell) of a given tuple's attribute erroneous?" Possible contexts are: the complete tuple where the cell appears; FDs for which the cell causes violations; and a sample of the tuples causing a violation with the cell in question.
2. *Tuple-based questions:* "Is a given tuple clean from *any* erroneous cell?" The user answers affirmatively if the tuple does not violate any FD or has correct values in every cell. Possible contexts include violated FDs and tuples with which the current tuple causes a violation. This context can be dynamic depending either on the cell selection or on the tuple being validated.
3. *FD-based questions:* "Is a given FD valid?" Possible contexts include a sample from the sets of violating and non-violating tuples.

Cost Model for Questions: An effective cost function for answering a question should quantify the effort provided by the user. Different types of questions might impose different costs in terms of user's effort. In fact, even within a single question type, different questions might have different costs. For example, the cost of a question for validating an FD with 2 attributes on the LHS and the one for validating an FD with 10 attributes on the LHS are different. Designing an appropriate cost function for expert interaction requires a more systematic and domain-dependent user study, which is beyond the scope of the paper. We assume that all of our algorithms can accept a black-box cost function and structure the expert interaction accordingly. Specifically, we assume that we are provided with a cost function that returns a cost associated to each question submitted to the user. In addition, the black-box function must satisfy some natural constraints – such as the cost being deterministic and positive (*i.e.,* non zero).

Utility of Question Types: The three types of questions have diverse trade-offs between the user's effort and the information that can be leveraged by our algorithms. Consider the cell-based question for which a pair (or set) of tuples with one highlighted cell are shown to the user. If the appropriate contextual information is given, this question type is often the simplest one to answer. However, the amount of information it reveals is not substantial. If the expert validates a violation, our confidence in the FD that could have identified this violation increases. However, it is possible that none of such FDs are useful FDs, *i.e.,* valid and relevant both for the domain expert and for error detection. Tuple-based questions are the most expensive case in practice (according to our cost function), especially if the table has a large number of attributes. However, the information they provide when the tuple is validated is substantial. Note that a relation with a single tuple can represent all possible $2^m \cdot m/2$ candidate FDs. As more tuples are added to this relation, it can concisely represent the (possibly exponential) set of satisfied FDs. Ideally, it might be possible to identify a small subset $T_S \subset T$ such that the set of FDs that hold over T_S is very close to those that hold over T. The FD-based question falls in between these two extremes. A single FD can concisely represent a large number of data errors as violations. Often, it is also possible that the user could readily answer whether a minimal FD over some key attributes is valid.

2.2 Problem Statement

In this paper, we address the problem of discovering FD-detectable errors with limited user involvement and we formulate it as follows. Given a fixed budget in terms of the number of questions that can be asked to a user, we want to maximize the number of erroneous cells that can be identified through FDs while minimizing false positives.

More formally, given a dirty table T with $\mathcal{E}_{\mathcal{F}}$, the set of errors that can be identified by FDs ($\mathcal{E}_{\mathcal{F}} \subseteq \mathcal{E}_{all}$ with $\mathcal{E}_{all}$, the set of all erroneous cells), a class of question types $\mathcal{Q}$, a cost function $C : q \in \mathcal{Q} \rightarrow \mathbb{R}$, and a budget B that limits the number of questions that the user can afford, *the budgeted FD error detection problem* aims at selecting a set of questions of type $\mathcal{Q}$ with cumulative cost of at most B that maximizes the number of detected violations from $\mathcal{E}_{\mathcal{F}}$ while minimizing the number of false positives.

3. FD-DETECTABLE ERRORS WITH BUDGETED USER FEEDBACK

We begin by providing a high level overview of our workflow and a meta-algorithm that will be instantiated by the latter sections. Next, we provide a formulation of our problem in graph theoretic terms that highlights its inherent combinatorial challenges.

3.1 Overall Workflow

Let T be a dirty table and T_C be its corresponding "clean" version. Let Σ_{T_C} be the set of FDs that hold on T_C. If we issue Σ_{T_C} over T, we can identify the set of FD-detectable errors over T, $\mathcal{E}_T$. Unfortunately, neither T_C nor Σ_{T_C} is available to us and hence there is no fully automatic way to identify Σ_{T_C}.

We begin by identifying a set of candidate FDs through the following simple observation. Without loss of generality, consider a minimal FD $\varphi : A \rightarrow C$ that holds on T_C. There are two scenarios: (a) φ either holds on T or (b) a specialization of it (say $AB \rightarrow C$) holds on T. Any other scenario is not feasible. Hence, if we discover the set of FDs that hold on T, each FD in this set must either be an FD in T_C or a specialization of it. Our process for generating the set of candidate FDs is as follows: (a) we run exact FD discovery over T to obtain Σ_T (b) for each $\sigma \in \Sigma_T$, we relax it by systematically removing one attribute at a time from its LHS till it is violated by more than a fixed threshold (say 10% of the tuples). We can see that each of the relaxed FDs do not violate more than 10% of the tuples. Let the set of relaxed FDs be Σ_{cand}. With an appropriate threshold, we can ensure that all FDs in Σ_{T_C} are contained in Σ_{cand}. On the flip side, there might be a number of FDs in Σ_{cand} that are not in Σ_{T_C}. We dub the later as false positive FDs as they identify some violations that are not true. Our objective now is to interact with the expert in an effective manner so that we obtain a smaller subset $\Sigma' \subseteq \Sigma_{cand}$ such that it has a high overlap with Σ_{T_C} (i.e., true FDs) and a low set difference (i.e., false FDs). Algorithm 1 provides the pseudocode for this process.

3.2 Bipartite Graph-based Interpretation

We now have three sets of FDs: Σ_{T_C} and Σ_T, that hold on T_C, T, respectively, and Σ_{cand}, the relaxed FDs from Σ_T. Let the set of violations detected by the Σ_{T_C} and Σ_{cand} *on* T be $\mathcal{E}_{T_C}$ and $\mathcal{E}_{cand}$. As $\Sigma_{T_C} \subseteq \Sigma_{cand}$, we have $\mathcal{E}_{T_C} \subseteq \mathcal{E}_{cand}$. Under a bounded budget, our objective is to ask a series of selected questions so that the number of true violations detected is maximized and the number of false positive detections is minimized.

Algorithm 1 Meta Algorithm

1: Get the set of candidate FDs and identify the set of candidate violations
2: **while** budget has not been exhausted **do**
3: Identify the next question that provides maximum information
4: Update the set of candidate FDs and candidate violations
5: **return** candidate FDs

Consider a bipartite graph where nodes in the left partition correspond to FDs in Σ_{cand} and nodes in the right partition correspond to violations in $\mathcal{E}_{cand}$. An edge connects an FD-node φ to a violation node v if the φ can detect v. Our objective is to determine a small number of FD nodes that maximize the number of violation nodes in $\mathcal{E}_{T_C}$ while minimizing those in $\mathcal{E}_{cand} \backslash \mathcal{E}_{T_C}$. We can also notice that the FD-based questions ask experts based on the RHS of the bipartite graph while cell-based questions ask on nodes on the LHS of the graph.

One of the advantages of the bipartite graph formulation is that it immediately highlights the computational challenges of our problem. Specifically, we can see that even a simplified version of the problem is NP-Complete. Given a set of FDs Σ_{T_C} and the corresponding violations on T namely $\mathcal{E}_{T_C}$, the minimum FD-error cover problem asks if there is a set of at most k FDs $\Sigma' \subseteq \Sigma_{T_C}$ with $|\Sigma'| \leqslant k$ that can detect all violations in $\mathcal{E}_{T_C}$. This is equivalent to the *set cover* problem where, given a universe $\mathcal{U}$ of items and a set $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ with each $S_i \subseteq \mathcal{U}$, the objective is to obtain a set $\mathcal{S}' \subseteq \mathcal{S}$ with $|\mathcal{S}'| = k$ such that their union is $\mathcal{U}$. This problem is known to be NP-Complete [16].

4. INSTANCE-BASED QUESTIONS

In this section, we consider the cell-based questions where the user is shown a single cell, *i.e.,* the value of a specific attribute A_i of a tuple t (i.e., $t[A_i]$), and asked if it is erroneous. The user is also shown some additional context such as one or more candidate FDs that the cell violates along with other tuples with the same value as the LHS of the FD but with values different from the RHS of $t[A_i]$. The user answers in the affirmative if the cell violates one or more FDs and negatively otherwise. The expert can answer "I don't know" if she is not sure of the response. We point to some interesting connections between our problem and the well-studied problems of weighted hitting set and truth discovery and we propose two interactive algorithms for asking effective cell-based questions.

Informativeness of cell-based questions: Compared with FD and tuple based questions, Cell-based questions often provide the least amount of information. Cell-based questions are often the easiest to answer – so easy that it can even be answered by non-expert users and could be effectively "crowdsourced" to the typical consumers of the data. In addition to this appealing property, there does exist scenarios where cell-based questions may provide substantial information. For example, consider a cell that is labeled as

Algorithm 2 Cell-Q-Hitting-Set

1: Σ, $\mathcal{E}_\Sigma$ be set of candidate FDs and violations respectively
2: $\forall f \in \Sigma$, $w(f) = 0$
3: $\forall v \in \mathcal{E}_\Sigma$, $w(v) = \frac{\sum_{f \in \Sigma(v)} w(f)}{|\Sigma(v)|}$
4: **while** budget B has not been exhausted **do**
5: $v \leftarrow \arg\min_{v \in \mathcal{E}_\Sigma} \frac{w(v)}{|\Sigma(v)|}$
6: Validate v with the user
7: **if** v is a true violation **then**
8: $\forall f \in \Sigma(v)$, $w(f) = w(f) + \delta$
9: Recompute the weight of affected violations
10: **else if** v is not a violation **then**
11: $\Sigma = \Sigma \backslash \Sigma(v)$ and update $\mathcal{E}_\Sigma$
12: $\mathcal{E}_\Sigma = \mathcal{E}_\Sigma \backslash \{v\}$
13: **return** Σ

erroneous by multiple candidate FDs. If the expert certifies it as erroneous, the information provided is small and our confidence in each of those FDs increases. On the other hand, if the expert certifies that the cell is not erroneous, the provided information is substantial. We can now categorize all those FDs as invalid instead of the much more expensive alternative of validating each and every FD with the expert (through FD-based questions).

An algorithm for selecting an effective set of cell-based questions must balance the two conflicting objectives. On the one hand, a violation identified by multiple candidate FDs is more likely to be a true violation as only one of those candidate FDs must be true for the violation to be true. On the other hand, the validation by the expert can provide a substantial payoff by single-handedly removing many false positive FDs. The two algorithms that we propose next seek to achieve such a balance.

4.1 Hitting Set-based Approach

(Weighted) Hitting set is a classical problem in complexity theory [16]. The input consists of a $\mathcal{U}$, the finite "universe" of elements and $\mathcal{S}$ which is a set of subsets of $\mathcal{U}$. Each element in $\mathcal{U}$ has also a weight. The minimum weighted hitting set problem seeks to identify $\mathcal{U}' \subseteq \mathcal{U}$ such that the intersection between $\mathcal{U}'$ and each set in $\mathcal{S}$ is non-empty and the cumulative weight of elements in $\mathcal{U}'$ is minimized. It has been shown that a simple greedy algorithm provides an approximation ratio of $\log(|U|)$. The algorithm iteratively: (a) picks the element e that minimizes the ratio of e's weight to the number of sets in $\mathcal{S}$ containing e, and (b) removes all the sets containing e from $\mathcal{S}$.

Example 1: [Hitting Set] Let $\mathcal{U} = \{1, 2, 3, 4, 5\}$ and $\mathcal{S} = \{\{1, 2, 3\}, \{1, 3, 5\}, \{2, 5\}\}$. Let the weights be $w(1) = 1, w(2) = 1, w(3) = 1, w(4) = 1$, and $w(5) = 2$. Both $\{1, 2\}$ and $\{1, 5\}$ are hitting sets but $\{1, 2\}$ is the optimal answer because its weight is 2 vs. 3 for the latter. □

Our Approach: Our problem of selecting a small set of violations can be reformulated as a hitting set problem as follows. The set of all violations corresponds to the universal set $\mathcal{U}$. The set of candidate FDs corresponds to $\mathcal{S}$ with the i-th set S_i being the set of violations identified by the i-th FD. By formulating it as a hitting set problem, we can identify a small set of violations that can potentially invalidate all the FDs in the best case. Realistically however, the expert can mark a violation as a true violation or as a false one. If a violation v_i is marked as true, then we must increase our confidence in the FDs that identified v_i and must avoid asking another v_j that is also identified by the same set of FDs. This can be achieved by assigning weights to each violation.

Our algorithm operates as follows. We initialize the weight of all candidate FDs to 1 (the minimum confidence) and the weight of all violations to the average of the weights of the FDs that identify it. Hence, the weight of each violation is also initialized to 1. We now pick the violation that minimizes the ratio of its weight to the number of FDs that identify it and ask the expert. If it is not a violation, we remove all FDs that identified it. Otherwise, we increase the confidence of each FD that identified it by a small parameter δ (say 0.1). The weight of each violation is then recomputed. These two steps are repeated till the budget is exhausted. Algorithm 2 provides the pseudo-code.

Algorithm 3 Cell-Q-SUMS

1: Σ, $\mathcal{E}_\Sigma$ be set of candidate FDs and violations respectively
2: $\mathcal{C}_\Sigma$, $\mathcal{C}_\mathcal{E}$ be confidence of Σ, $\mathcal{E}_\Sigma$
3: $\forall f \in \Sigma$, $c^1(f) = 1$
4: $\forall v \in \mathcal{E}_\Sigma$, $c^1(v) = 1$
5: Update $\mathcal{C}_\Sigma$, $\mathcal{C}_\mathcal{E}$ using Estimate-Confidence
6: **while** budget B has not been exhausted **do**
7: Pick violation v that provides maximum information
8: Ask the expert
9: Update $\mathcal{C}_\Sigma$, $\mathcal{C}_\mathcal{E}$ using Estimate-Confidence

4.2 Truth Discovery-based Approach

While the hitting set-based algorithm works adequately in practice, its performance is hampered by the fact that it is not leveraging the recursive relation between confidence over candidate FDs and candidate violations. Specifically, we are more confident about a violation if it was detected by one or more FDs with high confidence. Conversely, our confidence over an FD rests on the fact that it identifies violations that we are confident about.

This recursive definition is closely related to the area of "truth discovery" (See [4] for a survey). Informally, in truth discovery, we are provided with a set of sources and a set of claims. Each source makes a set of claims and each individual claim could be made by a subset of sources. When different sources provide conflicting information about a claim, the objective of truth discovery is to identify the factual true claim. Note that a recursive definition naturally applies here - we are confident about the trustworthiness of a source that makes claims that we believe in while our belief over a claim is increased if it is made by many (possibly independent) trustworthy sources.

We adapt the SUMS algorithm from [28] for our FD-detectable error discovery problem. At a high level, our algorithm starts with a default confidence over both FDs and violations. We then recursively recompute the confidence over an FD as the sum of confidence of the violations and vice-versa. This process is repeated till the values converge. Algorithm 3 provides the pseudo-code.

Note that we use different formulas for computing the confidence of an FD and a violation. The confidence of a violation is simply the sum of the confidence of the FDs that

Algorithm 4 Subroutine Estimate-Confidence

Input: Σ, $\mathcal{E}_\Sigma$ be set of candidate FDs and violations respectively
Initialize $\mathcal{C}_\Sigma$, and $\mathcal{C}_\mathcal{E}$, the confidence values of the candidate FDs and violations
while values have not converged **do**
  $\forall f \in \Sigma$, $c^i(f) = \log|\mathcal{E}_f| \frac{\sum_{v \in \mathcal{E}_f} c^{i-1}(v)}{|\mathcal{E}_f|}$
  Normalize the confidence values for FDs
  $\forall v \in \mathcal{E}_\Sigma$, $c^i(v) = \sum_{f in \Sigma(v)} c^{i-1}(f)$
  Normalize the confidence values for FDs
  $i = i + 1$
return $\mathcal{C}_\Sigma$, $\mathcal{C}_\mathcal{E}$

identify it. However, we cannot use the equivalent definition for estimating the confidence of an FD due to a subtle issue. Consider a candidate AFD f that has a large number of violations (say up to 10%). It is likely that f is a false positive FD. However, f would routinely get high confidence in each iteration under the summation formula due to the large number of violations it identifies. By computing the average, we penalize FDs that may identify too many violations (unless most of them are also high confidence ones). On the other hand, the logarithmic factor balances the confidence by providing a boost to FDs that cause a large number of violations. Specifically, given two FDs with each having the same confidence but one identifying 10 violations but the other 100 violations, we prefer the latter by providing higher weight ($\log|\mathcal{E}_f|$). After each iteration, normalization of the confidence must be made so as to limit them to a range of 0 and 1. The simplest way for the normalization would be to divide each confidence value of an FD (resp. violation) with the maximum value of confidence value of the FD (resp. violation) in a given iteration. Once the budget is exhausted, we pick all the FDs that have a confidence value higher than a expert specified threshold (e.g. 95%). We treat these FDs as true FDs and use them to detect the violations in T.

5. FD-BASED QUESTIONS

We now consider the scenario where the expert is asked schema based questions in the form of FDs. Specifically, the expert is shown a (possibly non-minimal) FD and asked if it is a valid FD. The expert answers in the affirmative if the FD is valid and the violations that it determines are indeed errors. The expert answers in the negative if the FD is invalid and the violations it identifies are not errors. The expert can always respond as "I don't know" if she is not sure either way. We relate the problem of choosing a small set of FDs to validate with the expert that maximizes the detection of true violations to the classical problem of "budgeted maximum coverage" [21]. We propose an interactive algorithm that identifies the FD to be asked to the expert that provides the maximum information at each iteration.

Informativeness of FD-based questions: Note that FD-based questions are more informative than the cell-based ones. Informally, this is due to the fact that a valid FD can single handedly represent/identify many violations. Hence, we obtain the same amount of information by validating a single FD as against validating each and every one of its violations. However, this increased informativeness comes at a cost. While the cell based-question can be answered by general users, it often requires an expert to answer an FD-based question.

Desiderata for FD-based questions: Given a set of candidate FDs (that might be FDs or AFDs), choosing the right set of FDs to ask is challenging for two main reasons. First, as minimal FDs have a straightforward interpretation in the context of the application/business logic, one might think that asking minimal FDs is the best approach to follow. However, asking non-minimal FDs or those that are implied by other FDs might still have substantial benefits. For example, consider two FDs $\varphi_1 : A \rightarrow C$ and $\varphi_2 : B \rightarrow C$. Instead of asking the expert to validate two FDs, the expert could be asked to validate a single (implied) FD of the form $\varphi_3 : AB \rightarrow C$. Typically, the violations identified by φ_3 are the union of those identified by φ_1 and φ_2. Second, it is not trivial to determine which of the candidate FDs with bounded violations to validate with the expert. On one hand, one can think of validating candidate FDs with a small number of violations, *i.e.,* those where removing a small number of tuples makes them exact FDs. However, these do not provide much bang for the buck as the number of violations detected with expert interaction is small. On the other hand, validating a candidate AFD with a large number of violations is also not very helpful as it might be a false positive FD.

We would like to note that these and many other issues can be elegantly abstracted by using the following two functions: (a) a cost assignment function that takes an FD and returns the cost of asking the FD to the expert and (b) a ranking function that given two FDs with the same cost, ranks an FD that provides more information higher.

Our algorithm can accept any arbitrary cost assignment function that returns non-negative values for a candidate FD. Our ranking function is based on three factors: (a) succinctness that rewards FDs that are concise and have smaller number of attributes on the LHS. (b) preferentially reward FDs that are more likely to be accurate and hence less likely to be false positive and (c) given two similar FDs, rewards the FD that cover a larger number of violations to one that identifies lesser number of violations. This is due to the fact that our budget is limited and we would prefer to minimize the number of FDs validated with the expert.

Budgeted maximum coverage-based formulation: In the *budgeted maximum coverage* problem, we are given a universe of elements $\mathcal{U}$ and a set of sets $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ with each of $S_i \subseteq \mathcal{U}$. We are also given a cost function $c(\cdot) : \mathcal{S} \rightarrow \mathbb{R}$ and a weight function $w(\cdot) : \mathcal{U} \rightarrow \mathbb{R}$. In other words, each set has a cost and each element has a weight. The objective then is to choose a subset $\mathcal{S}' \subseteq \mathcal{S}$ such that the total weight of the elements in $\mathcal{S}'$ is maximized while the total cost of $\mathcal{S}'$ is minimized. [21] shows that a variant of a greedy algorithm that picks the set maximizing the weight of uncovered elements has an approximation ratio of $\frac{1}{2} \cdot 1 - 1/e$.

The translation between the two problems is straightforward. Note that in our problem, we are given a set of candidate approximate FDs Σ_T and their corresponding violations. $\mathcal{U}$ is the set of all violations while $\mathcal{S}$ is the set of all AFDs. Each AFD $\varphi \in \mathcal{S}$ is represented as a set containing all the violations it identifies. The cost of φ is given by the cost assignment function. We initially set the weight of violation $v \in \mathcal{U}$ to 1 and generalize it to arbitrary values later. Now, our objective is finding a small set of questions to ask

Algorithm 5 FD-Q-Budgeted-Max-Coverage

$\Sigma' = \varnothing$
while budget B has not been exhausted **do**
  Selected AFD $\varphi \in \Sigma$ that maximizes $\frac{w'(\varphi)}{cost(\varphi)}$
  Validate φ with expert
  if φ is valid **then**
    Remove all violations identified by φ
  $\Sigma = \Sigma \setminus \varphi$, $\Sigma' = \Sigma' \cup \varphi$
return Σ'

Algorithm 6 Tuple-Sampling-Uniform

$T_S = \varnothing$
while budget has not been exhausted **do**
  Randomly sample tuple t
  if expert certified t as clean **then**
    Add t to T_S
Discover FDs Σ_{T_S} that hold on T_S.
return Σ_{T_S}

the expert such that their cumulative cost is within a budget and they maximize the number of detected violations.

Note that the approximation algorithm in [21] is for the non-interactive scenario where the objective is to pick a set of elements. In our case, this process is interactive where we need to validate the chosen FDs with the expert. Algorithm 5 provides the pseudo-code for the adapted algorithm.

6. TUPLE-BASED QUESTIONS

We now consider the tuple-based questions where the user is shown a tuple and asked if it is erroneous. The user answers in the affirmative if the tuple does not contain any erroneous cells and in the negative otherwise. We identify some interesting connections to the theoretical notion of Armstrong relations and use it develop efficient algorithms.

Tuple-based questions are particularly effective whenever it is possible for the user to answer such questions for the following key reasons: (1) there might be scenarios when the user can be relatively confident about the cleanliness of a tuple, *e.g.,* the tuple might come from a reliable source and might be considered clean for all practical purposes; and (2) tuple-based questions provide substantial amount of information than either cell-based or FD-based questions. Similarly to FD-based questions that can concisely encode more violations than the cell-based questions, a small set of tuples can encode a large number of FDs that in turn encode an even larger number of violations.

Representing FDs using tuples: Informally, we can consider a relation R to represent *both* the set Σ of FDs that hold over it and the set $\hat{\Sigma}$ of FDs that do not hold. A relation R can represent an FD explicitly or implicitly. R *explicitly* represents a valid FD $\varphi \in \Sigma$ (say $X \rightarrow A$), when there exist two tuples $t_i, t_j \in R$ such that $t_i[X] = t_j[X]$ and $t_i[A] = t_j[A]$. Similarly, R explicitly represents an invalid FD $\varphi' \notin \Sigma$ ($Y \rightarrow B$) when there exists a pair of tuples that serve as a counter example to φ' with $t_i[Y] = t_j[Y]$ and $t_i[B] \neq t_j[B]$. This formulation can be made more precise through the notion of the Armstrong relation.

Armstrong relation: Let Σ be the set of FDs and Σ^* be the set of FDs that can be logically inferred from Σ through Armstrong axioms. A database relation R_A is an *Armstrong relation* for a set of FDs Σ iff R_A satisfies all FDs in Σ^* and no other FD.

The dual relationship between a set of FDs and a set of tuples drives the development of our algorithm. Note that we cannot directly leverage the idea of Armstrong relations as we neither have the set of "true" FDs nor a clean dataset to discover them from. Hence, we have to settle for an approximation. Our new objective then is to come up with a small set T_S of tuples that are certified clean by the user such that the set of FDs it represents overlaps substantially with the correct FDs.

The scourge of false positive FDs: A key challenge that our approach must overcome is minimizing the number of false positive FDs. Recall that T is the dirty table given to us and T_C is the corresponding (hypothetical) cleaned version of it. Let Σ_R be the set of FDs that hold over a relation R. For example, Σ_{T_C} is the set of FDs that hold over T_C. Consider a relation $T_S \subset T$ initially populated with a single tuple $t_i \in T$. We can see that it implicitly represents each one of the $2^m \cdot m/2$ candidate FDs (see Section 2) that hold in T_S. The set of FDs that hold on T_S but not on T_C (*i.e.,* $\Sigma_{T_C} \backslash \Sigma_{T_S}$) are false positive FDs as the violations they detect are not true violations. Now, let us consider what happens if we add another tuple $t_j \in T$ to T_S. The updated Σ_{T_S} is (probably substantially) smaller than $2^m \cdot m/2$ as t_j might act as a counter example to many of the candidate FDs. In effect, the number of false positive FDs drops when we added t_j to T_S. If we are not careful, Σ_{T_S} might contain many false positive FDs that in turn snowball into a large number of false violations. Hence, a good algorithm must identify the best set of tuples (under budget) that minimizes the number of false positive FDs.

Basic Approaches: We start by describing two simple approaches to construct the relation T_S. The first approach, we dub as `Tuple-Sampling-Uniform`, works as follows. It chooses a tuple t uniformly at random from T and validates it with the expert. If the tuple is clean, it is added to T_S and rejected otherwise. This process is repeated till the query budget is exhausted. Despite its simplicity, this algorithms works reasonably well in practice and it is not hard to see why. Let p be the probability of finding a clean tuple from T (or equivalently the fraction of clean tuples in T) and the cleanliness of the chosen tuples can be modeled through a binomial distribution. As an illustration, if p is 0.9 and we get 100 tuples, then we would expect 90(100×0.9) out of them to be clean with a variance of 9 ($100 \times 0.9 \times 0.1$). The flip side of this approach is that a small fraction (specifically $1 - p$) of our budget is spent on dirty tuples. Algorithm 6 shows the pseudocode.

An improved approach, `Tuple-Sampling-Violation-Weighting` works as follows. Given T, we discover approximate FDs that hold on at least a fraction p of T. For each tuple t, we compute the number of AFDs for which t is part of the minimal number of tuples to be deleted to make the AFD to an FD. We can now perform a weighted sampling where the probability of a tuple being picked is inversely proportional to the number of violations. It is easy to see that this approach reduces the number of dirty data shown to the expert. Algorithm 7 shows the pseudocode.

Algorithm 7 Tuple-Sampling-Violation-Weighting

$T_S = \varnothing$
Discover FDs Σ_T from T
Relax Σ_T to obtain a set of candidate FDs, Σ_{cand}
Compute the set of candidate violations, $\mathcal{E}_{cand}$
$\forall t \in T$, $w(t) = |\Sigma_{cand}|$ - number of FDs in Σ_{cand} it violates
Normalize the weights $w(t)$ for all tuples: $w(t) = \frac{w(t)}{\sum_{t' \in T} w(t')}$
while budget has not been exhausted **do**
  Let t be the tuple chosen by weighted sampling
  if expert certified t as clean **then**
    Add t to T_S
Discover FDs Σ_{T_S} that hold on T_S.
return Σ_{T_S}

FD Closures and Saturation Sets: Before we describe an improved approach, we need to define two key terms. Given a set Σ of FDs and a set of attributes $X \subseteq R$, the closure of X denoted as X^*, is the set of attributes (including X) that are directly or indirectly dependent on X. A set of attributes X is said to be *saturated* [6] iff $X^* = X$.

Example 2: [Saturation Sets.] Using an example from [6], if $\Sigma = \{B \rightarrow C, AC \rightarrow B\}$, then the saturated sets are $\{A\}$, $\{C\}$, $\{B, C\}$, and $\varnothing$. □

Sampling with Saturation Sets: `Tuple-Sampling-Violation-Weighting` suffers from a glaring deficiency. While it minimizes the number of dirty tuples to be validated by the expert, it does not contribute much to the reduction of false positive FDs. In other words, it might easily add some tuple t_i to T_S that does not serve as a counter example to one or more false positive FDs. Our next algorithm `Tuple-Sampling-Saturation-Sets` addresses this issue. It works based on the observation that was proved in [6]. If R_A is an Armstrong relation for a set of FDs Σ, then for every saturated set of attributes, it has at least two tuples t_i, t_j such that they agree on all the attributes for the saturated set and disagree on others. For example, given a relation with three attributes A, B, and C and a saturated set $\{B, C\}$, it means that the Armstrong relation must contain two tuples that have the same values for B and C but differ on A.

Algorithm 8 provides the pseudo-code. Informally, we start by discovering the FDs for T and compute the saturated sets $\mathcal{S}$. Then we randomly sample a tuple t weighted inversely based on its number of violations (of AFDs). If the tuple t satisfies some saturated set(s), we request the expert to validate it. If it is certified clean, we add t to T_S and remove the satisfied sets from $\mathcal{S}$. This process is repeated till the budget is exhausted.

Algorithm 8 Tuple-Sampling-Saturation-Sets

Discover FDs Σ_T from T
$\mathcal{S}$ = saturated sets of $\mathcal{F}_T$
$T_S = \varnothing$
while budget has not been exhausted **do**
  Randomly sample tuple t
  if t satisfies some saturated set in $\mathcal{S}$ **then**
    If certified clean, add t to T_S and remove saturated sets satisfied by t from $\mathcal{S}$
Discover FDs Σ_{T_S} that hold on T_S.
return Σ_{T_S}

7. EXPERIMENTS

7.1 Experimental Setup

Hardware and Platform: All our experiments were performed on a quad-core 2.2 GHz machine with 16 GB of RAM. The algorithms were implemented in Python. We used Metanome [25] data profiling tool for discovery of functional dependencies.

Datasets: We conducted our experiments over one synthetic and two real-world datasets. We use the *Tax* generator from [7] for generating our synthetic dataset. It consists of 100K tuples corresponding to taxpayer information such as name, gender, address, salary along with tax rates and exemptions. The second dataset *Hospital* consists of health-care provider information from USA Medicare scheme and has around 115K tuples. It has attributes such as provider/hospital name, addresses, and type of services provided. The third dataset is *SP stock* that was obtained by [10] with almost 123K tuples. It provides information about the historical performance of S&P 500 stocks. Each tuple contains information about a stock such as the date, ticker name, starting and closing price along with high, low prices and the volume of trade. The numbers of exact FDs on these datasets are 364, 83 and 56 respectively.

Error Generation: We used BART [3], a state-of-the-art system for benchmarking data-cleaning algorithms. Given a set of data quality rules, it provides highly customized mechanisms to introduce errors into clean databases. Given a dataset and a set of FDs that hold on them, we generated two types of dirty datasets. We start by setting the maximum number of tuples violating any FD to the default value 20%. As an example, a dataset with 100K tuples and 10 FDs would have 20K tuples that violate them. In the *uniform error model*, we apportion the violations to each FD uniformly. In the sample scenario, each FD will be violated, on average, by 2K tuples. In the *systematic error model*, the distribution of errors that were identified by the FDs are skewed with a small set of FDs identifying most of the errors. In the *random error model*, BART can generate random errors such as typos, duplicate values, and missing values. In our experiments, we generated for each clean dataset, 10 different dirty dataset counterparts that are detectable by FDs. Our experimental results are computed as the average of 10 runs over these dirty datasets. Unless specified, our experiments use the systematic error model that is more representative of real-world errors.

Algorithms: We tested 4 algorithms described in our paper: `CellQ-HS` (Algorithm 2) and `CellQ-SUMS` (Algorithm 3) for cell-based questions, `FDQ-BMC` (Algorithm 5) for FD-based questions, and `Sampling-Violation` (Algorithm 8) for tuple-based questions. We compared them against intuitive baselines for each case. Algorithm `CellQ-Greedy` greedily chooses cells that are violated by a large number of candidate FDs. This heuristic often works well in practice due to the fact that if the cell is indeed a true violation, it increases our confidence over a large number of candidate FDs. On the other hand, if the cell is

not a valid violation, then it can potentially eliminate a large number of FDs from consideration. Algorithm `FD-Q-Greedy` also works similarly by greedily picking FDs that can identify the largest number of violations with an analogous reasoning. For tuple-based questions, we consider algorithms `Tuple-Sampling-Uniform` and `Tuple-Sampling-Violation-Weighting` described in Section 6 as baselines.

For each of the three types of questions, we also consider hypothetical oracle-based baselines that are aware of the true FDs and the corresponding FD errors. Thus, these baselines can pick the optimal choice at each step as to what question to ask the expert. These baselines show that, for a fixed budget, our proposed algorithms achieve a true and false positive rate that is very close to the optimal algorithms. Given a budget, `FDQ-Oracle` seeks to identify the set of FDs that maximize the number of true FD errors. Similarly, `CellQ-Oracle` seeks to identify the minimum number of cell-based questions that maximizes the number of true FDs and minimizes the number of false positive FDs. Finally, algorithm `TupleQ-Oracle` seeks to identify the minimum number of tuples to validate with the expert so as to achieve a fixed false positive rate (recall that tuple-based questions always have 100% true positive rate).

Cost Model: Recall that our algorithms can work with any arbitrary black box cost function. For the purpose of our experiments, we use the following intuitive cost model. The cost of validating a single cell is 1 while the cost of validating a tuple is m (the number of attributes). The cost of asking an FD based question is $\alpha^k \times |LHS|$ where $|LHS|$ is the size of FD's determinant. Since all the FDs we are interested in are normalized, the RHS is always a single attribute. Given a (possibly non minimal) FD σ' and the corresponding minimal FD σ, k is the difference between the number of attributes on the LHS of σ and σ' respectively. α is a weighting factor that penalizes asking large non minimal FD questions. For example, given a minimal FD $\sigma : A \rightarrow D$ and $\alpha = 2$, the cost of asking $\sigma_1 : A \rightarrow D$, $\sigma_2 : AB \rightarrow D$ and $\sigma_3 : ABC \rightarrow D$ are 1, 4 and 12 respectively.

Workflow Simulation: We simulated the interaction with the user as follows. For each of the "clean" version of the dataset, we computed the set of true FDs Σ_{T_C} that hold over it. Given a cell-based question of the form $t[A_i]$ for some tuple t and attribute A_i, we verify if it violates any FD from Σ_{T_C}. If so, we respond that the cell is erroneous and as clean otherwise. Given a tuple, we respond as erroneous if it violates some FD in Σ_{T_C} and clean otherwise. Given an FD σ', we answer as affirmative if the FD is a minimal FD σ or a specialization of one. For example, given a true minimal FD of the form $A \rightarrow C$, we answer in affirmative for candidate FDs $A \rightarrow C$ and $AB \rightarrow C$. We do not assume that the user is aware of the principles of FD inference such as Armstrong axioms.

Performance Measures: For all algorithms, we measure their efficacy through two factors: (a) the number of true violations identified and (b) the cost imposed on the expert based on the different types of questions asked. In addition, we are also interested in secondary metrics such as false positive and false negative detections. A detection is said to be false positive if it does not violate any "true" FDs while it is said to be false negative if it violates some "true" FD but our algorithm determines it not to be a violation. For example, a false negative can occur if the budget has been exhausted before the corresponding FD could be validated by the user.

7.2 Experimental Results

7.2.1 Cell-based Questions

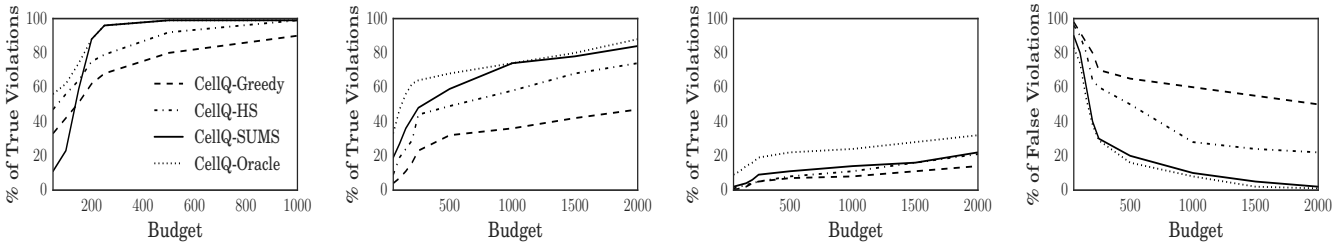
We first consider the cell-based questions where the user validates a set of cells that are then used to compute the confidence of the FDs that detected them. The user then treats all FDs with a confidence above a certain threshold (say 90%) as true FDs.

Figures 3(a)-3(c) shows the results for how the fraction of detected true violations varies with the budget. We see that, in contrast to a greedy algorithm, our algorithms quickly identify most of the true violations for a small budget. In fact, we observe that our algorithms can find a higher number of true violations while the greedy algorithm cannot. However, this detection comes with a cost. Our cell-based approach can produce too many false positive violations. Figure 3(d) shows that cell-based questions require a large number of queries before the number of false positives drops to acceptable values. Not surprisingly, the SUMS algorithm, which is based on truth discovery, performs best when the budget is limited. Furthermore, our algorithms achieve a true detection rate that is very close to the oracle-based baseline even for a limited budget.

7.2.2 FD-based Questions

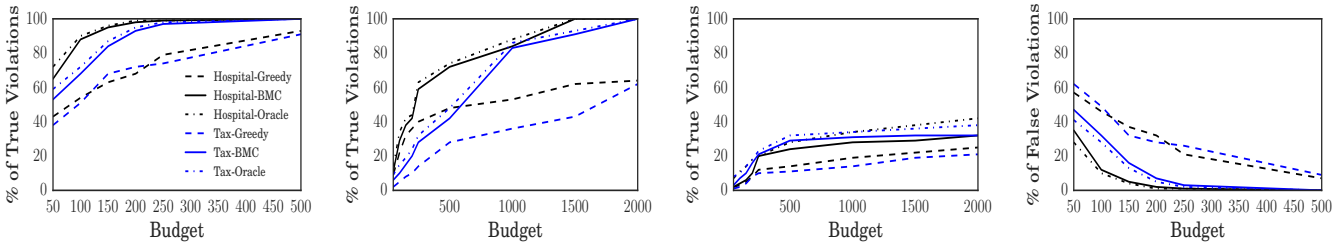
We next consider the FD-based questions. Recall that every violation identified by our FD-based question algorithms is a true violation as the corresponding FD was validated by the expert. To evaluate the effectiveness of our algorithms, we varied the budget and studied how it impacts the fraction of true violations that were detected.

Figures 4(a)-4(c) show the results for two datasets under systematic errors. Overall, we observe that, in each of the scenarios, our algorithm detects more violations for a fixed budget. Figure 4(a) shows that our algorithms quickly detect 100% of the true violations with a small budget. This is due to the fact that, in a systematic error model, some FDs identify most of the violations and our algorithm preferentially validates them first. Figure 4(b) shows the behavior for uniform errors where each FD is violated by a similar number of tuples. As expected, our algorithm incurs a higher cost as each of the FD needs to be validated in the worst case. However, our algorithm still outperforms the baseline because of two reasons: (a) it is budget-aware and hence detects more errors for a fixed budget and (b) it can ask the user to validate non-minimal FDs, which reduces the cost in some scenarios. The cost for validating Tax dataset is higher as it has more FDs (more than 300 compared to less than 100 for other datasets). Figure 4(c) shows the results for random errors. The number of detected violations is far smaller as most of the random errors (such as typos) are not detectable by FDs. Nevertheless, our algorithms identify the FD-detectable errors with limited budget. Finally, in contrast to cell-based algorithms, Figure 4(d) shows how the false negatives reduce dramatically when increasing the budget. Recall that in a systematic error model, few FDs could detect most of the errors. Our algorithms validates those FDs in early interactions with the expert. The rest of the FDs detect smaller and smaller number of errors resulting in a smaller detection utility with increasing budget.



(a) Budget vs. True Detections (Systematic errors) (b) Budget vs. True Detections (Uniform errors) (c) Budget vs. True Detections (Random errors) (d) Budget vs. False Detections (Systematic errors)

Figure 3: Performance of our algorithm for Cell-based Questions over Hospital Dataset (Legend for Figures 3(b)-3(d) are same as that of Figure 3(a))



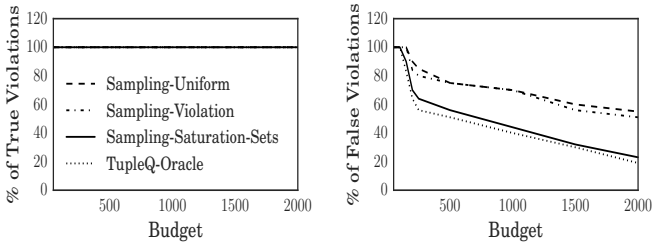
(a) Budget vs. True Detections (Systematic errors) (b) Budget vs. True Detections (Uniform errors) (c) Budget vs. True Detections (Random errors) (d) Budget vs. False Detections (Systematic errors)

Figure 4: Performance of our algorithm for FD-based Questions for Hospital and Tax datasets (Legend for Figures 4(b)-4(d) are same as that of Figure 4(a))

Once again, the set of FDs that are chosen to be verified with the expert are very similar for both the oracle-based baseline and our algorithms.

7.2.3 Tuple-Based Questions

We now consider the tuple-based questions. Figure 5(a) shows how the detection of true violations is affected by the budget. We observe that all tuple-based algorithms always detect 100% of the true violations. The reason is simple: given a subset T_S of clean tuples, the list of FDs that hold over T_S also holds over T_C, the clean version of the table. Hence, every violation in the table is always detected by the FDs. However, this appealing property comes with a flip side that is highlighted by Figure 5(b). These algorithms suffer from a large false positive detection rate. In particular, our algorithm suffers due to the challenges inherent in obtaining a small set of representative samples so as to have a perfect true violation detection but low false positive violations.



(a) Budget vs. True Detections (b) Budget vs. False Detections

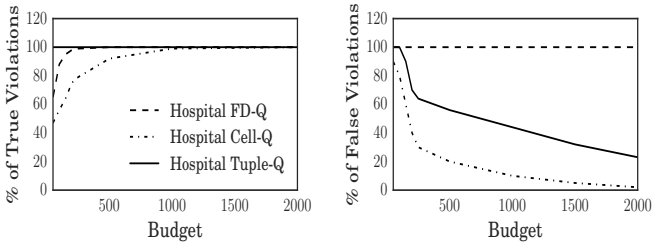
Figure 5: Performance of our algorithm for Tuple-based Questions over Hospital Dataset with Systematic errors (Legend for Figure 5(b) is same as that of Figure 5(a))

7.2.4 Comparative Performance of Algorithms

In this set of experiments, we evaluate how the FD-, cell-, and tuple-based questions compare against each other for a fixed budget. Figures 6(a)-6(b) shows the result. We observe that there is no single question type that serves as the silver bullet as each of them have their pros and cons. FD-based questions have the attractive property that they do not have any false positives, but might detect only a fraction of true violations for a small budget. On the other hand, tuple-based questions detect 100% of the true violations but suffer from a large false positive violation detection rate. The cell-based questions falls in the middle with better performance on false positive than tuple-based questions with worse performance on true violation detection.

7.2.5 Impact of Error Percentage

We proceed with an evaluation on how the fraction of tuples that are erroneous affect our algorithms. Here, we vary the number of erroneous tuples from 10% to 50%. How-



(a) Budget vs. True Detections (b) Budget vs. False Detections

Figure 6: Comparative performance of our algorithms over Hospital Dataset with Systematic errors (Legend for Figure 6(b) is same as that of Figure 6(a))

ever, each FD is still violated by at most 10% tuples. We evaluated all our algorithms for a fixed budget of 500.

Figure 7 report the results for detection of true violations. Tuple-based questions still detect 100% of all true violations. For a budget of 500, the impact on FD-based questions is minimal as the performance of FD-based algorithms are not dependent on the *number* of violations for each FDs. The impact on cell-based questions is also minimal as long as the proportion of violations between different FDs is fixed. Figure 8 reports the result for false detections. The increasing number of violations has an impact on the rate of false positive violation detections. This is due to the fact that tuple-based algorithms have to expend a substantial budget for finding each "clean" tuple. The false positive detections increases for cell based questions with increasing error percentage. This is due to the fact that the large number of cell violations make it harder to separate true FDs from false FDs which results in increased false positive violations. There is no impact on FD based questions as they do not lead to any false positive detections.

7.2.6 Impact of User Answers

The reader might now have the following question in mind: *What happens if the user cannot answer every question decisively?* We answer this question in this experiment. We simulated this scenario by making expert decline to answer a fixed fraction of questions as "I don't know" (IDK). We varied the rate from 50% to 100%.

Figure 9 shows the result. Note that the impact on tuple-based questions is substantial: the algorithm keeps asking the expert to validate tuples till she answers affirmatively. However, it has a deleterious effect on FD- and cell-based questions. For example, consider a set of errors that is identified by a single (minimal) FD. If the user answered "I don't know", the algorithm has to issue other non-minimal FDs while paying the penalty for asking such non-minimal FDs. The impact on cell-based is relatively minimal as each FD is often violated by a large number of cells and the algorithm can highlight other cells to determine if an FD is valid.

7.2.7 Runtime Performance

We would like to highlight that our focus is on detecting FD errors under a small budget. Most of the computationally expensive steps of our algorithms such as finding exact FDs over a dirty dataset and relaxing them to find approximate FDs with bounded violations can be considered as a preprocessing step. Hence, the key performance measure is the time taken from the moment the user answers a question to the moment the next question is asked.

Figure 10 shows the runtime performance of our algorithms per user interaction for the synthetic Tax dataset where we varied the number of tuples. As expected, the impact of tuple-based algorithms is minimal as the key step involves verifying if the chosen tuple satisfies any saturation set that is independent of the number of tuples. Similarly, the impact on FD- and cell-based questions is minimal or at most linear. Note that the time complexity of both the algorithms depends on the number of violations and not on the number of tuples or FDs.

7.2.8 Comparative Analysis of Algorithms

Let us summarize the relative strengths and weaknesses of all three approaches based on a number of dimensions.

1. **Expert Effort:** Cell based questions are often the easiest for an expert to answer. Validating the entire tuple might be more tiresome, especially if the relation has many attributes. FD validation falls in between.

2. **Fraction of True Violations Detected:** Under the perfect oracle model where the expert response is always correct, tuple based questions often have a 100% true violation detection rate. FD based questions have the next best performance, especially under the real-world scenario where few FDs can identify most of the errors. Cell based questions perform the worst as they require a substantially large number of validations to cover most (if not all) of the true violations.

3. **False Positive Detection:** Tuple based questions suffer from the worst false positive detection rate due to the inherent challenge in obtaining a small set of representative samples (*i.e.,* Armstrong relations) that have low false positive violations. Cell based questions also suffer from the false positive rate, albeit at a lower level. As the number of validations increase, there is a proportional decrease in the number of false positive violations. FD based questions do not suffer from false positives since each validated FD always detects true violations.

4. **Runtime Performance:** Tuple based questions are often the most efficient as the bottleneck involves verifying if a chosen tuple satisfies any saturation set. This is often independent of the number of tuples. The cell and FD based questions often require a running time proportional to the number of violations (both false positive and true positive). In a typical dataset, this number might be prohibitively large.

5. **Impact of Expert Responding as 'I Don't Know':** Given a fixed budget, the impact on tuple based questions is the highest. This is due to the fact that the rate of false positives is disproportionately high when the set of validated tuples are small. The impact on cell and FD based questions is relatively minimal as each FD is often violated by a large number of cells and the algorithm can highlight other cells to determine if an FD is valid.

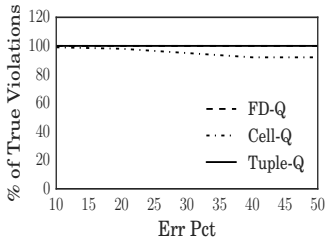


Figure 7: Impact of Pct of Erroneous Tuples on True Violations on Hospital Dataset

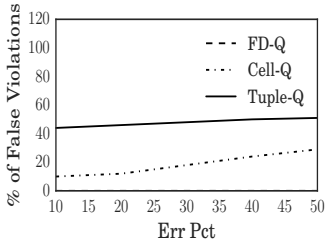


Figure 8: Impact of Pct of Erroneous Tuples on False Violations on Hospital Dataset

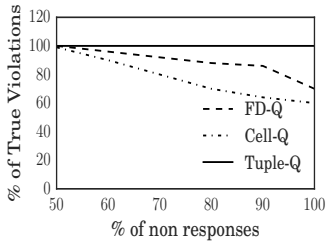


Figure 9: Impact of Expert IDK answers on Hospital Dataset

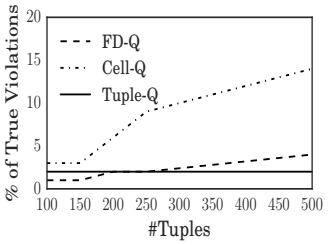


Figure 10: Runtime Performance on Tax Dataset

8. RELATED WORK

Discovering Integrity Constraints. Many ICs have been proposed to capture different types of data errors that violate these ICs. In practice, in many applications, these ICs are provided by domain experts through expensive eyeballing exercise. Different from them, our proposal finds all FD-detectable errors without having to determine explicitly all the true ICs first, which actually alleviates the experts' workload for finding relevant ICs for data error detection. There are various algorithms to automatically discover ICs. Many of them assume that there exists a clean yet representative sample of data, so the main issue is generally to improve the efficiency [18, 22, 27]. However, finding a set of clean yet representative ICs is generally hard. Moreover, there are algorithms to find approximate ICs when assuming that the data is not clean, such as discovering FDs [8, 19], CFDs [15], MDs [29], and DCs [11]. Unfortunately, as discussed earlier, a very large number of approximate ICs might exist when we assume that the dataset is dirty, and approximate IC discovery algorithms are very sensitive to dirty data. Consequently, asking the user to select the relevant ones from a large number of candidate ICs is often unfeasible in practice.

Different from these approaches, we *neither* assume that we have a clean sample data, *nor* ask the users to validate all approximate ICs. Our approach is to fill the gap between the large space of potentially useful approximate ICs and the limited validation capacity of the user. Also, we do not target at finding all relevant ICs. Instead, our focus is to find all FD-detectable errors, as some ICs are not helpful if there is no data violating them.

Evolving Data and Integrity Constraints. There are several proposals to deal with cases where, as data and business rules evolve, some of the ICs may no longer be valid while new ones need to be added. The authors of [9] propose a unified cost model for data and constraint repairs over a database that quantifies the trade-off of when an inconsistency requires a data repair compared to a constraint repair. The authors of [5] propose the notion of relative trust: data and FDs are not always equally trustworthy. Hence, they propose an algorithm for generating different suggestions to modify the data or the FDs in a minimal and non-redundant way. A user will then consult the suggested data or FD repairs and decide on the best way to resolve violations. In [24], the author focus on a narrower problem of modifying violated FDs by adding attributes to the antecedent of the dependency such that they are valid again.

In contrast, our approach faces a different situation where only data is given, and the user only wants to find all detectable errors, instead of valid ICs. From one hand, in our scenario, neither the data nor the ICs will evolve. From the other hand, the assumption that good ICs are given does not hold in our setting.

User Interaction for Data Cleaning. There are also several work that interacts with users for data cleaning. Guided data repair (GDR) [30] suggests possible repairs to the user who ultimately decides the right repair, which is in turn used to train a machine learning module. Falcon [17] interacts with the user to confirm SQL update queries from sample updates. The above approaches mainly aim to leverage the user's feedback in data repairing. We differ in that we take a step back to only detect data errors. A practical way to bridge our approach with other heuristic data repairing methods is to bootstrap the end-to-end data cleaning pipeline with our proposed approach to discover FDs, and then use them as input for other repairing algorithms to do the actual data repairing.

9. CONCLUSIONS AND FUTURE WORK

We have proposed a new framework for detecting data errors violating functional dependencies where the FDs are not explicitly provided by the experts. The goal was to solicit expert feedback up to a certain budget while maximizing the number of detected violations and minimizing the number of false positives. To achieve this goal, we have presented efficient algorithms to interact with the user on three types of questions, namely cell-based, tuple-based, and FD-based, and highlighted the relation to well studied problems in combinatorial optimization. We have demonstrated by extensive experiments that our solution can effectively and efficiently identify data errors by minimizing user involvement.

There are many promising directions for future work. One challenging direction is to extend our work to other ICs beyond FDs, such as those in the more general category of denial constraints. Another direction is to enhance the robustness of our algorithms where the expert may provide incorrect answers for a fixed fraction of questions. Going beyond error detection, solving the interactive budgeted data repair problem *without* the availability of FDs is an open problem and is also part of our future work.

10. REFERENCES

[1] Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, and N. Tang. Detecting data errors: Where are we and what needs to be done? *PVLDB*, 9(12):993–1004, 2016.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] P. C. Arocena, B. Glavic, G. Mecca, R. J. Miller, P. Papotti, and D. Santoro. Messing up with BART: error generation for evaluating data-cleaning algorithms. *PVLDB*, 9(2), 2015.

[4] L. Berti-Equille and J. Borge-Holthoefer. *Veracity of Data: From Truth Discovery Computation Algorithms to Models of Misinformation Dynamics*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2015.

[5] G. Beskales, I. F. Ilyas, L. Golab, and A. Galiullin. On the relative trust between inconsistent data and inaccurate constraints. In *ICDE*, pages 541–552. IEEE, 2013.

[6] J. Bisbal and J. Grimson. Database sampling with functional dependencies. *Information and Software Technology*, 43(10):607–615, 2001.

[7] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 746–755. IEEE, 2007.

[8] F. Chiang and R. J. Miller. Discovering data quality rules. *PVLDB*, 1(1), 2008.

[9] F. Chiang and R. J. Miller. A unified model for data and constraint repair. In *ICDE*, pages 446–457. IEEE, 2011.

[10] X. Chu, I. F. Ilyas, and P. Papotti. Discovering denial constraints. *PVLDB*, 6(13), 2013.

[11] X. Chu, I. F. Ilyas, P. Papotti, and Y. Ye. Ruleminer: Data quality rules discovery. In *ICDE*, 2014.

[12] M. Dallachiesa, A. Ebaid, A. Eldawy, A. K. Elmagarmid, I. F. Ilyas, M. Ouzzani, and N. Tang. NADEEF: a commodity data cleaning system. In *SIGMOD*, 2013.

[13] W. Fan, F. Geerts, and X. Jia. A revival of integrity constraints for data cleaning. *PVLDB*, 1(2):1522–1523, 2008.

[14] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *ACM Trans. Database Syst.*, 33(2), 2008.

[15] W. Fan, F. Geerts, J. Li, and M. Xiong. Discovering conditional functional dependencies. *IEEE Trans. Knowl. Data Eng.*, 23(5):683–698, 2011.

[16] M. R. Garey and D. S. Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.

[17] J. He, E. Veltri, D. Santoro, G. Li, G. Mecca, P. Papotti, and N. Tang. Interactive and deterministic data cleaning. In *SIGMOD*, pages 893–907, 2016.

[18] A. Heise, J. Quiané-Ruiz, Z. Abedjan, A. Jentzsch, and F. Naumann. Scalable discovery of unique column combinations. *PVLDB*, 7(4), 2013.

[19] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. TANE: an efficient algorithm for discovering functional and approximate dependencies. *Comput. J.*, 42(2):100–111, 1999.

[20] M. Interlandi and N. Tang. Proof positive and negative in data cleaning. In *ICDE*, 2015.

[21] S. Khuller, A. Moss, and J. S. Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999.

[22] S. Kruse, A. Jentzsch, T. Papenbrock, Z. Kaoudi, J. Quiané-Ruiz, and F. Naumann. Rdfind: Scalable conditional inclusion dependency discovery in RDF datasets. In *SIGMOD*, pages 953–967, 2016.

[23] J. Liu, J. Li, C. Liu, Y. Chen, et al. Discover dependencies from data - A review. *IEEE Transactions on Knowledge and Data Engineering*, 24(2):251–264, 2012.

[24] M. Mazuran, E. Quintarelli, L. Tanca, and S. Ugolini. Semi-automatic support for evolving functional dependencies. In *EDBT*, pages 293–304, 2016.

[25] T. Papenbrock, T. Bergmann, M. Finke, J. Zwiener, and F. Naumann. Data profiling with metanome. *Proceedings of the VLDB Endowment*, 8(12):1860–1863, 2015.

[26] T. Papenbrock, J. Ehrlich, J. Marten, T. Neubert, J.-P. Rudolph, M. Schönberg, J. Zwiener, and F. Naumann. Functional dependency discovery: An experimental evaluation of seven algorithms. *Proceedings of the VLDB Endowment*, 8(10):1082–1093, 2015.

[27] T. Papenbrock, S. Kruse, J. Quiané-Ruiz, and F. Naumann. Divide & conquer-based inclusion dependency discovery. *PVLDB*, 8(7):774–785, 2015.

[28] J. Pasternack and D. Roth. Knowing what to believe (when you already know something). In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 877–885. Association for Computational Linguistics, 2010.

[29] S. Song and L. Chen. Discovering matching dependencies. In *CIKM*, pages 1421–1424, 2009.

[30] M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas. Guided data repair. *PVLDB*, 2011.