Query-Guided Resolution in Uncertain Databases

Osnat Drien
Bar-Ilan University

Matanya Freiman
Bar-Ilan University

Antoine Amarilli
LTCI, Télécom Paris, Institut Polytechnique de Paris

Yael Amsterdamer
Bar-Ilan University

Abstract

We present a novel framework for uncertain data management. We start with a database whose tuple correctness is uncertain and an oracle that can resolve the uncertainty, i.e., decide if a tuple is correct or not. Such an oracle may correspond, e.g., to a data expert or to a crowdsourcing platform. We wish to use the oracle to clean the database with the goal of ensuring the correct answer for specific mission-critical queries. To avoid the prohibitive cost of cleaning the entire database and to minimize the expected number of calls to the oracle, we must carefully select tuples whose resolution would suffice to resolve the uncertainty in query results. In other words, we need a query-guided process for the resolution of uncertain data.

We develop an end-to-end solution to this problem, based on the derivation of query answers and on correctness probabilities for the uncertain data. At a high level, we first track Boolean provenance to identify which input tuples contribute to the derivation of each output tuple, and in what ways. We then design an active learning solution for iteratively choosing tuples to resolve, based on the provenance structure and on an evolving estimation of tuple correctness probabilities. We conduct an extensive experimental study to validate our framework in different use cases.

1 Introduction

Many scenarios involve data whose correctness is uncertain. Uncertainty may be due to data sources that are not fully reliable; due to the use of imperfect techniques such as those employed in Data Integration or Information Extraction; or due to stale data. An example for a large-scale uncertain database is that of NELL [69], consisting of 50M facts that were automatically extracted from different websites. Each fact is associated with a probability, based on the confidence level that NELL assigns to it.

While uncertain data can be highly valuable, some tasks may require certainty, e.g., mission-critical tasks. For such tasks, users may be willing to invest manual effort and time to resolve the uncertainty, for instance, by hiring data experts to verify the automatically extracted data against its sources. To avoid

exhaustive cleaning of the entire database, which can be prohibitively costly, previous work suggested approaches for semi-automated, interactive or crowd-powered cleaning processes (e.g., [8, 11, 19, 20, 70, 22, 32, 60, 64, 81, 93, 96]). Typically, such processes clean only a part of the data and/or only specific types of data errors, such as constraint violations.

In the present work, we study the problem of *query-guided* uncertainty resolution. We start from a database whose data correctness is uncertain (i.e., which may contain incorrect tuples) and a query (or a set of queries) that capture the data relevant for analysis. Our goal is to identify the *precise set of correct query results*, by resolving the uncertainty of input tuples. Of course, we can do this by naively verifying every input tuple, thereby achieving a certain and correct input database. However, as mentioned above, verifying all input tuples may be too costly. We thus aim at verifying a subset of the input tuples that suffices to determine the correct results of the given query. As we will show, there exist such subsets that are typically significantly smaller than the full database. Resolving the uncertainty of a tuple is abstractly modeled as a probe to an oracle: in practice, the oracle may be data experts, crowd workers, high-quality external sources, etc. Therefore, our challenge is identifying which tuples to probe in order to minimize the number of oracle calls. Our novel solution addresses this challenge by accounting, in a fine-grained manner, for how the data is derived and for the probabilities of probe answers.

Example 1.1. *Consider a data analyst whose task is to identify promising entrepreneurs at the early stages of their careers. The analyst uses data on company acquisitions and founders from NELL, and hires the services of a data expert in order to ensure that business recommendations are made only based on correct data. Clearly, the effort of such an expert should be minimized.*

Now, assume that the analyst describes a part of NELL that is of interest via the simple query "select all the names of company founders". In this case, the data expert only needs to verify tuples describing company founders, and not even all such tuples: in order to verify if a person is a correct query result, it suffices to find one *correct input tuple describing a company they founded (or, alternatively, establish that all such tuples are incorrect). An estimation of correctness probabilities can greatly help here, in choosing between input tuples that associate different companies with the same founder, since e.g. the tuple from the most reliable source may also be the likeliest to prove this person is indeed a company founder. This analysis naturally gets more complex for queries that are more complex. Moreover, the choice of a tuple to verify should account for its effect on determining the correctness of other output tuples (e.g., the output of another query that selects company names), and for its effect on estimating the correctness of similar input tuples.*

Solution Overview. Towards addressing this problem, we present an end-to-end framework for query-guided resolution in uncertain databases. Figure 1 shows the main components and flow of our framework: first, the user issues a query over the data (Step 1, on the left). The query is evaluated along with fine-grained Boolean provenance tracking (Step 2). Similarly to previous work on probabilistic databases (e.g., [24, 57, 75, 87, 27]), each output tuple is annotated by a Boolean provenance expression, where each variable in the Boolean

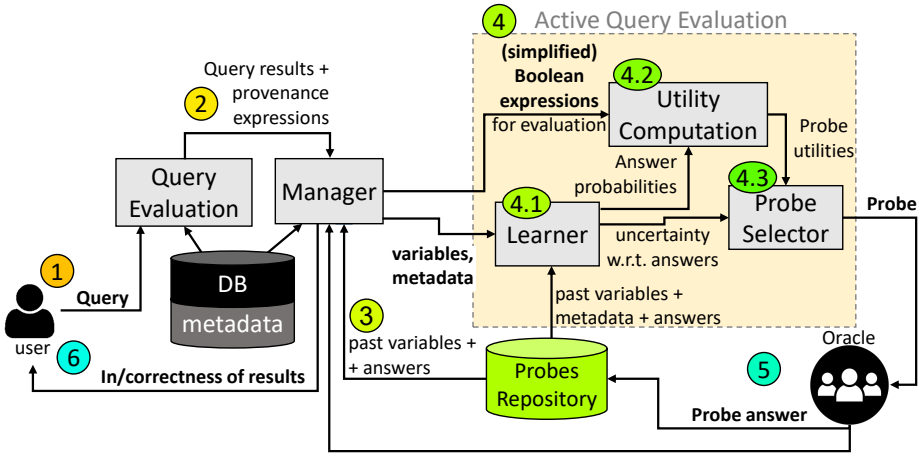


Figure 1: Framework architecture

expressions stands for the correctness of an input tuple, and each expression evaluates to True iff its output tuple is correct. Unlike the semantics of [87] for query evaluation on probabilistic databases, we now choose input tuples to verify with the help of the oracle, i.e., variables whose truth value will be revealed. We first plug into the Boolean expressions the known truth values from previous probes that are stored within the Probes Repository, if exist. Such known probe answers are assigned to the relevant variables, and the expressions are simplified accordingly (Step 3). If previously resolved data suffices to determine the correctness of query results, we can stop at this point. Otherwise, we invoke Active Query Evaluation (Step 4) to select the next variable to verify. The Learner module (Sub-step 4.1) uses a classifier trained on the Probes Repository, in order to predict correctness based on metadata such as the data source and type. Probe answer probabilities are then passed on to the Utility Computation module (Sub-step 4.2) which computes a score reflecting the expected contribution of each candidate probe towards evaluating the Boolean expressions. The Probe Selector (Sub-step 4.3) weighs the utility of each candidate probe and its expected contribution to lowering the uncertainty of the learned model. The chosen probe is sent to the oracle and its answer is recorded in the Probes Repository. Steps 3-5 repeat iteratively, where depending on the choice of learning algorithm, we may update the Learner by training it on the new answers, or by retraining it from scratch on the entire data. At each point of this iterative process, the user can view the current subset of query results determined to be (in)correct.

More concretely, in this work we focus on Select-Project-Join-Union (SPJU) queries, for which provenance can be computed in the form of monotone Boolean expressions [47, 52], i.e., without negation (see Section 2.3). As a Learner suitable for a small training set (Step 4.1), we use a Random Forest (RF) classifier, and estimate the probe answer probability as the fraction of trees classifying a candidate probe as correct. We feed this classifier to *Learning Active Learning* (LAL) [59], to quantify the reduction in the uncertainty of our estimation that each candidate probe is expected to yield (See Section 4).

Acquisitions

Acquired	Acquiring	Date	
A2Bdone	Zazzer	7/11/2020	a_0
microBarg	Fiffer	1/5/2017	a_1
fPharm	Fiffer	1/2/2016	a_2
Optobest	microBarg	8/8/2015	a_3

Roles

Organization	Role	Member	
A2Bdone	Founder	Usha Koirala	r_0
A2Bdone	Founding member	Pavel Lebedev	r_1
A2Bdone	Founding member	Nana Alvi	r_2
microBarg	Co-founder	Nana Alvi	r_3
microBarg	Co-founder	Gao Yawen	r_4
microBarg	CTO	Amaal Kader	r_5

Education

Alumni	Institute	Year	
Usha Koirala	U. Melbourne	2017	e_0
Pavel Lebedev	U. Melbourne	2017	e_1
Nana Alvi	U. Sau Paolo	2010	e_2
Nana Alvi	U. Melbourne	2017	e_3
Gao Yawen	U. Sau Paolo	2010	e_4
Amaal Kader	U. Cape Town	2005	e_5

Table 1: Example probabilistic database.

For utility Computation (step 4.2), we develop functions that compute the expected contribution of a probe based on an analysis of algorithms from the field of *Interactive Boolean Evaluation* [4, 15, 28, 43, 53, 90], and the related work on Consent Management [31]. Briefly, Interactive Boolean Evaluation algorithms get, as input, a Boolean expression and (independent) variable probabilities, and sequentially select variables for which the truth value is revealed. The goal is to minimize the overall number of observed variables, corresponding to our oracle probes. The algorithms of [31], get as input a database with (independent) probabilities on tuples and a query, and select tuples for which to ask consent, corresponding to our oracle probes. In contrast, in our setting, we are given an uncertain database with metadata and a query, while *answer probabilities may be unknown to the algorithm and correlated*. Due to this difference, solutions from this previous work are inadequate for our setting. Instead, we build on three such algorithms but recast each of them in a utility function that assigns numeric scores to all probes (see Sections 4-6). This enables us, in contrast with the aforementioned lines of previous work, to develop an end-to-end solution, accounting for the (gradual) learning of answer probabilities and for actively selecting probes based on considerations of both utility and expected uncertainty reduction.

Contributions. The main contributions of this work are:

1. *Problem formulation.* We define the problem of query-guided uncertainty resolution: start with an uncertain database, metadata and a query, and find the *precise* set of ground-truth query answers using a *minimal* number of oracle calls for verifying tuples.

2. *Framework architecture.* Our end-to-end framework is designed to solve the problem through an iterative process of learning to estimate oracle answer probabilities, and using this estimation to select the most effective oracle calls. As explained above, the framework features a unique combination of techniques

from fields such as provenance, Active Learning and Interactive Boolean Evaluation.

3. *Analysis.* We conduct a foundational analysis of our optimization problem and show that already for restricted query classes it is intractable.

4. *Algorithms and Implementation.* In spite of the intractability of achieving an optimal solution, we develop an effective end-to-end solution to our problem, which computes the precise set of correct query answers, and, as we show empirically, does so by probing only a relatively small subset of the input database. In particular, our solution incorporates active learning of probe answer probabilities, which we show to be helpful in identifying effective probes and reducing the overall number of probes. We describe concrete implementation choices and use them in a prototype system.

5. *Experimental study.* We conduct an experimental study over real, large-scale databases. We test the effectiveness of our framework as a whole and isolate the effect of its modules, for both real and synthesized ground truth. We then provide practical guidelines of which solution component to use in different cases.

Novelty. There is a large body of work dealing with uncertain data. Some approaches propagate this uncertainty to query answers, for instance, compute the probability of query answers (e.g., [3, 10, 24, 57, 75, 87, 79, 27]), compute the subset of certain or consistent query answers, i.e., with probability 1 (e.g., [7, 45, 63, 95]), or condition the probabilities over additional information such as data constraints, e.g., FDs (e.g., [91, 56, 72]). A different line of work, including, e.g., [19, 70, 64] focuses on an optimization problem that is in a sense *dual* to ours: maximally reducing the uncertainty using a given cleaning budget. Similarly to these lines of previous work, our solutions use (estimated) tuple correctness probabilities; unlike these works, we identify the *full ground truth query answers to given queries*, to support cases when such answers are required, e.g., when the data is mission-critical. Our optimization focuses on minimizing the number of tuples verified, corresponding to oracle calls, and hence the techniques that we employ are of a different flavor.

The rest of this paper is organized as follows. The formal model and problem statement are given in Section 2. Then, in Section 3, we show intractability results. We then describe the modules of our framework: the Learner in Section 4, utility computation in Section 5, and probe selection in Section 6. Our implementation and experimental results appear in Section 7. We overview related work in Section 8 and conclude in Section 9.

2 Preliminaries

We next provide formal definitions along with examples of our setting, starting with a general model of uncertain databases that serves, in our framework, as a vehicle for deciding which parts of the data to resolve. We then present the unique aspects of active uncertainty resolution.

2.1 Uncertain Databases and Possible Worlds

An uncertain database associates each tuple with a unique variable, which intuitively should be assigned True if the tuple is correct and False otherwise. A truth assignment to these variables yields a *possible world* of the database, which consists of exactly the tuples whose variables were assigned to True. This follows a standard model of an uncertain database (e.g., [52]). We will further introduce below a probability distribution over the variables, but will not assume that it is given to us as input; instead, we will learn the distribution as part of the process.

Definition 2.1 (Uncertain databases)**.** *An uncertain database is represented as* $\bar{D} = (D, X, L)$ *where* D *is a relational database,* X *is a set of Boolean random variables and* $L : \text{tuples}(D) \to X$ *is an* injective *(one-to-one) mapping of each tuple to a variable standing for the event that this tuple is correct.*

Example 2.1. *Table 1 outlines an uncertain database with three relations:* Acquisitions*, including data on companies acquired by other companies;* Roles*, including data on roles of different organization members; and* Education*, including data on university alumni. The rightmost column in each shows the variable in* X *annotating the tuple.*

SPJU queries are the queries obtainable using the Positive Relational Algebra operators of Selection, Projection, Inner Join and Union [1]. In particular, this allows us to use negation inside the Selection or Inner Join predicates (e.g., `Year!=2017`), but not query operators for negation or difference (e.g., `Year NOT IN ...`). SPJU queries are *monotone*, i.e., for relational databases D, D' with the same schema, if $D \subseteq D'$ then $Q(D) \subseteq Q(D')$. For convenience, we will use SQL syntax in our examples.

By assigning truth values to $\bar{D}$'s variables, we obtain a *possible world* where only a subset of the tuples is correct. Given an (SPJU) query, we can accordingly define the correctness of output tuples.

Definition 2.2 (Truth valuations)**.** *A truth valuation* $\text{val} : X \to \{\mathit{True}, \mathit{False}\}$ *yields a* possible world $D_{\text{val}} = \{t \in D \mid \text{val}(L(t)) = \mathit{True}\} \subseteq D$*, namely, the subset of database tuples whose variables have been mapped to True. Given also a query* Q *over* $\bar{D}$*, a tuple in its output* $t \in Q(D)$ *is said to be correct w.r.t.* val *iff* t *is in the output of* $Q(D_{\text{val}})$*. A* probability distribution π *over* X *assigns, to every truth valuation, or, equivalently, to every possible world, a probability value in* $[0, 1]$ *(such that the sum of probabilities over all valuations is 1).*

Intuitively, the probability distribution π captures the likelihood of input tuples to be correct. We will not assume that π is given as input, but rather will learn it based on metadata.

Example 2.2. *Recall Example 1.1 regarding an analyst seeking promising entrepreneurs. This analyst may e.g. issue the SPJU query in Figure 2 over the DB in Table 1, which returns companies acquired since 2017 along with institutes in which founders of these companies had studied. The query results are shown in Table 2 (ignore, for now, the annotations on the left). In the event that the first two tuples in the Acquisitions relation are incorrect (*$\text{val}(a_0) = \text{val}(a_1) = \mathit{False}$*) the query has no correct results since the join of the Acquisitions table with the other tables would necessarily be empty. If, alternatively, the first tuple of each*

```
SELECT DISTINCT a.Acquired, e.Institute
FROM Acquisitions AS a, Roles AS r, Education AS e
WHERE a.Acquired = r.Organization AND
      r.Member = e.Alumni AND a.Date >= 2017.01.01 AND
      r.Role LIKE '%found%' AND e.YEAR <= year(a.Date)
```

Figure 2: Query over the example database

of the three relations is correct (i.e., $\mathrm{val}(a_0) = \mathrm{val}(r_0) = \mathrm{val}(e_0) = \mathit{True}$*), the first result tuple of the query,* `(A2Bdone, U. Melbourne)`*, derived from these input tuples, is correct.*

2.2 The Active Model

We next model the process of revealing the ground truth of the random variables, i.e., the correctness of the corresponding input tuples towards determining the full ground truth query results, by probing an oracle. As mentioned above, our oracles may be domain experts or crowd workers, and they may verify correctness e.g. by looking at the Web page from which the data has been extracted. I.e., we regard, in this work, the oracle answers as the ground truth. We discuss what happens when this is not the case in Section 9.

Let $\mathrm{val}^* : X \to \{\mathrm{True}, \mathrm{False}\}$ be a ground truth valuation reflecting the correctness of input tuples; val^* may be (fully or partially) unknown and can be discovered by oracle probes, where each probe is a choice of a random variable $x \in X$ for which $\mathrm{val}^*(x)$ is revealed.

Our problem, called `OPT-RESOLVE` (standing for optimally resolving uncertainty) is then as follows. We are given as input an uncertain database $\bar{D} = (D, X, L)$, an SPJU query Q over D and an oracle that we can probe to reveal the ground truth valuation $\mathrm{val}^*(x)$ for any $x \in X$. There is an underlying probability distribution π over X, but it is not given to us as input. Our goal is to compute the precise ground truth for the query result, i.e., $Q(D_{\mathrm{val}^*})$, while performing a minimal number of sequential oracles probes. Since the total number of probes will typically depend on the obtained probe answers, we will aim at minimizing the *expected* number of probes with respect to π.

`OPT-RESOLVE` may be viewed as an exploration-exploitation problem: oracle probes are not only useful for directly revealing the ground truth but may also be used for estimating (parts of) the underlying π, and thereby guiding the selection of subsequent probes.

Since our goal in this work is to optimize the number of oracle probes, sequential probing is the preferable strategy, enabling probe selection to be optimized based on all previous probe answers. In Section 6, we explain how probes can be selected and issued in parallel, to improve latency as a secondary consideration.

2.3 Provenance

A key component in our solution is that of Boolean provenance [3, 6, 5, 9, 10, 17, 18, 30, 29, 35, 41, 40, 47, 46, 50, 52, 54, 68, 77, 82, 84, 99], computed alongside query evaluation (step 2 in Figure 1). Briefly, given an uncertain database $\bar{D} = (D, X, L)$ and a query Q over D, we compute the representation $Q(\bar{D}) = (Q(D), \mathrm{Bool}[X], L')$ that consists of the "standard" query result and

Acquired	Institute	
A2Bdone	U. Melbourne	$(a_0 \wedge r_0 \wedge e_0) \vee (a_0 \wedge r_1 \wedge e_1) \vee (a_0 \wedge r_2 \wedge e_3)$
A2Bdone	U. Sau Paolo	$(a_0 \wedge r_2 \wedge e_2)$
microBarg	U. Melbourne	$(a_1 \wedge r_3 \wedge e_3)$
microBarg	U. Sau Paolo	$(a_1 \wedge r_3 \wedge e_2) \vee (a_1 \wedge r_4 \wedge e_4)$

Table 2: Result of the example query.

a labeling function L' that maps each output tuple t to a Boolean expression over the input annotations. Crucially, L' has the property that a valuation val satisfies $L'(t)$ if and only if t is correct w.r.t. val [47, 52]. In other words, executing Q over the possible world corresponding to val is equivalent to assigning truth values to the provenance expressions annotating $Q(\bar{D})$ and retaining only the tuples whose annotation evaluated to True. For SPJU queries, provenance can be computed in polynomial time [52] as monotone k-DNF expressions, i.e., DNF without negation and where terms (conjunctions) include at most k variables.

Given a query Q over an annotated database $\bar{D}$, we will denote the set of Boolean expressions by $\Phi(Q, \bar{D}) = \{L'(t) \mid t \in Q(D)\} \subseteq \text{Bool}[X]$, and for brevity use Φ when Q and $\bar{D}$ need not be specified.

Example 2.3. *The rightmost column of the query result in Table 2 shows the provenance annotation for each output tuple (in this case, it is in 3-DNF). In accordance with Example 2.2, since every provenance term contains either a_0 or a_1, if $\text{val}(a_0) = \text{val}(a_1) =$ False then all four expressions are evaluated to False, i.e., every output tuple is incorrect. Similarly, if $\text{val}(a_0) = \text{val}(r_0) = \text{val}(e_0) =$ True, the first conjunction in the first Boolean expression and hence the entire expression evaluates to True, hence the first output tuple is correct.*

3 Intractability Results

We now study the complexity of `OPT-RESOLVE`, namely, of identifying the precise set of correct query results using a minimal expected number of oracle calls. First, we note that, based on previous work, we can directly derive an intractability result for the problem in its full generality, as follows. Recall that Interactive Boolean Evaluation is the optimization problem of probing the truth values of variables in order to decide the truth value of a given Boolean formula. The work of [4] shows that this optimization problem is hard for k-DNFs, which is the general shape of our provenance expressions (see Section 2.3). We can reduce this problem to `OPT-RESOLVE` in the same manner that was done for consent management in [31] (Theorem III.5).

A natural question to ask is then, can we find restricted cases for which finding an optimal solution is tractable (in data complexity)? To this end, we next show that `OPT-RESOLVE` is intractable even for restricted cases: even for Selection-Join (SJ) queries, i.e., queries that include no projection and no union and for Selection-Projection-Union queries (SPU), that include no join; and even in the case when input tuple probabilities are *independent and known* in advance to the algorithm, i.e., the event that each tuple is correct is captured by a binary random variable, whose probability to be True is independent of

the other variables and given as additional input to the algorithm. For brevity, we refer to such variables in the sequel as "known" and "independent".

Theorem 3.1. *If $P \neq NP$, there exists a Selection-Join (SJ) query Q for which there is no algorithm to solve the problem* `OPT-RESOLVE` *that is polynomial in the database size. This holds even in the setting where the probabilities of database tuples are known and independent.*

Proof. We prove the intractability of `OPT-RESOLVE` for this class of queries via a reduction from the NP-hard problem `VERTEX COVER`. First, we define a fixed SJ query Q, which is used in the reduction: `SELECT * FROM Vars v1, Vars v2, Terms t WHERE v1.a = t.a1 AND v1.a = t.a2`. Given an input to `VERTEX COVER`, namely, a graph $G = (V, E)$, our goal is to find a minimum cardinality vertex cover, i.e., a subset of V such that every edge in E contains some vertex of the subset. Define an uncertain database $\bar{D} = (D, X, L)$ such that D includes the 1-ary relation `Vars` and the 2-ary relation `Terms`. For each $v \in V$, we define a tuple $t_v = (v) \in$`Vars`, such that $L(t_v) = x_v$; and for each edge $(u, v) \in E$, we define a tuple $t_{u,v} = (u, v) \in$`Terms`, such that $L(t_{u,v}) = x_{u,v}$. By this construction, the query result $Q(\bar{D})$ contains, per $(u, v) \in E$, exactly one tuple (u, v, u, v), constructed of two `Vars` tuples corresponding to vertices and one `Terms` tuple corresponding to the edge. The provenance of such an output tuple is therefore $x_u \wedge x_v \wedge x_{u,v}$.

Next, we can show that for π that assigns to variables sufficiently low probabilities, given to the algorithm as additional input, and a constant False ground truth, any optimal solution to `OPT-RESOLVE` would sequentially probe a minimal number of variables sufficient to prove that all provenance expressions are False – termed a *minimal 0-certificate*.

Let us use as the input to `OPT-RESOLVE` the database $\bar{D}$, query Q, empty metadata and the above mentioned π, and assume that the resulting optimal strategy issues probes $x_1, \ldots x_m$ for the constant False ground truth valuation. We now show that we can construct from $x_1, \ldots, x_m$ a minimum vertex cover for G of size m: for each probed variable x_v, add to the cover the vertex v; for every probed variable $x_{u,v}$, add to the cover the vertex u. This is indeed a cover of G: for every edge (u, v), at least one variable of $x_u \wedge x_v \wedge x_{u,v}$ is probed, so either u or v are in the cover. In the other direction, given a vertex cover of size m', we can construct a 0-certificate of size m' by probing x_v for each v in the cover. Hence, by the minimality of the 0-certificate, the vertex cover we defined is also minimal. This concludes our reduction. Note that some parts of this proof are adapted from the proof technique of Theorem 1 in [4], including the connection between 0-certificates and optimal evaluation algorithms (Lemma 1 of [4]); however, their proof was for a single 2-DNF, and so our proof includes novel constructions including the query Q, the uncertain database $\bar{D}$, and the particular reduction for the resulting provenance expressions. □

Similarly, we prove the intractability of `OPT-RESOLVE` for SPU queries, whose provenance is in the form of disjunctions (since the queries are join-free).

Theorem 3.2. *If $P \neq NP$, there exists a Selection-Projection-Union (SPU) query Q for which there is no algorithm to solve the problem* `OPT-RESOLVE` *that is polynomial in the database size. This holds even in the setting where the probabilities of database tuples are known and independent.*

Proof. We prove the intractability of `OPT-RESOLVE` for this class of queries via a reduction from the NP-hard problem `3-VERTEX COVER`, i.e., for graphs where all vertices have degree ≤ 3. In this proof, we define a fixed SPU query Q: `SELECT e1 FROM Graph UNION SELECT e2 FROM Graph UNION SELECT e3 FROM Graph`. Given an input $G = (V, E)$ to `3-VERTEX COVER`, we define an uncertain database $\bar{D} = (D, X, L)$ such that D includes the 3-ary relation `Graph`. For each $v \in V$ contained in edges $e_{v_1}, e_{v_2}, e_{v_3}$ we define a tuple $t_v = (e_{v_1}, e_{v_2}, e_{v_3})$ in `Graph`, such that $L(t_v) = x_v$. (We can handle vertices with < 3 edges e.g. by setting some of the tuple values to `NULL` and adding a selection criterion to Q to avoid `NULL` results.) By the construction of Q, the query result $Q(\bar{D})$ contains, per $e = (u, v) \in E$, exactly one tuple (e), constructed of the union of the two occurrences of e in the input tuples t_u, t_v. The provenance of such an output tuple is therefore $x_u \vee x_v$.

Next, we can use a counterpart of the connection between minimal 0-certificates and optimal algorithms, noted in the proof of Theorem 3.1, and show that for sufficiently *high* probabilities, through an optimal solution to `OPT-RESOLVE`, we can obtain a minimal 1-certificate $x_{v_1}, \ldots x_{v_m}$ for the provenance of $Q(\bar{D})$, i.e., a minimal proof that all expressions are True. We can show, similarly to the proof of Theorem 3.1, that in this case, $x_{v_1}, \ldots x_{v_m}$ is a minimal vertex cover for G, which concludes our reduction. □

The above intractability results stand in contrast with previous optimality results by [15, 31]: both SJ and SPU queries yield provenance in the form of conjunctions and disjunctions, respectively. In particular, since variable repetitions can be eliminated using absorption ($x \vee x = x$), each provenance expression is in *read-once* DNF, namely, does not include multiple occurrences of variables. A PTIME optimal solution to Interactive Boolean Evaluation given a single read-once DNF expression was shown in [15]. This result was extended in [31] to an optimal PTIME solution for multiple DNF expressions that do not include multiple occurrences of variables across expressions. The intractability in our case thus stems from variable repetitions across multiple provenance expressions.

4 Learning Probabilities

Our theoretical results indicate that achieving the optimal, minimum number of oracle calls, even for restricted query classes, is generally infeasible. In this section and the following, we nevertheless propose practical solutions to our problem, further accounting for probability estimation by active learning from metadata, and showing how to incorporate the gradual learning of probabilities in the Boolean evaluation process.

We start by "zooming-in" on the sub-components of the Learner Module (Sub-step 4.1), as depicted in Figure 3. These sub-components provide, for each candidate probe, estimates of (1) the probability this probe will be answered affirmatively and of (2) the expected uncertainty reduction following the probe answer.

Estimating probabilities based on metadata. *Metadata attributes*, together with the ground truth correctness of tuples that were already probed, can be used to estimate the correctness probability of the remaining tuples.

Definition 4.1 (Metadata)**.** *A metadata attribute is a pair* $\mathtt{a} = v$ *of an attribute name* $\mathtt{a} \in \mathcal{A}$ *and value* $v \in \mathcal{D}_{\mathtt{a}}$*, respectively, for some attribute name and value domains* $\mathcal{A}, \mathcal{D}$*. The function meta* : $D \to \mathrm{P}[\mathcal{A}, \mathcal{D}]$ *maps each tuple* $t \in D$ *to a set of metadata attributes.*

The *Known Probes Repository* (see Figure 3) is a (possibly empty) set of tuples in D, along with their metadata and correctness truth values obtained from past probes (e.g., of other queries).

Example 4.1. *In the database of Figure 1, the metadata of the first tuple, annotated by* a_0*, may include attributes reflecting its source, e.g.,* `source` = `example.com`*, attributes related to its schema, e.g.,* `rel_name` = `Acquisitions` *and attributes related to its content or even containing (parts of) its content, e.g.,* `has_value` = `A2Bdone`*,* `has_value` = `Zazzer`*, etc.*

We describe, in Section 7, practical derivation of such metadata from general-purpose datasets, and discuss the effect of available metadata. In what follows, we model the probability distribution for tuple correctness as a function of the metadata attributes, and leverage past probe answers to estimate (relevant parts of) this function. We discuss alternative probabilistic models in Sections 8 and 9.

Learning a classifier. The first step of the Learner is training a classifier to predict probe answers based on the metadata of the input tuples. We use two learning modes: *offline*, where the classifier is only trained once in the first Learner invocation, based on the initial known Probes Repository; and *online*, where the classifier is updated with each incoming probe answer.

The choice of the learning algorithm may depend on the setting (e.g., data domain and type of available metadata). However, since we are interested in minimizing oracle calls, active learning solutions that require relatively large training data such as Deep Neural Networks (e.g., [73, 85]), Generative Adversarial Networks (e.g., [65]), and Support Vector Machines (e.g., [13, 48]) are less adequate for our purpose.

In our implementation, we use Random Forests (RFs) for the classifier (e.g. [25]). This model admits highly efficient training algorithms and can be trained on a small initial input, which makes RF a lightweight solution particularly suitable for an active learning setting [16, 59]. We have also conducted experiments with a Naïve Bayes classifier [94], which performed similarly or slightly worse than RF and therefore these results are omitted.

Estimating probabilities. In our framework, we do not predict the answers of Boolean probes, but rather estimate the *probability* that each answer is True. We thus use a Probability Estimator (see Figure 3), which gets as input the classifier and candidate probes (variables whose truth value is unknown, along with their metadata). The Probability Estimator computes, for each variable x, the probability that x is classified as "correct". In particular, Random Forests admit a common generalization as probability estimators by considering each tree as a "vote" for the class it assigns to x and using the percentage of votes as the probability of each class.

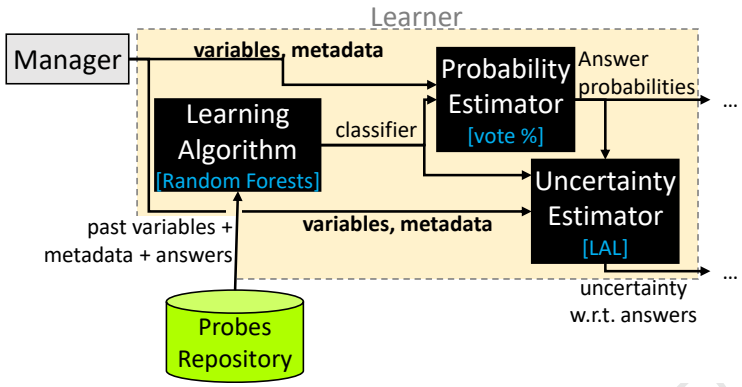


Figure 3: "Zoom-in" on the Learner Module (Sub-step 4.1 of Figure 1) with implementation choices in square brackets.

Estimating uncertainty reduction. In the online learning mode, we consider the effect of probes in extending the training set of known probe answers, thereby improving the classifier and thereby also improving the estimation of probe answer probabilities (the improved model will in turn lead to better probe selection in subsequent steps, and so forth). Concretely, we quantify the expected contribution of a candidate probe to decreasing the *generalization error* of the model, namely, prediction errors stemming from insufficient training. In our implementation, we use a recent solution called *Learning Active Learning (LAL)* [59]. Briefly, LAL uses a regressor that is trained on an annotated dataset (which does not need to come from the domain of interest). The regressor is transferred to predict the error reduction for an instance in a specific learning state (in our case, a candidate probe and a trained classifier), treating it as a regression problem.

5 Utility Computation

Using the estimated probe answer probabilities, the Utility Computation module (step 4.2 in Figure 1) estimates the *utility* of each candidate probe towards evaluating the Boolean provenance expressions, i.e., verifying the correctness of query output tuples. Recall that, as explained above, Interactive Boolean Evaluation algorithms get as input Boolean expressions with an independent probability that each variable is True, and sequentially reveal the truth values of variables in order to reveal the truth values of formula. The following example illustrates why we need to assign utility scores to variables rather than directly use such algorithms.

Example 5.1. *Consider the following scenario: in the query output in Table 2, some Interactive Boolean Evaluation algorithm may choose a_0 as the next variable to probe, based on some properties of a_0. However, it could be the case that a_1 is similar to a_0 in terms of the properties that the algorithm uses (e.g., a False answer to either a_0 or a_1 would lead to the evaluation of two Boolean expressions as False). In this case, if a_1 leads to a greater improvement of the learned model, it may be preferable to select a_1. The utility scores allow us to*

quantify and compare the expected contribution of each variable, and to combine this quantity with other considerations.

An optimal utility score for `OPT-RESOLVE` would assign, at each point, the highest score to the variable that minimizes the expected number of subsequent probes; however, the intractability results of Section 3 imply that computing such a utility value is not possible in PTIME. Instead, we recast three PTIME algorithms for Interactive Boolean Evaluation in utility functions that capture the principle by which probes are selected in each original algorithm. In particular, for any Boolean expression and probabilities, the probe that the algorithm would have chosen is given the highest utility score. In case of a tie, when the algorithm chooses arbitrarily among a set of probes, the corresponding utility would assign these probes equal scores. We also explain below, for each utility function, how it accounts for the evaluation of *multiple* provenance expressions.

Definitions and notations. Formally, we are given a query answer $Q(\bar{D}) = (Q(D), \Phi, L')$, computed by the Query Evaluation Module (Figure 1). We are also given a partial truth valuation $\text{val} : X_{\text{known}} \to \{\text{True}, \text{False}\}$ for variables whose truth value has been revealed and recorded in the Known Probes Repository. For the other, unobserved variables, we are given an estimation $\tilde{\pi} : X - X_{\text{known}} \to [0, 1]$ of the variable probabilities computed by the Learner Module, and our goal is to compute $\text{util}(\Phi, \tilde{\pi}, \text{val}, x)$, i.e., the utility of x for Φ with respect to our current knowledge on the probabilities and truth values.

Since we will gradually construct partial valuations, we denote by $\text{val}_{\text{x=True}}$ the valuation identical to val except that it also assigns True to a variable $x \in X - X_{\text{known}}$, and define $\text{val}_{\text{x=False}}$ in the same way. We abuse notation and denote by $\text{val}(\varphi)$ the Boolean expression obtained from φ by replacing each x in the domain of with $val(x)$. For a set Φ, $\text{val}(\Phi) = \{\text{val}(\varphi) \mid \varphi \in \Phi\}$. Given a formula $\varphi \in \Phi$, denote by $\text{nt}(\varphi)$ and $\text{nc}(\varphi)$, respectively, the number of terms (conjunctions) in the DNF of φ and the number of clauses (disjunctions) in its CNF.

Q-Value. The first utility function that we define, called *the Q-Value utility function* (by its connection to [28], as explained below), measures the utility of a variable according to the number of DNF terms (i.e., conjunctions) and CNF clauses (i.e., disjunctions) expected to be evaluated when revealing this variable's truth value. The utility function is defined as:

$$\begin{aligned}\text{util}(\Phi, \tilde{\pi}, \text{val}, x) = \\ \sum_{\varphi \in \Phi} \{ &\text{nt}(\varphi) \cdot \text{nc}(\varphi) - \tilde{\pi}(x) \cdot \text{nt}(\text{val}_{\text{x=True}}(\varphi)) \cdot \text{nc}(\text{val}_{\text{x=True}}(\varphi)) \\ &- (1 - \tilde{\pi}(x)) \cdot \text{nt}(\text{val}_{\text{x=False}}(\varphi)) \cdot \text{nc}(\text{val}_{\text{x=False}}(\varphi)) \} \quad (1)\end{aligned}$$

Consider a Boolean expression $\varphi \in \Phi$, whose truth value is unknown at the current point of evaluation, and a variable x whose truth value is unknown. The Q-Value utility is maximized if, by probing x, we are guaranteed to discover the truth value of φ, i.e., either the number of terms or the number of clauses is 0 regardless of the probe answer, which means that the two subtracted products are 0. It is also maximized when we are certain that x is True/False

(i.e., $\tilde{\pi}(x) \in \{1, 0\}$) and in this case φ is evaluated. Specifically, multiplication is used so that it suffices that one of the multiplicands is 0. When Φ includes multiple expressions, we observe that the sum of individual expression utilities reaches its maximum iff each individual utility reaches its maximum, i.e., all the expressions are evaluated, which is our goal. Hence, our utility function sums over the utilities of individual expressions. Our design of the utility function is based on several results from [28], including the use of multiplication to capture alternatives (we either want to evaluate all terms or all clauses, Lemma 1 of [28]) and the use of sum to capture joint evaluation of multiple expressions (Theorem 7 of [28]).

RO. Next, we introduce a utility function whose value is maximized for variable(s) least likely to be True, in the DNF term(s) most likely to be True. Intuitively, such variables are useful for proving that a Boolean expression is True, since this requires finding one DNF term that evaluates to True, and we choose the term most likely to be True; and it is also useful for proving that a Boolean expression is False, since for that we need to find a False variable in each DNF term, and the variable we choose is most likely to be False in its term. We extend this intuition to account for multiple expressions, by assigning higher utility scores to variables in the terms most likely to be True *overall*, across all expressions.

To this end, we first define the weight of a term T in the DNF of any $\varphi \in \Phi$ to be $\mathrm{W}(T) = \frac{1}{|T|} \prod_{x \in T} \tilde{\pi}(x)$. Intuitively, if variables are independent, this reflects the probability of a term to be True, and we divide it by the number of probes required to evaluate the term in the worst case. We then write the RO utility function as

$$\begin{aligned}\mathrm{util}(\Phi, \tilde{\pi}, \mathrm{val}, x) = (1 - \tilde{\pi}(x)) \\ + \max_{\varphi \in \mathrm{val}(\Phi)} \max_{T \in \{\mathrm{terms}(\varphi) | x \in \mathrm{vars}(T)\}} \alpha \cdot (\mathrm{W}(T) + \varepsilon) \quad (2)\end{aligned}$$

In Formula (2), to ensure that the variables of higher-weight terms get higher utility scores, we multiply the term weight by a factor α large enough to ensure that $\alpha \cdot (\mathrm{W}(T) + \varepsilon) > 1$ (second summand), and then take the highest score over all the terms across all the Boolean expressions in which the variable occurs. For example, we can choose α to be $\frac{1+\varepsilon}{\varepsilon + \min_{T \in \mathrm{terms}(\varphi)} W(T)}$. Next, the first summand is higher for variables more likely to be False. Since this summand is ≤ 1, utility is strictly greater for variables occurring in terms with maximal weight. This utility score is inspired by the Interactive Boolean Evaluation algorithm of [15], which does not compute a score per variable, but directly selects the lowest probability variable in the highest probability term, breaking ties arbitrarily. Thus, as desired, the variable it selects coincides with our maximal utility variable. In [31], the algorithm of [15] was extended to account for multiple formulas and multiple variable occurrences, by taking the conjunction of Boolean expressions; this coincides with our utility, which is based on the *overall* likeliest terms.

General. In contrast with the Q-Value utility function (Formula (1)), which balances between proving that expressions are True or False by assigning high scores to variables that promote *either* of them, we now take a different approach that, with each step, aims to promote *only one* of them. For that, we use two

separate functions: one of the functions is higher for variables that advance the evaluation towards proving that expressions are True, and the other does the same for False. If we knew that all the expressions are True (or False), the corresponding function would be preferable; but since we do not know the truth values of expressions, and since some expressions may be True and some False, we use the two functions *alternately*.

We have observed above (RO utility function) that to prove the falseness of an expression φ, we need to find one False variable in each of its DNF terms. We therefore use, as the utility score of a variable x, the number of terms that will be set to False if x is False. To account for proving that multiple expressions are False, we count the *overall* number of terms set to False in the expression set Φ. We multiply this by the probability that x is False. Formally,

$$\text{util}(\Phi, \tilde{\pi}, \text{val}, x) = (1 - \tilde{\pi}(x)) \sum_{\varphi \in \Phi} \{\text{nt}(\varphi) - \text{nt}(\text{val}_{\text{x=False}}(\varphi))\} \tag{3}$$

For proving the truth of Boolean expressions, we can similarly use a function that counts CNF clauses set to True; however, unlike the Q-Value function, we can avoid here the expensive computation of CNF by using Formula (2), which, as explained above, also promotes proving that expressions are True. This function is inspired by the Interactive Boolean evaluation algorithm of [4], which alternates between two algorithms, trying, respectively, to prove that a given Boolean expression is True or False.

Example 5.2. *Recall our running example (the input database of Table 1 and query output of Table 2). To exemplify utility computation, we describe it for the utility functions* `RO` *and* `General`*. Assume that at the current point of evaluation:* $X_{\text{known}} = \{a_0\}$ *(i.e., only* a_0 *has been probed),* $\text{val}(a_0) =$ *True, the estimated probability* $\tilde{\pi}$ *of* a_1, r_1, e_1, r_4, e_4 *is 0.1, and* $\tilde{\pi}$ *of the rest of the variables is 0.9. We can now use our utility functions to assign scores to the possible next probes. For instance, if we use the* `General` *utility function, and at this point the choice is made by Formula (3), then* a_1 *gets the maximal utility* $(1-0.1)(1+2) = 2.7$*. Recall that this function targets proving the falseness of Boolean expressions – indeed,* a_1 *is likely to be False (first multiplicand in the above computation) and in this event to falsify a maximal number of terms (second multiplicand).*

If, alternatively, the choice is made by Formula (2), used in the `RO` *utility as well as alternately in the* `General` *utility, then* e_0, e_2, e_3, r_0 *and* r_2 *get the same, maximal utility score* $(1 - 0.1) + \alpha(0.81/2 + \varepsilon) = 1211.5$*. The reason is that after assigning True to* a_0 *and simplifying, all of these variables participate in terms of two variables with probability 0.9, e.g.,* $r_0 \wedge e_0$*, whose weight is accordingly* $.81/2$*, and are intuitively the likeliest terms to be evaluated to True by the current estimation of* $\tilde{\pi}$*. Note that in both cases, the corresponding Boolean Evaluation algorithms of [4, 15, 31] would have arbitrarily chosen one of the variables with highest utility. However, in our case we are able to break ties or decide among variables with similar utility by considerations of actively learning the probabilities, as discussed next.*

6 Probe Selector

The modules described thus far quantify, at each point, the expected utility and estimated uncertainty reduction associated with each candidate probe. The

role of the Probe Selector is then to select probes based on these two factors, balancing between exploitation (probes utility) and exploration (probes uncertainty reduction). Unlike classic exploration-exploitation problems such as Multi-Armed Bandit [92], exploitation in our case has a different flavor, since each probe can be selected at most once and its answer may eliminate other candidate probes. The Probe Selector will thus implement a function f evaluated over $\text{util}(\Phi, \tilde{\pi}, \text{val}, x)$ (the utility of x for Boolean evaluation) and $\text{uncertainty}(x, \text{meta}(x))$ (the effect of x on the uncertainty of the Learner) designed to balance the optimization goals. Let x, x' be variables in X and denote $u = \text{util}(\Phi, \tilde{\pi}, \text{val}, x)$, $u' = \text{util}(\Phi, \tilde{\pi}, \text{val}, x')$, $v = \text{uncertainty}(x, \text{meta}(x))$ and $v' =$
$\text{uncertainty}(x', \text{meta}(x'))$. We identify two desiderata on f, formalized as follows, for a certain $\varepsilon \geq 0$.

1. *Monotonicity.* If $u \geq u'$ and $v \geq v'$ then $f(u, v) \geq f(u', v')$.
2. *ε-Convergence to Utility (ε-CtU).* If $v, v' \leq \varepsilon$ and $u > u'$ then $f(u, v) > f(u', v')$.

Intuitively, the monotonicity desideratum requires that probes that are both more useful for evaluation and for learning are preferred. The ε-CtU criterion formalizes the assumption that with sufficient training the learner computes probabilities that are "accurate enough", and thus uncertainty reduction is no longer a consideration in probe selection. In this case, when uncertainty is smaller than some threshold ε, probes should be ranked according to their utility only.

We next list possible choices for the function f that balance between the utility (u) and the uncertainty (v), and satisfy the above desiderata.

- $f(u, v) := u \cdot (v+1)$. This function is in line with previous work on combining scores in Information Retrieval (e.g., [38]). The use of $+1$ is required for the fulfilment of the CtU desideratum: as the model becomes more accurate and uncertainty reduction decreases, this function approaches the utility score.
- $f(u, v) := \alpha u + \beta v$, a linear combination of the factors, also very commonly used in IR [83, 98].
- $f(u, v) := u$, which vacuously fulfills both desiderata, yet is based only on utility.
- $f(u, v) := \text{I}_{v \leq \Theta} u + \text{I}_{v > \Theta}(v + \mathsf{MAX\_UTIL})$. This function chooses according to uncertainty if the estimated reduction is greater than some predefined threshold Θ, and according to utility otherwise, using indicator functions. The addition of the maximal utility $\mathsf{MAX\_UTIL}$ (which depends on the utility function) is required for the fulfilment of the monotonicity desideratum.

In our experimental study, we have tested all of these functions and have empirically chosen $u \cdot (v + 1)$, which outperformed the rest. Intuitively, the reason was that this function converges faster to the utility function compared with $\alpha u + \beta v$, but still gives some weight to improving the learned model in early stages of the evaluation.

Parallel probe selection. So far, we have assumed probes are selected sequentially based on the answers to previous probes and on the Learner that is trained over previous probes. However, in many cases we can issue probes concurrently without compromising the primary optimization goal, namely the total number of probes. Specifically, when we can partition the set of provenance expressions into subsets that are disjoint in terms of their variables, we verify them separately and concurrently. During the iterative evaluation process, and even if no partition was initially possible, tuple correctness becomes increasingly known, and so variables, terms and expressions are simplified away. We are thus typically able to gradually partition our set of Boolean expressions into smaller subsets handled concurrently until evaluation is complete.

Of course, for enhanced parallelism, such splitting may also be done when the expressions are not fully disjoint; but then sub-optimal probes may be selected. It is left to future work to study this trade-off between the number of probes and parallelism.

7 Experimental Study

We have implemented our solutions in a prototype system and use it in our experimental study to test the performance of our solutions in realistic scenarios. Our implementation uses Python 3.7 with the Scikit-learn package (`https://scikit-learn.org`) for the Learner, using 100 different trees in the RF by default. We have used boolean.py (`https://pypi.org/project/boolean.py/`) to handle Boolean operations, and MongoDB (`https://www.mongodb.com/`) for the database.

7.1 Experimental Setting

We next describe our experimental settings, and the results follow.

Datasets. We show results for two benchmarks. First, the `NELL` dataset [69] includes 50M facts in the format entity-relation-value, e.g., `Volkswagen acquired Audi`, extracted from textual Web sources. A subset of 1.3M facts from this data includes manual (in)correctness labels, and we used this subset including the manual labels that served as our ground truth oracle answers.[1] The available metadata includes the data source and categories, as well as metadata derivable from the data itself such as values and relation names. The data also includes correctness probabilities, but by default, we learn correctness probabilities from scratch (see a comparison in Section 7.2).

Our second dataset, `TPC-H` (SF1) [89], includes ∼8M tuples. This dataset does not include probabilities nor correctness indications, and therefore we generate them synthetically, using one of two options: By default, we construct probabilities using a *random decision tree*: inner leaves are random decisions based on metadata, and the leaves are randomly drawn probabilities. For each tuple, we apply the decision tree on its metadata to obtain a probability and then randomly draw a correctness value according to this probability. This simulates a realistic setting when correctness probabilities are affected by the

[1] `http://rtw.ml.cmu.edu/rtw/resources`

```
SELECT DISTINCT  a.team, b.sport, c.league
FROM athleteplaysforteam  as a, athleteplayssport  as b,
     athleteplaysinleague  as c, generalizations  as d
WHERE  a.athlete = b.athlete AND a.athlete=c.athlete AND
       d.entity = b.sport AND
       (d.value LIKE  '%sport%' or d.value LIKE '%hobby%')
```

Figure 4: NELL Query MS1

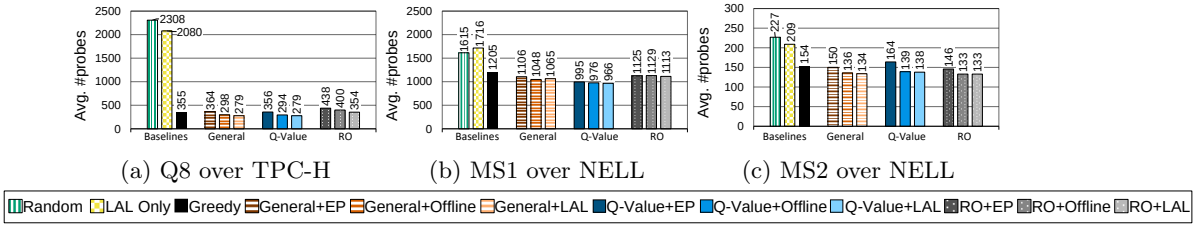
data/metadata. In some experiments, we isolate the effect of Boolean expression structure or probability values by using a fixed probability (0.5 by default), and drawing a correctness value for each tuple with this probability.

Query workload. We have conducted experiments on 18 queries. From the TPC-H query workload, we retained queries Q1-Q10 (see [89]), which are without nesting or negation. We stripped out aggregation (GROUP BY without aggregation is equivalent to projection). NELL does not include a query load, thus we have created one by hand, as follows.

We observed that a main factor in our context is the *skewness* of the resulting provenance: namely, whether there is a small set of variables, that together cover all DNF terms, i.e., at least one of them occurs in every term. We use the term *cover size* to denote the size of this set, where a smaller cover size indicates greater skewness. We then categorize queries as skewed (cover size ≤ 10), moderately skewed (cover size 11-50) and non-skewed (larger cover size). We have split the queries into three groups, according to the skewness of their provenance. In TPC-H, Q1, 3, 4 and 6 are unskewed; Q5, 7 and 8 are skewed; and Q9-10 are moderately skewed. Q1 and Q6 of TPC-H are SP queries and hence their provenance is read-once. Their provenance is in the form of disjunctions so all of our baselines require only a few probes for verifying the correctness of their output. For NELL, we created queries by choosing relations for which there were comparatively many facts (e.g., for company acquisitions as in our running example, and for sports), and then added natural join and projection. See, e.g., an example query in Figure 4, selecting teams with their corresponding sport and league. Following our observations about the impact of skewness in TPC-H, here too we selected, based on their provenance, representative queries with different levels of skewness, and named them accordingly (S, MS and NS for skewed, moderately skewed and non-skewed respectively).

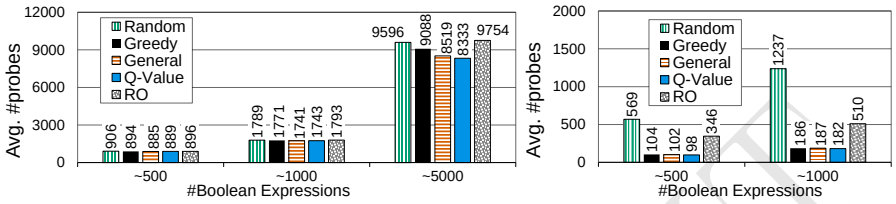
For lack of space, we restrict our analysis to 5 representative queries of the two datasets and three skewness groups. Table 3 shows statistics for these queries, namely, the number of expressions in the output provenance (corresponding to output tuples), the number of unique variables occurring in these expressions and the cover size. In the latter, we use "-" for queries where a cover of size < 50 was not found (non-skewed queries).

Compared Solutions. Since our problem setting is novel, we are not aware of existing systems that can be compared directly with ours (see Section 8). Following our intractability results in Section 3, we cannot compare our algorithms to an optimal solution, even if we have an accurate probability estimation. We thus compare the performance of the following solutions. First, as a sanity

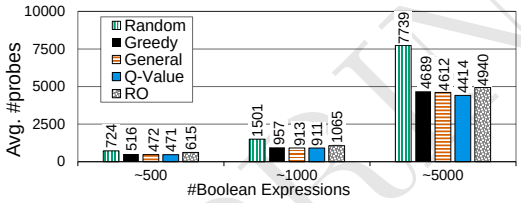


(a) Q8 over TPC-H (b) MS1 over NELL (c) MS2 over NELL

Figure 5: Overall performance comparison of our solutions for example queries



(a) Q3: non-skewed example (b) Q8: highly skewed example



(c) Q10: moderately skewed example

Figure 6: Effect of varying sizes of result subsets, for TPC-H queries

check, we consider baselines that do not follow our framework design, and in particular ignore variable probabilities.

- **Random.** Variables are probed in a random order.
- **Greedy.** Variables with the maximal number of occurrences in the DNF provenance are greedily probed.

Next, we examine solutions that instantiate our framework with a combination of choices for its modules. The function used by the Utility Computation Module is one of `RO`, `Q-Value`, and `General`, described in Section 5. For the *Learner* module, we use a Random Forest classifier with LAL, and vary the (re-)training approach:

- **EP.** Learning is never executed, and the learner returns equal probability 0.5 for every variable.
- **Offline.** The learner estimates probabilities based on the initial probe repository, but does not update the model based on subsequently obtained information.
- **LAL.** The classifier is retrained with each probe answer to refine probability estimation, and LAL is applied to recompute expected error reduction.

Dataset	NELL		TPC-H		
Query	MS1	MS2	Q3	Q8	Q10
# Expressions	469	93	11,895	2,602	41,253
# Unique variables	1,991	232	50,782	68,797	201,725
Term Size	4	4	3	8	4
Cover Size	13	17	-	6	25

Table 3: Statistics for representative queries

The *Probe Selector* module can combine uncertainty and utility or use only one of them. When active learning (LAL) is used, by default, we combine the utility score u and uncertainty reduction score v to prefer probes that both advance Boolean evaluation and learning. For that, we use the function $u{\cdot}(v{+}1)$, described in Section 6. Multiplying by $v+1$ ensures that as the model becomes more accurate and uncertainty reduction decreases, probe selection is dominated by the utility score. When active learning is not used (`EP` and `Offline`), the selector chooses probes based on utility only. We further examine a **LAL only** approach where the Probe Selector uses only the learner uncertainty, like in standard active learning.

Combinations of these choices yield full instantiations of our framework. For example, `Q-Value`+`Offline` uses the `Q-Value` utility function, learns the probabilities based on initial data only and selects probes based on utility only.

General setting and default parameters. All experiments were run on a PC with Intel processor i7-1260P (2.10 GHz) and 32GB of DDR4 memory. The reported results for experiments that involve randomization are averaged over at least 10 executions. We populate the Known Probes Repository (see Section 4) with a fixed number of initial probes (by default 1280) drawn randomly from tuples *not* participating in the provenance of query results. These are used as training data for the Learner Module.

Pre-processing. Our execution can be divided into two phases. In the preprocessing stage, we execute the query and compute the provenance of its output tuples. This step requires under 30 seconds on average and up to 1.5 minutes for TPC-H, our largest dataset. A particular preprocessing step required for `Q-Value` is the conversion of DNF provenance to CNF, which was executed for most output tuples directly (95-99%, depending on the query). For the remaining tuples, the computation exceeded our bound for the number of CNF clauses, so we have split their provenance into smaller Boolean expressions, to allow comparison with `Q-Value`: given a large DNF expression $\varphi = \bigvee_{l=1}^{m}(\wedge_{j=1}^{k} x_{lj})$, we partition its terms into smaller DNF expressions $\varphi_1 = \bigvee_{l=1}^{B}(\wedge_{j=1}^{k} x_{lj}), \varphi_2 = \bigvee_{i=B+1}^{2B}(\wedge_{j=1}^{k} x_{lj}), \ldots$ i.e., disjunctions of at most B terms of φ, where B is a fixed bound and the choice of terms is done randomly. As a result, the CNF of each φ_i is of size $\mathrm{O}(B \cdot k^B)$ rather than $\mathrm{O}(m \cdot k^m)$. Evaluating all of the φ_i expressions determines the value of φ, but may require more probes. We experimentally show below that for most queries it is still worthwhile to perform splitting and use `Q-Value`.

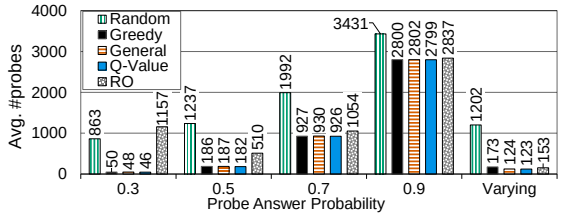


Figure 7: Effect of answer probabilities (Q8)

Metrics. Recall that our algorithms are correct by design as they find the truth value of all expressions. In the experimental study below, we focus on number of issued probes, which is our optimization goal. For completeness, we also report execution times.

7.2 Overall Performance Evaluation

We start by analyzing the overall performance of our solutions, and then dive into the contribution of each component. Figure 5 shows the overall performance of the baselines and instantiations of our framework over representative queries: Q8 of TPC-H and MS1 and MS2 of NELL.

First, observe the improvement contributed by each component. For instance, the TPC-H database includes 8M tuples. If we focus only on tuples participating in the provenance of Q8, we are left with 68K tuples (i.e., unique variables) to verify. Using naïve sequential probe selection over the provenance, e.g., by `Random`, reduces this number to ∼2500. Considering the Boolean expression structure reduces it to ∼350 (`Greedy` and `EP` variants). If we further use probability learning we need only ∼300 probes (Offline variants) and finally active learning saves 20 more probes (`Q-Value` and `General`+LAL). A similar gradual gain for each component is observed for the other queries and datasets, where the percentage of gain depends on properties such as the provenance size and shape.

`Q-Value` or `General` with active learning are generally the strongest competitors across different settings and queries. Per choice of utility function, we can see the gradual decrease in the number of probes between the `EP`, `Offline` and LAL choices, resulting from adding probability estimation and active learning, respectively. The `Random` and `Uncertainty-only` baselines perform poorly in every setting. `Greedy` is outperformed by a smaller gap: it makes 27% more probes in Figure 5a, 25% in 5b and 15% in Figure 5c. This is still significant if we consider that answering probes incurs human effort, monetary cost and/or latency on a larger scale than probe computation.

Omitted from the graph is the performance of variants using the probabilities provided with NELL instead of learning them, which perform about 10% more probes than with active learning, for the three utility functions. For example, `General` performs 150 probes on MS2 with NELL probabilities, compared with 134 probes with our Learner and LAL. In light of the substantial manual effort invested in curating NELL, *this exemplifies that resolving a smaller, query-relevant subset of the data can achieve higher quality with a fraction of the effort.*

	Avg.	Median	Max.	90th %ile
`Learner`	0.329	0.295	1.151	0.463
`LAL`	0.186	0.161	0.855	0.297
`Q-Value`	0.117	0.063	3.601	0.156
`General`	0.048	0.014	1.721	0.088
`RO`	0.022	0.011	0.199	0.068
`Selector`	0.01	0.007	0.106	0.018

Table 4: Execution times per probe (seconds), Q8

Execution times. We have also measured the execution times of each component of our framework. The results for a representative query (Q8) appear in Table 4. The table shows the average, median, maximum and 90th percentile time it takes to execute each of our components in a choice of a probe along the execution, which is typically well under a second. We report times separately for the Learner (including the retraining of the classifier and probability estimation) and LAL (including uncertainty reduction estimation), and for our different utility functions. The total time it takes to choose a probe depends on the choice of components, but even for the most costly such combination (Q-value with LAL, Learner and Probe Selector), the total cost of the 90th percentile is 0.9 seconds. As another example, the `Offline`+`General` combination (`General` with the Probe Selector) takes 0.06 seconds on average to select a probe. Generally, the few most costly probes are the first probes, and execution time drastically decreases as we progress. Overall, the total probe selection time is marginal with respect to the time it would take to answer probes by humans.

7.3 The Effectiveness of Utility Computation

We now analyze the contribution of the Utility Computation module in isolation, by using a Learner sufficiently trained to produce accurate probabilities.

The effect of output size and skewness. To examine the effect of the number of tuples in the query output, in isolation from other factors, we have executed our algorithms on T output tuples (500, 1000, and 5000 tuples) selected uniformly at random, along with their provenance, from the output of different queries; this resembles, e.g., the use of a `LIMIT` operator in SQL (over a random ordering of the output). Figure 6 shows the results for this experiment. We observe that the number of probes generally grows *sub-linearly* in the output size, since while there are more expressions to evaluate, each probe may advance the evaluation of more expressions. We also observe that trends across different queries are affected by the skewness of variable frequencies: Q3 yielded a non-skewed provenance, where each variable repeats, if at all, only a few times. As shown in Figure 6a, the difference (absolute and relative) between algorithms increases with the number of output tuples that need to be verified, up to a difference of 1421 probes (17%) between `Q-Value` and `RO` over 5000 output tuples. In contrast, Q8 and Q10 are highly and moderately skewed respectively, i.e., some of their variables are much more frequent than others. Consequently, in Figures 6b and 6c, we can see that algorithms that do not take variable

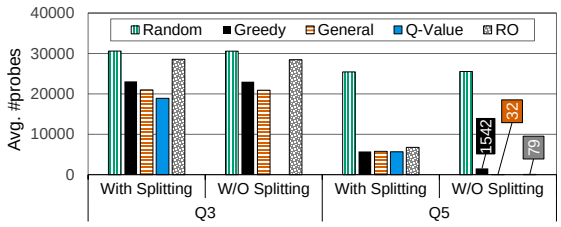


Figure 8: The effect of splitting large Boolean expressions

frequency into account (`Random`, `RO`) are outperformed by a large margin by the others. E.g., `Random` makes 580% more probes than `Q-Value` in Figure 6b. The gap between our algorithms and `Greedy` also grows. E.g., `Greedy` makes 13% more probes than `Q-Value` for 5000 output tuples in Figure 6c.

The effect of probe answer probabilities. Figure 7 shows the number of probes issued by our different algorithms over 1000 output tuples from the result of Q8 for equal probabilities 0.3-0.9 as well as varying probabilities (yielded by the Random Decision Tree). Observe that all the algorithms make more probes with the increase in probability, since it decreases the number of False terms that are easier to eliminate.

The relative performance of `RO`, however, improves as the probability increases (from 2415% more probes compared with `Q-Value` for probability 0.3 to 1% for probability 0.9), since its strategy is to find terms likely to be evaluated to True. `Greedy` performs well for fixed probabilities, but when probabilities vary it is outperformed by the other algorithms (making, e.g., 41% more probes than `Q-Value`).

The effect of expression splitting. Recall that to allow CNF computation and hence comparison with `Q-Value`, we have split long Boolean expressions. The majority of queries have yielded some expressions that needed to be split, but their number was small (e.g., 2% of the expressions in Q3, 5% in Q10). In Figure 8 we analyze the performance of algorithms with and without splitting, for the two queries where the largest portion of splits was performed. Since, as explained in Section 7.1, in both cases the precise correct query output is computed (we can recover the correctness of the original output tuples from the split ones), we compare the effect of splitting on the number of oracle calls. Without splitting, we show only the performance of other algorithms and not of `Q-Value`. For Q3 (and all other queries we tested), performance was very similar with and without splitting. An exception was Q5, which originally included only 5 large expressions and hence required significant splitting. In this case, `General` achieves the best performance by a large gap and improves both absolutely and relatively to other algorithms; and `Greedy`, which performs comparatively well without splitting, performs poorly (it makes 4718% more probes than `General`).

7.4 The Effectiveness of Learning

We now turn to examining the effect of learning probabilities on the number of issued probes. In Figure 9, we compare three configurations that differ only in

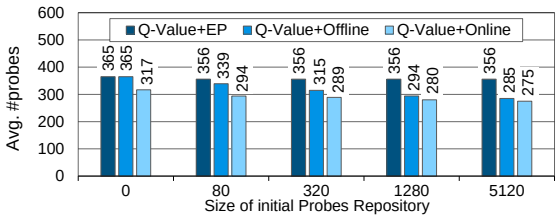


Figure 9: The effect of learning (Q8)

their mode of learning: no learning (`EP`), learning only from the initial Known Probes Repository (`Offline`) and retraining with each iteration (`Online`). Each of the three is combined with the same Utility Computation Module (`Q-Value`) and the same Probe Selector (selecting only based on utility). We further vary the size of the initial Known Probes Repository, to compare the effect of offline and online learning. Since for a small initial repository, there is insufficient data to properly train the RF, we use `EP` to select probes until the probes repository is of size at least 20; i.e., for an empty initial Probes Repository, the `Offline` variant is not trained at all and identical to `EP`, and the `Online` variant executes 20 iterations selecting probes as in `EP` and then starts performing learning with each iteration. We observe the following.

1. In all settings, `Online` is better than `Offline`, which in turn is better than `EP`. Importantly, we observe that the effect of online probes (collected during the current session) on performance is more significant than increasing the initial Known Probes Repository. E.g., given a Known Probes Repository of size 1280, `Offline` has issued more probes than `Online` has issued when given a Known Probes Repository of size 320. `Online` with an empty initial Repository performs less probes than `Offline` with an initial repository of 80 probes.

2. When we increase the size of the initial Known Probes Repository, both learning variants improve, and `Offline` narrows the gap from `Online`, from 15% additional probes to only 3%, since initial probabilities become more accurate. The performance of `Online` is less affected by the size of the initial repository, which suggests that `Online` is effective at performing the right probes.

This experiment also shows the potential effect of available metadata: `EP` simulates a scenario where the available metadata is irrelevant for correctness estimation; `Offline` represents learning from tuples not used in the query; and `Online` learns from tuples that are used in the query. We have analyzed the importance of different attributes for learning using the feature importance tool provided with the Scikit-learn package. For example, for the NELL dataset, recall that facts are in the form entity-relation-value. For `Offline` and `Online` learning, the most important features were `entity` and `value`. Entities and values repeat across tuples, so e.g., if a tuple stating that `kevin_garnett` plays basketball is correct, this increases the correctness likelihood of other tuples regarding kevin_garnett, since at least we know this is a valid entity name. This can explain the empirical advantage of `Online` over `Offline`, since the entities and values that we see during online learning are more relevant to the query and thus also more relevant for estimating the correctness probability of other tuples used in the query. Next in importance was the data source, which may have only been third since it was missing for many tuples. We have also observed

a gradual increase in the importance of `relation` with online learning, again, since many tuples used in a given query are from the same relation. Note that features like `entity`, `value` and `relation`, which are extracted from the data, are always available for learning.

7.5 Conclusions from the Experiments

Our primary goal in this paper is to identify the precise set of ground truth query results while minimizing the number of oracle probes. Across multiple datasets and queries, our solutions were able to do so using a few hundreds of probes, which can be handled by a data expert; or a few thousands of probes, which can be handled by crowdsourcing (as was the case in other data cleaning solutions, e.g., [60, 93]) or automated oracles. In any case, the cost of our algorithm is much less than the alternatives of verifying the full database or even just the tuples participating in the derivation.

We are also able to characterize which of our solutions to use in different cases. If the provenance per tuple is small, with only a few larger expressions (that can be split), `Q-Value+LAL` is recommended as the best performer in most experiments. Otherwise, `General+LAL` is recommended since it is more scalable, and in particular does not require CNF computation.

Finally, our results shed light on solutions to other variants of our problem setting. In a setting when users are willing to relax the task in order to save further probes, they can do so by (i) limiting the set of output tuples, and particularly discarding output tuples with "costly" provenance (non-skewed), following the results of Figure 6; and (ii) modifying the query, e.g., by adding projection to obtain fewer and larger expressions following Figure 8. In a setting when users are willing to issue more probes in order to optimize computation time, we can observe that algorithms that do not involve online learning are faster; among these, `General+Offline Greedy` are fast options that issue a low number of probes (though higher than our solutions that use LAL, see e.g., Section 7.2).

8 Related Work

Multiple lines of work are related to ours. Some of these are used as building blocks in our solutions (the model of Boolean data provenance, probabilistic databases and possible worlds), others are adapted to our framework (algorithms for Interactive Boolean Evaluation and Consent Management), and others are just similar in terms of high-level approach (characterizing query classes based on their provenance structure). Yet we stress that our contributions are all novel, and the overall solution is the first, to our knowledge, that is designed to start with an uncertain database and get to correct SPJU query answers by actively selecting tuples to verify.

Data cleaning. There is a large body of work on identifying and correcting errors in databases [51], and solutions vary in terms of which errors are detected (e.g., constraint violations [2, 11, 21, 37, 66, 81] or statistical analysis [97]), what means are used to correct them (e.g., algorithmic [2, 12, 21, 37, 39, 66, 81] or Machine Learning-based [71, 80]), and the involvement of humans in the

loop (e.g.,[8, 11, 19, 20, 70, 22, 32, 60, 64, 93, 96]). Most of these solutions aim to clean the database as a whole, whereas we aim to resolve a minimal fragment relevant to given queries.

Some work within this line adopts a query-guided approach for cleaning, e.g., [11, 93, 96], but does not use probabilities to decide which query-relevant tuples to verify. [19, 70, 64] and followup work perform query-guided cleaning of probabilistic databases, but in a sense, tackle a problem *dual* to ours: maximally reducing the uncertainty under a cleaning budget. For this reason, and for other differences in their settings, their analysis and solution techniques are very different from ours, and cannot be directly used in our experimental setting. For instance, they focus on queries such as selection, MAX [19], and Top-k [70] where each output tuple corresponds to an input tuple and output probabilities can be efficiently computed (entity-based queries). For general SPJU queries, considered in this work, this is not the case [87], and in fact, a key challenge that our solutions address is the intricate dependence of output on input in the selection of tuples to resolve.

Uncertain databases and data provenance. There is a large body of work on query evaluation with respect to uncertain or probabilistic databases [3, 10, 24, 57, 75, 87, 27] where, as explained in the Introduction, the probabilities of tuples in the query results are computed based on the input probabilities. Additional work in this field is related to Conditioning Probabilistic Databases which deals with computing the uncertainty if conditions such as FDs are added to the uncertain database [91, 56, 72, 88, 100]. As constraints add information and eliminate possible worlds, they may reduce the uncertainty of the probabilistic mode. We are inspired by these works in the use of provenance [47, 52] to derive output correctness and in some properties of the probabilistic model (e.g. possible worlds). Yet, these works do not aim at the active revelation of ground truth, and hence cannot be compared to our solutions theoretically or experimentally. We also mention *consistent query answering* (e.g., [7, 45, 63, 95]), which seeks certain answers that appear in the query result when evaluated with respect to *every repair* of an inconsistent database. Some of this work deals in particular with probabilistic databases [63, 95] and with approximating the certain answers [45]. Our work differs from this line in several respects: first, since we reveal the ground truth answers, our answers are a *superset* of the consistent answers. Moreover, consistent query answers are typically defined with respect to constraints on the legal repairs, whereas in our case we do not build on such constraints. Consequently, our techniques and results are also very different.

Crowdsourced databases. Works on crowd data-sourcing involve an interplay of algorithmic and human processing. Database systems such as [36, 62, 67, 76] execute and optimize queries that include operators such as filter, join, and sort, which are implemented as crowdsourcing tasks. There is also work on crowdsourced data cleaning [11, 19, 20, 70, 22], fact checking [49], Entity Recognition [61] and other related tasks (reviewed in "Data cleaning" above). The present work relates to this research area, if oracles are implemented by crowdsourcing. In particular, as in other crowdsourcing work, the involvement of humans necessarily increases latency and results are not instantly obtained;

yet when human involvement is required, as in mission-critical tasks, these tools and ours are helpful in reducing cost and effort.

Interactive Boolean Evaluation and Consent Management. The problem of Interactive Boolean Evaluation, which is a component of our framework, has been studied extensively in different contexts. These include system testing, e.g., [15, 90], BDD design, e.g., [23, 34, 55], active learning [43] and its connection to other problems such as Stochastic Set Cover [28, 53]. The only work that we are aware of that use Interactive Boolean Evaluation in databases is in [31], in the context of consent management [14, 33, 42, 58, 74, 78]. Differently from our work, [31] does not study probability learning, but rather assume probabilities are independent and given. In terms of theoretical analysis, our results in Section 3 refine Theorem III.5 of [31] by showing hardness for more restricted query classes.

9 Conclusions and Future Work

In this paper, we have proposed a modular end-to-end framework for query-guided uncertainty resolution, which involves active learning for selecting input tuples to be sent for verification by an oracle so that query answers may be decided with certainty. We have proposed solutions for each of the identified sub-tasks and our experimental study has shown the efficacy of the solutions in identifying the correct query results while sending for verification only a small fraction of the input database.

We conclude by mentioning a few of the many variants and extensions of the problems we have studied here that are of potential interest. First, in this work, we regarded the oracle answers as the correct ground truth, similarly to other work on interactive data cleaning with experts (e.g., [8, 11, 19, 20, 70, 60, 93]). In our ongoing work, we examine the effect of erroneous/noisy oracles on our correctness results. For example, we observe that not every erroneous probe answer affects the truth value of an output tuple; we study the identification of "critical" (subsets of) variables, that may lead to an error in our results, and the probability of an error. This problem is also related to previous work on learning with noisy oracles (e.g., [26, 44, 86]). Second, we also intend to study the effect of *constraints*, such as Functional Dependencies, on the problem at hand. Constraints are known to have significant effects on the complexity of query processing (and specifically in the context of probabilistic databases), and it is intriguing to explore their effect in our context as well. Third, we have focused in this paper on SPJU queries, which is the class typically studied in the context of probabilistic databases [24, 57, 75, 87, 27]. A future research direction includes extending our results to additional query languages and data models. Last, validation of some tuples may require more effort than the validation of others. Given estimations on the cost/latency associated with each such validation request, we can incorporate them to the probe selection process and examine their effect on our results.

References

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

[2] F. N. Afrati and P. G. Kolaitis. Repair checking in inconsistent databases: algorithms and complexity. In *ICDT*, 2009.

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

[4] S. R. Allen, L. Hellerstein, D. Kletenik, and T. Ünlüyurt. Evaluation of monotone DNF formulas. *Algorithmica*, 77(3), 2017.

[5] Y. Amsterdamer, S. B. Davidson, D. Deutch, T. Milo, J. Stoyanovich, and V. Tannen. Putting lipstick on Pig: Enabling database-style workflow provenance. *PVLDB*, 5(4), 2011.

[6] Y. Amsterdamer, D. Deutch, and V. Tannen. Provenance for aggregate queries. In *PODS*, 2011.

[7] M. Arenas, L. E. Bertossi, and J. Chomicki. Answer sets for consistent query answering in inconsistent databases. *Theory Pract. Log. Program.*, 3(4-5), 2003.

[8] A. Assadi, T. Milo, and S. Novgorodov. Cleaning data with constraints and experts. In *WebDB*, 2018.

[9] A. Bates, B. Mood, M. Valafar, and K. R. B. Butler. Towards secure provenance-based access control in cloud environments. In *CODASPY*, 2013.

[10] O. Benjelloun, A. D. Sarma, A. Y. Halevy, M. Theobald, and J. Widom. Databases with uncertainty and lineage. *VLDB J.*, 17(2), 2008.

[11] M. Bergman, T. Milo, S. Novgorodov, and W. C. Tan. Query-oriented data cleaning with oracles. In *SIGMOD*, 2015.

[12] L. E. Bertossi, S. Kolahi, and L. V. S. Lakshmanan. Data cleaning and query answering with matching dependencies and matching functions. *Theory Comput. Syst.*, 52(3), 2013.

[13] M. Bloodgood. Support vector machine active learning algorithms with query-by-committee versus closest-to-hyperplane selection. In *ICSC*, 2018.

[14] P. A. Bonatti, L. Ioffredo, I. M. Petrova, L. Sauro, and I. S. R. Siahaan. Real-time reasoning in OWL2 for GDPR compliance. *Artif. Intell.*, 289, 2020.

[15] E. Boros and T. Ünlüyurt. Sequential testing of series-parallel systems of small depth. In *Computing tools for modeling, optimization and simulation*. Springer, 2000.

[16] L. Breiman. Random Forests. *Machine Learning*, 45(1), 2001.

[17] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *ICDT*, 2001.

[18] J. Cheney, L. Chiticariu, and W. C. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4), 2009.

[19] R. Cheng, J. Chen, and X. Xie. Cleaning uncertain data with quality guarantees. *PVLDB*, 1(1), 2008.

[20] R. Cheng, E. Lo, X. S. Yang, M. Luk, X. Li, and X. Xie. Explore or exploit? effective strategies for disambiguating large databases. *PVLDB*, 3(1), 2010.

[21] X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. In *ICDE*, 2013.

[22] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye. KATARA: A data cleaning system powered by knowledge bases and crowdsourcing. In *SIGMOD*, 2015.

[23] F. Cicalese, E. S. Laber, and A. M. Saettler. Diagnosis determination: decision trees optimizing simultaneously worst and expected testing cost. In *ICML*, 2014.

[24] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDB J.*, 16(4), 2007.

[25] D. DeBarr and H. Wechsler. Spam detection using clustering, random forests, and active learning. In *CEAS*, 2009.

[26] O. Dekel, C. Gentile, and K. Sridharan. Selective sampling and active learning from single and multiple teachers. *J. Mach. Learn. Res.*, 13, 2012.

[27] G. V. den Broeck and D. Suciu. Query processing on probabilistic data: A survey. *Found. Trends DBs*, 7(3-4), 2017.

[28] A. Deshpande, L. Hellerstein, and D. Kletenik. Approximation algorithms for stochastic Boolean function evaluation and stochastic submodular set cover. In *SODA*, 2014.

[29] D. Deutch, N. Frost, and A. Gilad. Provenance for natural language queries. *PVLDB*, 10(5), 2017.

[30] D. Deutch, T. Milo, S. Roy, and V. Tannen. Circuits for datalog provenance. In *ICDT*, 2014.

[31] O. Drien, A. Amarilli, and Y. Amsterdamer. Managing consent for data access in shared databases. In *ICDE*, 2021.

[32] A. Fariha, A. Tiwari, A. Meliou, A. Radhakrishna, and S. Gulwani. CoCo: Interactive exploration of conformance constraints for data understanding and data cleaning. In *SIGMOD*, 2021.

[33] K. Fatema, E. Hadziselimovic, H. J. Pandit, C. Debruyne, D. Lewis, and D. O'Sullivan. Compliance through informed consent: Semantic based consent permission and data management model. In *PrivOn at ISWC*, 2017.

[34] A. Fiat and D. Pechyony. Decision trees: More theoretical justification for practical algorithms. In *ALT*, 2004.

[35] J. N. Foster, T. J. Green, and V. Tannen. Annotated XML: queries and provenance. In *PODS*, 2008.

[36] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. CrowdDB: answering queries with crowdsourcing. In *SIGMOD*, 2011.

[37] F. Geerts, G. Mecca, P. Papotti, and D. Santoro. Cleaning data with Llunatic. *VLDB J.*, 29(4), 2020.

[38] S. Gerani, M. J. Carman, and F. Crestani. Proximity-based opinion retrieval. *SIGIR*, 2010.

[39] A. Gilad, D. Deutch, and S. Roy. On multiple semantics for declarative database repairs. In *SIGMOD*, 2020.

[40] B. Glavic and G. Alonso. Perm: Processing provenance and data on the same data model through query rewriting. In *ICDE*, 2009.

[41] B. Glavic and G. Alonso. Provenance for nested subqueries. In *EDBT*, 2009.

[42] A. Goldsteen, S. Garion, S. Nadler, N. Razinkov, Y. Moatti, and P. Ta-Shma. Brief announcement: A consent management solution for enterprises. In *CSCML*, 2017.

[43] D. Golovin and A. Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *JAIR*, 42, 2011.

[44] D. Golovin, A. Krause, and D. Ray. Near-optimal bayesian active learning with noisy observations. In *NeurIPS*. Curran Associates, Inc., 2010.

[45] S. Greco, C. Molinaro, and I. Trubitsyna. Computing approximate certain answers over incomplete databases. In *AMW*, volume 1912, 2017.

[46] T. J. Green, G. Karvounarakis, Z. G. Ives, and V. Tannen. Provenance in ORCHESTRA. *IEEE Data Eng. Bull.*, 33(3), 2010.

[47] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, 2007.

[48] H. Guo and W. Wang. An active learning-based SVM multi-class classification model. *Pattern Recognit.*, 48(5), 2015.

[49] N. Hassan, G. Zhang, F. Arslan, J. Caraballo, D. Jimenez, S. Gawsane, S. Hasan, M. Joseph, A. Kulkarni, A. K. Nayak, V. Sable, C. Li, and M. Tremayne. ClaimBuster: The first-ever end-to-end fact-checking system. *PVLDB*, 10(12), 2017.

[50] M. Herschel, R. Diestelkämper, and H. Ben Lahmar. A survey on provenance: What for? what form? what from? *VLDB J.*, 26(6), 2017.

[51] I. F. Ilyas and X. Chu. *Data Cleaning*. ACM, 2019.

[52] T. Imielinski and W. L. Jr. Incomplete information in relational databases. *JACM*, 31(4), 1984.

[53] H. Kaplan, E. Kushilevitz, and Y. Mansour. Learning with attribute costs. In *STOC*, 2005.

[54] G. Karvounarakis, Z. G. Ives, and V. Tannen. Querying data provenance. In *SIGMOD*, 2010.

[55] O. Keren. Reduction of average path length in binary decision diagrams by spectral methods. *IEEE TOCS*, 57, 2008.

[56] C. Koch and D. Olteanu. Conditioning probabilistic databases. *PVLDB*, 1(1), 2008.

[57] D. Koller. Probabilistic relational models. In *ILP*, 1999.

[58] G. Konstantinidis, J. Holt, and A. Chapman. Enabling personal consent in databases. *PVLDB*, 15(2), 2021.

[59] K. Konyushkova, R. Sznitman, and P. Fua. Learning active learning from data. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, editors, *NIPS*, 2017.

[60] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg. ActiveClean: Interactive data cleaning for statistical modeling. *PVLDB*, 9(12), 2016.

[61] J. Kuperus, C. J. Veenman, and M. van Keulen. Increasing NER recall with minimal precision loss. In *EISIC*, 2013.

[62] G. Li, C. Chai, J. Fan, X. Weng, J. Li, Y. Zheng, Y. Li, X. Yu, X. Zhang, and H. Yuan. CDB: A crowd-powered database system. *PVLDB*, 11(12), 2018.

[63] X. Lian, L. Chen, and S. Song. Consistent query answers in inconsistent probabilistic databases. In *SIGMOD*, 2010.

[64] X. Lin, Y. Peng, J. Xu, and B. Choi. Human-powered data cleaning for probabilistic reachability queries on uncertain graphs. In *ICDE*, 2018.

[65] Y. Liu, Z. Li, C. Zhou, Y. Jiang, J. Sun, M. Wang, and X. He. Generative adversarial active learning for unsupervised outlier detection. *IEEE TKDE*, 32(8), 2020.

[66] E. Livshits, B. Kimelfeld, and S. Roy. Computing optimal repairs for functional dependencies. In *PODS*, 2018.

[67] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller. Crowdsourced databases: query processing with people. In *CIDR*, 2011.

[68] A. Meliou, W. Gatterbauer, and D. Suciu. Bringing provenance to its full potential using causal reasoning. In *TaPP*, 2011.

[69] T. M. Mitchell, W. W. Cohen, E. R. H. Jr., P. P. Talukdar, B. Yang, J. Betteridge, A. Carlson, B. D. Mishra, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. A. Platanios, A. Ritter, M. Samadi, B. Settles, R. C. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling. Never-ending learning. *Commun. ACM*, 61(5), 2018.

[70] L. Mo, R. Cheng, X. Li, D. W. Cheung, and X. S. Yang. Cleaning uncertain data for top-k queries. In *ICDE*, 2013.

[71] F. Neutatz, B. Chen, Z. Abedjan, and E. Wu. From cleaning before ML to cleaning for ML. *IEEE Data Eng. Bull.*, 44(1), 2021.

[72] F. Olmedo, F. Gretz, N. Jansen, B. L. Kaminski, J.-P. Katoen, and A. McIver. Conditioning in probabilistic programming. *TOPLAS*, 40(1), 2018.

[73] N. Ostapuk, J. Yang, and P. Cudré-Mauroux. ActiveLink: Deep active learning for link prediction in knowledge graphs. In *WWW*, 2019.

[74] H. J. Pandit, D. O'Sullivan, and D. Lewis. Queryable provenance metadata for GDPR compliance. In *SEMANTICS*, 2018.

[75] K. Papaioannou, M. Theobald, and M. H. Böhlen. Supporting set operations in temporal-probabilistic databases. In *ICDE*, 2018.

[76] A. G. Parameswaran, H. Park, H. Garcia-Molina, N. Polyzotis, and J. Widom. Deco: declarative crowdsourcing. In *CIKM*, 2012.

[77] J. Park, D. Nguyen, and R. S. Sandhu. A provenance-based access control model. In *PST*, 2012.

[78] E. A. Politou, E. Alepis, and C. Patsakis. Forgetting personal data and revoking consent under the GDPR: challenges and proposed solutions. *J. Cybersecur.*, 4(1), 2018.

[79] C. Re, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, 2007.

[80] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré. HoloClean: Holistic data repairs with probabilistic inference. *PVLDB*, 10(11), 2017.

[81] E. K. Rezig, M. Ouzzani, W. G. Aref, A. K. Elmagarmid, A. R. Mahmood, and M. Stonebraker. Horizon: Scalable dependency-driven data cleaning. *PVLDB*, 14(11), 2021.

[82] S. Roy, V. Perduca, and V. Tannen. Faster query answering in probabilistic databases using read-once functions. In *ICDT*, 2011.

[83] R. L. T. Santos, B. He, C. Macdonald, and I. Ounis. Integrating proximity to subjective sentences for blog opinion retrieval. In *ECIR*, 2009.

[84] P. Senellart, L. Jachiet, S. Maniu, and Y. Ramusat. ProvSQL: Provenance and probability management in PostgreSQL. *PVLDB*, 11(12), 2018.

[85] Y. Shen, H. Yun, Z. C. Lipton, Y. Kronrod, and A. Anandkumar. Deep active learning for named entity recognition. In *ICLR*, 2018.

[86] Y. Sogawa, T. Ueno, Y. Kawahara, and T. Washio. Active learning for noisy oracle via density power divergence. *Neural Networks*, 46, 2013.

[87] D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases*. Morgan & Claypool, 2011.

[88] R. Tang, R. Cheng, H. Wu, and S. Bressan. A framework for conditioning uncertain relational data. In *DEXA*, volume 7447, 2012.

[89] TPC-H Benchmark, 2019. `http://www.tpc.org/tpch/`.

[90] T. Ünlüyurt. Sequential testing of complex systems: a review. *Discrete Applied Mathematics*, 142(1-3), 2004.

[91] M. van Keulen, B. L. Kaminski, C. Matheja, and J. Katoen. Rule-based conditioning of probabilistic data. In *SUM*, volume 11142, 2018.

[92] J. Vermorel and M. Mohri. Multi-armed bandit algorithms and empirical evaluation. In *ECML*, 2005.

[93] J. Wang, S. Krishnan, M. J. Franklin, K. Goldberg, T. Kraska, and T. Milo. A sample-and-clean framework for fast and accurate query processing on dirty data. In *SIGMOD*, 2014.

[94] G. I. Webb, E. Keogh, and R. Miikkulainen. Naïve bayes. *Encyclopedia of machine learning*, 15, 2010.

[95] J. Wijsen. Foundations of query answering on inconsistent databases. *SIGMOD Rec.*, 48(3), 2019.

[96] J. Xu, D. V. Kalashnikov, and S. Mehrotra. Query aware determinization of uncertain objects. *IEEE TKDE*, 27(1), 2015.

[97] M. Yakout, L. Berti-Équille, and A. K. Elmagarmid. Don't be SCAREd: use SCalable Automatic REpairing with maximal likelihood and bounded changes. In *SIGMOD*, 2013.

[98] W. Zhang, C. Yu, and W. Meng. Opinion retrieval from blogs. In *CIKM*, 2007.

[99] N. Zheng and Z. Ives. Compact, tamper-resistant archival of fine-grained provenance. *PVLDB*, 14(4), 2020.

[100] H. Zhu, C. Zhang, Z. Cao, and R. Tang. On efficient conditioning of probabilistic relational databases. *Knowledge-Based Systems*, 92, 2016.