Sensitivity Analysis and Explanations for Robust Query Evaluation in Probabilistic Databases

Bhargav Kanagal
bhargav@cs.umd.edu

Jian Li
lijian@cs.umd.edu

Amol Deshpande
amol@cs.umd.edu

Dept. of Computer Science, University of Maryland, College Park MD 20742

ABSTRACT

Probabilistic database systems have successfully established themselves as a tool for managing uncertain data. However, much of the research in this area has focused on efficient query evaluation and has largely ignored two key issues that commonly arise in uncertain data management: First, how to provide *explanations* for query results, e.g., "Why is this tuple in my result?" or "Why does this output tuple have such high probability?". Second, the problem of determining the *sensitive* input tuples for the given query, e.g., users are interested to know the input tuples that can substantially alter the output, when their probabilities are modified (since they may be unsure about the input probability values). Existing systems provide the *lineage/provenance* of each of the output tuples in addition to the output probabilities, which is a boolean formula indicating the dependence of the output tuple on the input tuples. However, lineage does not immediately provide a quantitative relationship and it is not informative when we have multiple output tuples. In this paper, we propose a unified framework that can handle both the issues mentioned above to facilitate robust query processing. We formally define the notions of *influence* and *explanations* and provide algorithms to determine the top-‘ influential set of variables and the top-‘ set of explanations for a variety of queries, including *conjunctive* queries, *probabilistic threshold* queries, *top-k* queries and *aggregation* queries. Further, our framework naturally enables highly efficient incremental evaluation when input probabilities are modified (e.g., if uncertainty is resolved). Our preliminary experimental results demonstrate the benefits of our framework for performing robust query processing over probabilistic databases.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query Processing*; H.2.m [**Database Management**]: Miscellaneous; G.3 [**Mathematics of Computing**]: Probability and Statistics

General Terms

Algorithms, Design, Management, Performance

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGMOD'11, June 12–16, 2011, Athens, Greece.
Copyright 2011 ACM 978-1-4503-0661-4/11/06 ...$10.00.

Keywords

Boolean algebra, Lineage, Probabilistic Databases, Query processing, Sensitivity Analysis

1. INTRODUCTION

Probabilistic databases have established themselves as a successful tool for managing uncertain data and for performing query processing over such data. Models for expressing tuple uncertainty, attribute uncertainty and complex correlations have been proposed in the literature, and a large number of systems [1, 10, 31, 6, 32, 21, 24] have been developed for performing efficient query processing. In each of the different probabilistic database systems, the users specify *probability distribution functions* for the uncertain entities in the probabilistic database. For instance, in Mystiq [10, 30], the user specifies the probability distribution for the tuple-existence using the parameter p, i.e., tuple belongs to the table with probability p and is absent with probability 1 p. In PrDB [32, 24], the user specifies the joint probability distribution for a set of correlated tuples in the database. Given a query as input, the probabilistic database system uses these pdfs to determine the resulting output tuples and their associated probabilities.

Although efficient algorithms have been developed for query processing in prior work, they do not lend themselves to *robust query processing*. We illustrate this with a motivating application of Information extraction and integration [23, 18, 7, 15, 12, 13]. Consider an application which extracts *structured entities* from unstructured text on the Internet and populates a relational database. These systems utilize entity recognition (e.g., identifying people, locations, companies), relationship detection (e.g., affiliations), sentiment analysis (e.g., from reviews), entity resolution (e.g., "Y! Labs" and "Yahoo! Labs" refer to the same entity), and other complex machine learning algorithms. These algorithms rely on probabilistic models such as Bayesian networks, CRFs and similarity logics to assign probabilities to the extracted tuples. Bayesian inference is #P-hard, therefore the probabilities computed are usually approximate (with no guarantees on bounds [9]), and further, similarity metrics are usually ad-hoc. Hence, the probabilities assigned to the tuples are often noisy, imprecise, and erroneous. An example of such a probabilistic database is shown in Figure 1(a,b,c). Given such a probabilistic database, a user may be interested to pose a query such as Q : "List *reputed* car sellers in the *San Jose* area who offer *Hondas*". Existing systems provide answers as shown in Figure 1(d) (Q's result shown). This output relation, however, is not very informative for the user since it does not provide any intuition about two important issues.

1. First, existing systems do not provide *explanations* for the query results – e.g., "Why is tuple t in the output result?" or "Why

does output tuple t_1 have such a high probability, as compared to tuple t_2?". As noted by Re et al. [30], even in a biological domain, where scientific decisions are made based on several uncertain hypotheses, it is critical to know the input hypotheses that contribute significantly to the output decision. Hence, it is essential to provide this information in addition to the actual results, to the users of the system.

2. Second, current systems do not provide the *sensitive* input tuples for a query, i.e., the set of tuples that can potentially modify the result probabilities the most. This information is critical in most application domains that need to handle uncertain data, because of the inexact nature of the probability values. Hence, they would be interested to know the input tuples that are likely to change the output probabilities significantly, when their probabilities are modified. In other words, we need to find the set of input tuples which highly *influence* the output probability values. Providing this information helps the user to focus his/her effort in *procuring more accurate probabilities* for this particular set of input tuples.

Finally, since input tuple probabilities may be modified several times, the system also needs to solve the ensuing problem of supporting incremental updates to the query results by exploiting previously executed computation. For example, the users may choose to resolve uncertainty using data cleaning techniques based on Cheng et al. [5] or by using techniques based on value of information [26]. Or the users may procure more accurate values for the probabilities, e.g., by running more iterations [29] or by querying an expert.

One approach that systems like Trio [31] use to address this problem is to additionally provide the *lineage* of each of the output tuples – a boolean formula which qualitatively explains the reasons for occurrence of the output tuple. However, lineage does not *directly quantitatively* specify the relationship between the output tuple probability and the input tuple probabilities. Also, lineage can be very large (e.g., projection of a million tuples on to a single tuple results in a lineage formula of size one million), hence it is difficult for the user to obtain any useful information from the lineage alone. Re et al. [30] originally considered the problem of computing influential input tuples and explanations for boolean conjunctive queries. They propose alternative forms of *approximate lineage* and develop algorithms to recover influential tuples from them, while our focus is on efficient exact computation based on the original lineage itself. Our proposed definitions of influence and explanations generalize Re et al.(Section 3)'s definition (which is restricted to boolean conjunctive queries) and are applicable to a variety of queries. Our techniques also handle aggregation and top-k queries in addition to conjunctive queries. Recently, Meliou et al. [27] proposed techniques for determining the *causes* for a result tuple and developed algorithms for computing the degree of the cause (called *responsibility*) for conjunctive queries in a relational database (not for probabilistic databases). Their techniques are based on Halpern and Pearl's [20, 19] fundamental definition of causality. While we use a different definition for causality that is more suited for probabilistic databases, both definitions are quite related and can be reasoned in terms of possible world semantics. We discuss the exact relationship between the two approaches in Section 3.4.

In this paper, we develop a robust query processing framework for probabilistic databases by augmenting our probabilistic database system with support for performing low overhead sensitivity analysis and by providing explanations for query answers. Our system provides an option for the user to *mark* a query for performing either sensitivity analysis or explanation analysis over the results of

lid	SID	Name	Address	?
y_1	239	Honda of San Jose	12345	0.3
y_2	239	Honda Company, San Jose	12345	0.3
y_3	231	Ford Company	12207	0.8
y_4	340	Toyota Car Company	12209	0.9

(a) Location

tid	VIN	SID	Model	Price	?
x_1	1A0	239	Honda	3500	0.3
x_2	1A1	231	Honda	4500	0.8
x_3	2B2	231	Honda	4500	0.5
x_4	2B3	231	Toyota	4500	0.4
x_5	2B4	231	Ford	4500	0.1

(b) CarAds

rid	SID	Rep	?
z_1	239	Good	0.3
z_2	231	Bad	0.7

(c) Reputation

239	$x_1(y_1 + y_2)$	0.8

(d) Result for query Q_1

Figure 1: Relations extracted by an information extraction system about car sellers and car advertisements. All tables have *tuple uncertainty*.

the query. When a query is marked for sensitivity analysis, we provide the set of top-ℓ (where ℓ is a user specified parameter) influential input tuples for the query and when a query is marked for explanations analysis, we provide the set of input tuples of size ℓ which provides the best explanation for the query results. We would like to note here that our system currently looks at the problem by initially computing the lineage of the output tuples and subsequently executes sensitivity and explanation analysis. In future, we plan to approach the problem using *extensional* [10] techniques, without explicitly computing the lineage. Our primary research contributions are as follows:

We provide definitions for *influence* of tuple(s) on a query result and *explanations* that are applicable to a wide range of database queries including conjunctive queries, aggregation queries and top-k queries.

Sensitivity Analysis

1. We show that the problem of identifying top-ℓ influential variables for conjunctive queries is #P-complete.
2. For conjunctive queries that lead to *read-once / 1OF* lineage formulas [33, 16, 28], we provide linear time algorithms (in size of the lineage). For the general case, we provide algorithms that are exponential in the *treewidth* [14] of the boolean formula.
3. We develop algorithms for identifying influential variables for aggregation (SUM/COUNT/MIN/MAX) queries.
4. We develop *novel pruning rules* to speed up the computation of influential variables for top-k queries (by probability).

Explanation Analysis

1. We show that the problem of determining the top-ℓ explanations for conjunctive queries is NP-hard in general. As with sensitivity analysis, we develop algorithms for identifying the best explanations for queries that lead to read-once lineages.
2. We develop novel techniques for computing explanations for aggregation (SUM/COUNT/MIN/MAX) queries.

For conjunctive queries and aggregation queries, we provide *incremental* algorithms for re-evaluating query results when input probabilities are modified.

Outline: In the next section, we provide a brief background for probabilistic databases and some of the key concepts we use in the rest of the paper. In Section 3, we formally state the sensitivity analysis and the explanation analysis problems and their semantics. In Section 4 and Section 5, we develop techniques to solve the sensitivity analysis problem and the explanation analysis problem

respectively. In Section 6, we briefly describe some techniques for incrementally re-evaluating query results when input probabilities have been modified. We conclude with some initial experimental evaluation in Section 7.

2. PRELIMINARIES

In this section, we provide background for probabilistic databases and the different concepts discussed in the paper. We start with a description of tuple uncertainty probabilistic databases.

2.1 Tuple Uncertainty Probabilistic Database

We assume a *tuple uncertainty* probabilistic database model similar to Dalvi et al. [10], i.e., each tuple t_i exists with probability p_i. Also, the existence of a tuple is independent of the existence of the other tuples in the database. We associate each tuple t_i in the database with a *binary random variable* x_i such that $x_i = 1$ if t_i belongs to the database and $x_i = 0$ otherwise. Note that $p_i = Pr(x_i = 1)$. Let $P = \{p_1, p_2, \ldots, p_n\}$ be the set of input probabilities.

2.2 Queries

In this paper, we consider conjunctive queries, top-k queries (by probability) and aggregation queries. We describe each in turn.

Conjunctive queries: A conjunctive query is a fragment of first-order logic restricted to $\exists$ and $\wedge$. In SQL, they correspond to `select-project-join` queries, restricted to equi-joins and conjunctions in the `where` clause. As shown by Das Sarma et al. [31], a conjunctive query can be evaluated over a probabilistic database by first computing the *lineage/provenance* for each output tuple, which is a boolean formula that represents all possible derivations of the output tuple, and subsequently evaluating the probabilities of the lineage formulas. The general problem of conjunctive query evaluation has been shown to be #P-complete [10]. However, if the lineage formula can be represented using a *read-once* or 1-OF form [28, 33] (a boolean formula in which each literal appears exactly once), then the probability of the lineage can be computed in polynomial time. For example, the boolean formula $(x_1 \wedge x_2) \vee (x_2 \wedge x_3)$ can be rewritten as $x_2 \wedge (x_1 \vee x_3)$ which is a read-once formula. On the other hand, $(x_1 \wedge x_2) \vee (x_2 \wedge x_3) \vee (x_3 \wedge x_1)$ cannot be rewritten as a read-once formula. A read-once formula can be represented as an *AND/OR* tree as shown in Figure 2. The probability of a read-once formula can be computed using a bottom up algorithm over the AND/OR tree, which computes probabilities of intermediate nodes using the following equations. Suppose that z is a node with children x_1 and x_2:

$$\text{if } z = x_1 \wedge x_2 \qquad p(z) = p(x_1)p(x_2) \tag{1}$$

$$\text{if } z = x_1 \vee x_2 \qquad p(z) = 1 - (1 - p(x_1))(1 - p(x_2)) \tag{2}$$

Given a boolean formula in DNF form, we can check whether it can be represented using a read-once representation in time linear in the size of the formula using the *co-graph recognition* algorithm of Golumbic et al. [16]. We illustrate this algorithm with an example in Section 2.3. If the boolean formula cannot be represented in a read-once format, we use *Shannon expansions* as described by Olteanu et al. [28] in order to compute its probability (See Section 2.4).

Probabilistic Threshold/Top-k Queries: We consider:

1. Probabilistic threshold queries: A conjunctive query Q is specified along with a threshold probability value τ; the output is the *set* of the output tuples of Q whose probabilities exceed τ. We assume that the output probabilities are not part of the result.
2. Top-k queries by probability: This query requires us to return the set of top-k tuples sorted by probability. This is different from threshold queries because in this case, we only return k tuples, whereas threshold queries are not restricted to k output tuples.

Aggregation Queries: The final class of queries we consider are aggregation queries such as SUM, MIN, MAX and AVG. In this paper, we only consider aggregates over a single table containing a set of independent base tuples. Each base tuple has a real valued score attribute A and the above aggregation functions operate on the scores. We use a_i as a shorthand notation for $t_i.A$, the score of tuple t_i. For both the sensitivity analysis problem and explanation analysis problem, we consider the *expected values* version where the output of the query is the expected value of the aggregate. For example, the answer to SUM is $E[\sum_i a_i x_i]$, where x_i is the binary random variable defined in Section 2.1.

2.3 Detecting read-once lineages

In this section, we describe Golumbic's algorithm [16] which takes as input, a DNF boolean expression ϕ and determines if ϕ can be rewritten using a read-once representation. It also returns the read-once rewriting if it exists. The algorithm is *complete*, i.e., if it cannot determine a rewriting, then it does not exist. The algorithm starts by constructing a *co-occurrence graph* of the boolean formula. The co-occurrence graph of the formula is an undirected graph in which the nodes are literals of the formula and an edge exists between 2 literals if they occur together in some clause in the DNF formula. In the next step, the algorithm checks if the co-occurrence graph is a *co-graph* [8]. If yes, then the algorithm computes the *co-tree* representation of the co-occurrence graph, which is the required read-once representation. We illustrate the algorithm with an example.

Suppose we have a boolean formula: $(x_1 \wedge x_2) \vee (x_2 \wedge x_3) \vee (x_3 \wedge x_4) \vee (x_4 \wedge x_1)$. The co-occurrence graph G for the formula is shown in Figure 2(ii). Since the graph is connected, the algorithm creates an AND node ($\wedge$) in the co-tree (Figure 2(iv)) and constructs the complement of G, G^c as shown in part(iii). If G^c is connected, then the formula does not have a read-once representation. Otherwise, it considers each component separately. It creates an OR node in the co-tree for each component, which are made the children of ($\wedge$). The algorithm recursively continues until we reach singleton graphs, which are added as leaves, or we reach a termination due to non-existence of a read-once representation.

A small caveat with this approach is that the algorithm works only for *normal* boolean formulas. A boolean formula is normal if we can reconstruct it from its co-occurrence graph. For example, $x_1 \wedge x_2 \wedge x_3$ is normal but $(x_1 \wedge x_2) \vee (x_2 \wedge x_3) \vee (x_3 \wedge x_4)$ is not normal. Note that the co-occurrence graphs for both formulas are the same. In recent work, Sen et al. [33] have shown that lineage formulas are always normal (for conjunctive queries without self-joins), hence we do not need to make this check. Next, we discuss Shannon expansions of a boolean formula, which we use for handling non-read-once lineages.

2.4 Shannon Expansions

Shannon expansions is a technique for representing a boolean formula as a XOR of two smaller sub-functions of the formula. Using Shannon expansions, we can express a non-read-once formula using a number of (exponential, in the worst case) read-once formulas and thereby compute its probability. Given a boolean formula ϕ, and a variable x that appears in ϕ, we can represent ϕ by means of the identity:

$$\phi = (x \wedge \phi_{x=1}) \vee (\bar{x} \wedge \phi_{x=0})$$

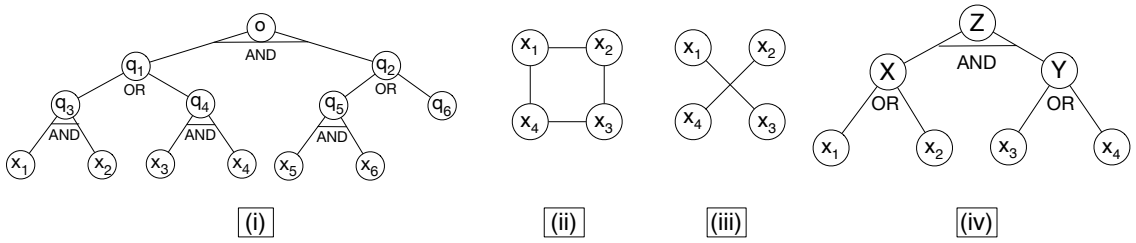

Figure 2: (i) Boolean formula $(x_1x_2 + x_3x_4)(x_5x_6 + x_7)$ represented using an AND/OR tree. Leaves denote variables of the formula, internal nodes are intermediate expressions. (ii, iii, iv) Steps involved in generating the read-once formula for $(x_1 \wedge x_2) \vee (x_2 \wedge x_3) \vee (x_3 \wedge x_4) \vee (x_4 \wedge x_1)$

$_{x=1}$ and $_{x=0}$ are called the Shannon co-factors of w.r.t. x. $_{x=1}$ is the boolean formula obtained by setting $x = 1$ in the formula and $_{x=0}$ is the boolean formula obtained by setting $x = 0$ in the formula. $\bar{x}$ denotes the negation of variable x. The XOR between the terms is because the assignments satisfying the first term are mutually exclusive of the assignments satisfying the second term.

Consider the boolean formula $= (x_1 \wedge x_2) \vee (x_2 \wedge x_3) \vee (x_3 \wedge x_4)$. Suppose we want to expand the formula along x_2. The expansion we get is:
$x_2(\ _{x_2=1})\ \ \bar{x_2}(\ _{x_2=0}) = x_2 \wedge (x_1 \vee x_3)\ \ \bar{x_2} \wedge (x_3 \wedge x_4)$.
Note that both $(x_1 \vee x_3)$ and $(x_3 \wedge x_4)$ are read-once and hence their probabilities can be evaluated easily. Also, since the two terms in the expansion are mutually exclusive, we can simply add their probabilities to get the probability of the original boolean formula.

3. FORMAL PROBLEM STATEMENT

In this section, we formally state the sensitivity analysis and the explanation analysis problems.

3.1 Sensitivity Analysis

We start by defining the notion of *influence* of a given input probability on a query result. Queries on a probabilistic database can be of two categories based on the type of output: *value* queries and *set* queries. We define influence for each query type in turn.

Value queries: The output of a value query is either a single numerical value v or a set of numerical values $\{v_1, v_2, \ldots, v_n\}$. Examples of value queries include boolean conjunctive queries (output value is a probability) and aggregation queries (output value is the expected value of the aggregate).

DEFINITION 1. *The influence of an input tuple* t *on a value query with output* v *is given by the derivative* $\frac{\partial v}{\partial p}$*, where* p *is the probability that* t *exists. If the output is a set of values, then the influence is the sum of the influences of* t *on each output value.*

For conjunctive queries, even if we change the probabilities of the input tuples, the output set of tuples remains the same, only the probabilities of the output tuples change. Thus, the sensitivity of an input probability on a boolean conjunctive query result represents how the output probability changes when the input probability is changed. Influence of an input tuple t on the output tuple with lineage formula , denoted by $\text{in}_t(\ )$ is given by the derivative $\frac{\partial p(\lambda)}{\partial p}$. An alternate definition of influence of a tuple on a conjunctive query, was proposed by Re et al. [30]. Re et al. define influence of a tuple as the difference between the output probabilities obtained in two cases, first by assuming that the tuple exists and second by assuming that the tuple does not exist. As we show in Theorem 1 in Section 4.1.1, the two definitions are equivalent.

For aggregation queries, we measure how the *expected* value of the aggregate changes when the input probability is modified, i.e., for the AVG aggregate, we define influence of t_i to be the derivative $\frac{\partial E[AVG]}{\partial p_i}$.

Set Queries: Examples of Set queries are probabilistic threshold queries and top-k queries. If we change the probability of an input tuple, either new tuples enter the result set or existing tuples leave the result set. To define influence over discrete sets such as these, we introduce the notion of -influence.

DEFINITION 2. *Input tuple* t *with probability* p *is -influential on the output set* S *if using* $p +$ *in place of* p *modifies the result from* S *to* S'*, where* $S' \neq S$*. The degree of influence is the cardinality of the symmetric difference* $S \ \ S' = (S \cap S') \cup (S' \cap S)$.

Note that is a parameter this is provided by the user. In Section 4.3, we show how to provide hints to the user to set .

Formal Problem: Sensitivity Analysis
Given a probabilistic database and a query, determine the set of top-ℓ influential/ -influential variables for the query. We use ℓ to distinguish it from conventional top-k queries.

3.2 Explanation Analysis

Intuitively, an explanation for a query result is a set of tuples which provides the best reason for obtaining the particular result tuple and its associated probability. It is critical to consider a *set* of tuples and their combined contribution rather than contributions by each individual tuple individually. For example, consider an output tuple with lineage $(a \wedge b) \vee (c \wedge d)$. Suppose that probability of a and b are very high compared to that of c. The contributions of a and b treated individually are very high compared to that of c. However, the contribution of the set $\{a, c\}$ is higher than the contribution of $\{a, b\}$ since we can bring the output probability down to 0 by setting a and c to false (which is not possible with $\{a, b\}$). Hence we define the *contribution* for a set of tuples as follows.

DEFINITION 3. *A contribution of a set of input tuples* S *is defined as the change in the output obtained when we set the probabilities of all tuples in* S *to zero.*

For value queries, the change in the output is simply the difference between the two resulting values. For example, in the case of boolean conjunctive queries, this corresponds to the set of tuples which cause the maximum change in the output probabilities if their probabilities are set to 0. For set based queries, the change is the cardinality of the symmetric difference between the resulting sets, as we did with sensitivity analysis. For explanation analysis, we only consider value queries in the rest of the paper. We leave the problem of set queries to future work.

Formal Problem: Explanation Analysis
Given a probabilistic database and a value query, determine the set of input tuples of size ℓ which has maximum contribution among all subsets of size ℓ.

3.3 Warmup: SUM/COUNT

As a warmup, we illustrate the concepts we just introduced using SUM=COUNT queries. Using linearity of expectation, we can see that $SUM = E[\sum_i a_i x_i] = \sum_i a_i p_i$. COUNT is just a special case of SUM where $a_i = 18i$. Therefore, the influence of tuple t_i is simply $\frac{\partial SUM}{\partial p_i} = a_i$. We can just sort the input tuples by their attribute values (i.e., a_i) and return the top-ℓ influential tuples.

It is also easy to see the contribution of the set S of tuples is $\sum_{t_i \in S} a_i p_i$. Therefore, the explanations for COUNT correspond to the ℓ tuples with maximum $a_i p_i$ values.

Recomputing the results for SUM=COUNT queries is also straightforward. For instance, if we change p_i, the probability t_i, to p_i^0, then the new query answer is $E[SUM] \quad a_i p_i + a_i p_i^0$, which can be done in constant time.

3.4 Relation to Meliou et al. [27]

Meliou et al. [27] define the notion of responsibility of a tuple t for a query answer/non-answer as the inverse of the size of the smallest *contingency set* for the tuple. A contingency set is a set of conditions that need to be satisfied for the tuple t to *cause* a difference to the output. The bigger the contingency set, the smaller is the set of possible worlds that the tuple can influence. If the size of the contingency set is s, then the total number of possible worlds that can be influenced is of the order of: $O(\frac{2^n}{2^{s-1}})$, n being the number of input tuples. In our definition of influence, we measure the sum of the probabilities of all possible worlds that are influenced by the given input tuple since all possible worlds need not have the same probability. Hence, for the case of probabilistic databases, the concept of influence is more meaningful, although the two concepts are closely related.

4. SENSITIVITY ANALYSIS

In this section, we discuss our algorithms for solving the sensitivity analysis problem, i.e., computing the top-ℓ influential variables for a given query. In the first part, we discuss value queries and subsequently we discuss set-based queries.

4.1 Value queries

We start by developing algorithms to compute influential variables for boolean conjunctive queries and extend the algorithms for arbitrary conjunctive queries. For ease of exposition, we list the following straightforward lemma that is used throughout the paper. Suppose q is a value query and suppose that the input tuple probabilities are $P = \{p_1, p_2, \ldots, p_n\}$. Let $P_{p_i \quad a}$ be the same vector as P except the ith entry being replaced by a.

LEMMA 1. *If $q(P) = q(p_1, \ldots, p_n)$ is a linear function of p_i, i.e., $q(P) = c \quad p_i + d$ where c, d are constant w.r.t. p_i, then*

$$\frac{\partial q(P)}{\partial p_i} = \frac{q(P_{p_i \quad a}) \quad q(P_{p_i \quad b})}{a \quad b}.$$

for any $a, b \in [0,1]$ and $a \neq b$.

4.1.1 Boolean conjunctive queries

We first show our definition of the influence of a tuple is equivalent to the definition proposed by Re et al. [30]. Recall that they define influence of a tuple to be the difference between the output probabilities obtained in two cases, first by assuming that the tuple exists and second by assuming that the tuple does not exist.

THEOREM 1. *(1) Given a boolean formula Φ, which is a function of input variables $x_1, x_2, \ldots, x_n$, $p(\Phi)$ is linear in each p_i treated individually, i.e., $p(\Phi) = (c_i p_i + c_i^0)$ for each i.*

(2) Our definition of the influence is equivalent to the one proposed by Re et al. [30], i.e., $\frac{\partial \Pr[\lambda(t)]}{\partial p(t)} = \Pr[\Phi_{x_i=1}] \quad \Pr[\Phi_{x_i=0}]$.

PROOF. Consider a boolean random variable x_i which appears in the formula Φ. Using Shannon expansion,

$$\Phi = (x_i \wedge \Phi_{x_i=1}) \vee (\overline{x_i} \wedge \Phi_{x_i=0})$$
$$\Rightarrow p(\Phi) = p(x_i = 1)p(\Phi_{x_i=1}) + p(x_i = 0)p(\Phi_{x_i=0})$$

(This is because the two terms are *mutually exclusive*)

$$\Rightarrow p(\Phi) = p_i p(\Phi_{x_i=1}) + (1 \quad p_i)p(\Phi_{x_i=0})$$
$$\Rightarrow p(\Phi) = p_i c_i + c_i^0$$

Here, c_i and c_i^0 are constants, i.e., independent of p_i. The second part is a easy consequence of the first part and Lemma 1. □

As indicated in Section 2, conjunctive queries are evaluated by first computing the lineages of the output tuples. In the case of boolean queries, we have a single output lineage for which we need to compute the influential variables. According to Theorem 1, the probability of a boolean formula is linear in each input tuple treated individually, i.e., $p(\Phi) = c_i p_i + c_i^0$. Hence, it is enough to determine c_i values for each input tuple and then select the top-ℓ among them. However computing the influence values for all input tuples is #P-complete as shown in Theorem 2.

THEOREM 2. *The problem of computing the influences of all variables for a non-read-once lineage is #P-complete.*

PROOF. We prove via a counting reduction from the problem of computing $p(\Phi)$ where Φ is a k-DNF formula, which is a well known #P-complete problem. Assume that we can indeed compute the influence of all variables on a non-read-once boolean formula Φ of size n in polynomial time. Suppose that the variables in Φ are $x_1, x_2, \ldots, x_n$. Using Theorem 1, can write the probability of Φ as:

$$p(\Phi) = p(x_1)p(\Phi_{x_1=1}) + (1 \quad p(x_1))p(\Phi_{x_1=0})$$
$$= p(x_1)(p(\Phi_{x_1=1}) \quad p(\Phi_{x_1=0})) + p(\Phi_{x_1=0})$$
$$= p(x_1)infl_{x_1}(\Phi) + p(\Phi_{x_1=0})$$

Note that $\Phi_{x_1=0}$ is a boolean formula with $n \quad 1$ variables.

It can be expanded further.

$$= p(x_1)infl_{x_1}(\Phi) + p(x_2)infl_{x_2}(\Phi_{x_1=0}) + p(\Phi_{x_1=x_2=0})$$
$$= p(x_1)infl_{x_1}(\Phi) + p(x_2)infl_{x_2}(\Phi_{x_1=0}) + \ldots$$
$$= \sum_i p(x_i)infl_{x_i}(\Phi_{x_1=\ldots=x_i=0})$$

Hence, computing the influences of a variable for an arbitrary DNF is at least as hard as computing the probability of the formula (#P-hard). Moreover, since $infl_{x_i}(\Phi) = \frac{p(\lambda) \quad p(\lambda_{x_i=0})}{p(x_i)}$, our problem is in #P. Therefore, it is #P-complete. □

Although the general problem is hard, we can devise algorithms for the special case when the boolean formula is *read-once*. We discuss this case first.

Read-once lineage: In this case, we develop a recursive algorithm for computing the influences of all input tuples in $O(n)$ time (n is the size of the lineage formula). Consider the lineage shown in Figure 2(i). In order to compute the influence of x_1 on the output probability, i.e., $\frac{\partial o}{\partial x_1}$, we can use the chain rule from calculus,

$$\frac{\partial o}{\partial x_1} = \frac{\partial o}{\partial q_1}\frac{\partial q_1}{\partial q_3}\frac{\partial q_3}{\partial x_1} \tag{3}$$

The terms on the RHS can be obtained by taking appropriate derivatives of Equations 1 and 2. Suppose z is a node with two children x_1 and x_2. Then,

$$\text{if } z = x_1 \wedge x_2: \quad \frac{@z}{@x_1} = x_2 \;\&\; \frac{@z}{@x_2} = x_1$$

$$\text{if } z = x_1 \vee x_2: \quad \frac{@z}{@x_1} = 1 \quad x_2 \;\&\; \frac{@z}{@x_2} = 1 \quad x_1$$

Algorithm 1 deriv(x), Read as derivative w.r.t x

```
1: if parent(x) = null {x is root} then
2:     deriv(x) = 1
3: else
4:     if parent(x) is an AND node then
5:         deriv(x) = deriv(parent(x)) * Pr(sibling(x))
6:     else
7:         deriv(x) = deriv(parent(x)) * (1 - Pr(sibling(x)))
```

Using the chain rule and the above equations, we develop a recursive algorithm as follows. Each node in the AND/OR tree stores the derivative of the output probability with respect to itself. We use this to compute the derivatives of its children using the above recursive equations in a top-down manner to finally get the derivative with respect to the leaf nodes (input tuples). Note that the probabilities of all the nodes are precomputed in a single $O(n)$ pass as a preprocessing step. The relevant snippet of the algorithm is shown in Algorithm 1. After computing the influences of each of the input variables, we determine the top-ℓ influential variables either by sorting $O(n \log n)$ or by making a linear scan over the input tuples $O(n\ell)$, based on the value of the input ℓ. We also cache the computed influence values for the input tuples – those can be used for quickly recomputing results in certain cases. Although we illustrated the algorithm for binary trees, our implementation can be easily extended to handle k-ary AND/OR trees.

Non-read-once lineage: Next, we consider the sensitivity analysis for non-read-once formulas. We propose a heuristic for evaluating the influences, which is similar to the Dtree construction algorithm of Olteanu et al. [28]. Essentially, we perform a sequence of *Shannon expansions* to expand a non-read-once lineage to a set of *mutually exclusive* read-once formulas. The complexity of the operation is exponential in the *treewidth* [14] of the boolean formula. The complete algorithm is shown in Algorithm 2. We now explain the main aspects of the algorithm.

Algorithm 2 infl(; τ)

```
Require: Boolean formula , influence vector τ
1: if  is read-once then
2:     return infl_read_once( )
3: else
4:     Select boolean variable x in  that repeats most times
5:     Shannon expansion:  = (x ∧ x=1)  (x̄ ∧ x=0)
6:     τ = (1  p(x)) in ( x=0; τ) + p(x) in ( x=1; τ)
7:     τ[x] = Pr( x=1)  Pr( x=0) {influence of x itself}
8: return τ
```

In Step 1, we check if the boolean formula has a read-once representation using Golumbic's algorithm (Section 2.4). If it has a read-once representation, then we use the previous algorithm itself. Otherwise, we expand the boolean formula using Shannon expansion, selecting the variable that appears the most number of times. The expansions of the boolean formula are stored in a binary tree data structure which we call as a Dtree (after Olteanu et al. [28]). Each node in the Dtree corresponds to a boolean formula. Once we obtain a read-once formula, we stop expanding and compute the influences (local) of all the variables in the formula (Step 2). Over the nodes we execute the Shannon expansion, we also compute the influence for the variable over which using Step 7. Each node in the tree has an *influence vector* of size n, where n is the total number of variables in the input lineage. The values in the vector correspond to the *local* influences of the variables on the boolean formula corresponding to the node. This vector is recursively updated, based on the childrens' vectors in Step 6. Finally, the influence vector at the root of the tree has influences of all the variables in the formula.

We illustrate the above algorithm using an example.

EXAMPLE 1. *Consider the boolean formula given by* $= a_1b_1c_1 + a_1b_2c_2 + a_2b_3c_1 + a_3b_4c_1$. *It cannot be represented as a read-once formula. Hence, the algorithm first performs Shannon expansion around* c_1 *(since it appears* 3 *times) as shown below.*

$$= c_1(a_1b_1 + a_1b_2c_2 + a_2b_3 + a_3b_4) + \overline{c_1}(a_1b_2c_2) = c_1 \;_1 + \overline{c_1} \;_2$$

We can easily see that both $_2 = a_1b_2c_2$ *and* $_1 = a_1(b_1 + b_2c_2) + a_2b_3 + a_3b_4$ *are already in read-once format and no more expansion occurs. Now, the influence vectors at* $_1$ *and* $_2$ *are computed based on Algorithm 1. In addition, the influence of* c_1, $p(\ _1)$ $p(\ _2)$ *is computed. Following this, the influence vector at the parent node is updated. Note that since* a_1 *appears in both* $_1$ *and* $_2$ *nodes of the tree, its influence on is available at the influence vector of the root node.*

4.1.2 Conjunctive queries

Here, we consider arbitrary conjunctive queries which return multiple output results. As defined earlier (Section 2), the influence of an input variable is the sum of its influences on each of the output tuples. Note that even though the output tuples may be in read-once format, the set of output tuples may be correlated with each other, since the input tuples may be shared among the lineages of the output tuples [10]. Hence, we cannot use the naïve approach of looking at the top-ℓ set of influential tuples for each output tuple and use them to determine the overall top-ℓ. For example, suppose we are interested in determining the top-3 influential tuples for a set of correlated output tuples. A single input tuple might be influential for the output tuples combined, but it may not be enough to appear in the top-3 lists of influential tuples for each of the individual output tuples. However, summing up all the influences would be large enough for it to be in the top-3 list; since the naïve approach does not consider this tuple at all, it fails. Instead, we use a brute force approach where we sum up the influences of a given input tuple on each of the output tuples and pick the top-ℓ input tuples based on this value. We are currently working on developing more efficient algorithms for conjunctive queries based on extensional techniques.

4.1.3 Aggregation queries

Here, we determine the influence of each input variable on the expected value of the aggregate. We provide algorithms for determining the top-ℓ influential variables for MIN/MAX and AVG aggregates.

MIN/MAX: We only consider MAX here. The algorithm for MIN is very similar and we omit it here. We assume all a_is are positive and there is a dummy tuple with value 0 and probability 1 to avoid the empty possible world where MAX is undefined. We assume tuples are sorted in a non-increasing order of their scores. Recall x_i is the indicator variable of the existence of t_i. It is easy to see

that

$$MAX = E[\max_i x_i a_i] = \sum_i a_i p_i \prod_{j<i}(1 - p_j).$$

It is easy to see from the above formula that MAX is a linear function of p_i for any i. Recall $P_{p_i = a}$ is the same vector as P except the ith entry being replaced by a. By Lemma 1, we have

$$\frac{@MAX(P)}{@p_i} = \frac{MAX(P) - MAX(P_{p_i = 0})}{p_i}.$$

Now, we describe our algorithm. We first show how to compute MAX in linear time. Suppose we denote by max[i; j] the maximum of the random tuples $t_i, t_{i+1}, \ldots, t_j$. If we assume $a_i \geq a_{i+1} \geq \ldots \geq a_j$, then, max[i; j] = a_i with probability p_i and max[i; j] = max[i + 1; j] otherwise. Therefore, we have that

$$E[max[i; j]] = p_i a_i + (1 - p_i)E[max[i + 1; j]]. \quad (4)$$

When we compute MAX, we store all values E[max[i; n]] for all i. We can also easily compute $\prod_{j=1}^{i}(1 - p_j)$ values for all i in linear time. Now, we show how to quickly compute $MAX(P_{p_i = 0})$ for each i in constant time, provided we have already computed MAX, E[max[i; n]]$\forall$i and $\prod_{j=1}^{i}(1 - p_j)\forall i$. In fact, it is not hard to see that

$$\begin{aligned} MAX(P_{p_i = 0}) &= \sum_{j<i} a_j p_j \prod_{k<j}(1 - p_k) + \sum_{j>i} a_j p_j \prod_{k<j, k\neq i}(1 - p_k) \\ &= MAX \prod_j (1 - p_j) E[max[i; n]] \\ &\quad + \prod_{j<i}(1 - p_j) E[max[i + 1; n]] \end{aligned}$$

Therefore, the overall running time for finding the top-‘ influential tuples is O(n).

AVG: Now we consider the problem of computing top-‘ influential variables for AVG. Formally, AVG is defined to be

$$AVG = E\left[\frac{\sum_i a_i x_i}{1 + \sum_i x_i}\right].$$

Note that we have included a dummy tuple with value 0 and probability 1 to keep the denominator non-zero. The following theorem plays a central role for the streaming algorithm in [22] and is also crucial for computing the influence.

THEOREM 3. *([22]) For the probabilistic tuples* $t_1, \ldots, t_n$*, let* p_i *and* a_i *be the existence probability and the value of* t_i*, respectively. Define function* $h_{AVG}(x) = \sum_i a_i p_i x \prod_{j\neq i}(1 - p_j + p_j x)$*. Then,* $AVG = \int_0^1 h_{AVG}(x)dx$.

From the above theorem, it is easy to see that $h_{AVG}(x)$ is linear in p_i. Since the integral is over x, AVG is also linear in p_i. Thus, from Lemma 1, we have

$$\frac{@AVG(P)}{@p_i} = \frac{AVG(P) - AVG(P_{p_i = 0})}{p_i}.$$

It is known that computing AVG for a dataset of size n (in particular, expanding $h_{AVG}(x)$) can be done in $O(n \log^2 n)$ time [22]. Computing $AVG(P_{p_i = 0})$ once AVG is already computed additionally takes only linear time given the expansion of $h_{AVG}(x)$ (see Section 6 on recomputing query results). For each tuple t_i, we need to compute $\frac{\partial AVG}{\partial p_i}$. Therefore, the overall running time is $O(n^2)$.

4.2 Set queries

In this section, we discuss set-based queries and we develop techniques for computing -influential variables for these queries. We start with probabilistic threshold queries.

4.2.1 Probabilistic Threshold Queries

To evaluate a probabilistic threshold query PT(Q;), we first run the conjunctive query Q and subsequently select all output tuples with probability more than . Since the lineage formulas generated by conjunctive queries are monotone and positive [30], increasing the probability of an input tuple can only *increase* the probability of the output tuples, thereby increasing the number of output tuples; similarly, decreasing the probability of an input tuple might remove those output tuples whose probabilities become less than . We only consider the case when the input probabilities are increased. The symmetric case of decreasing the input probabilities is analogous and is not discussed here. Our objective is to compute the top-‘ -influential variables (Section 2). We rank an input tuple by its degree of -influence, i.e., the number of output tuples that will be added to the output set, if we increase its probability by (Definition 2).

We briefly discuss the naive algorithm before presenting techniques for improving it. In the first step, we compute the output tuple lineages and subsequently, the influences of each input tuple on each output tuple. In the second step, we go over each input tuple (t_i) and determine the number of output tuples (C_i) that would cross the threshold if we increase its probability by . Finally, we compute the top-‘ influential input tuples from this list. This technique is quite inefficient as the complexity of the first step is O(no), where o is the number of output tuples and n can potentially include all input tuples. We reduce the complexity of this operation by considering only those output tuples which contribute to the C_i values, by developing three pruning rules.

Rule 1: Ignore output tuples with probability > – increasing the probability of its input tuples will not change the output.

Rule 2: Restrict attention to output tuples with probability > . The influence of an input tuple over a single output tuple for conjunctive queries is always less than 1, since influence is defined as a difference between two probability values. Therefore, the probability of an output tuple can at most increase by and output tuples with probability values less than cannot get into the result set.

Next, we propose another pruning rule which works only for read-once lineages. Consider output tuple O_i. Suppose it has probability $o_i <$. The input tuples that can drive this probability to must have influence at least equal to = (o_i)= . So, we only need to increment C_j values of those input tuples whose influences exceed . We modify the *infl_read_once* routine of Section 4.1.1 to only return the input tuples that satisfy the above requirement by exploiting the following property:

THEOREM 4. *The derivatives of the nodes in an AND/OR tree monotonically decrease as we go down the AND/OR tree.*

PROOF. The proof is very easy to see using recursion. Recall the recursive equations used to update the probabilities of internal nodes in the AND/OR tree.

$$\text{if } z = x_1 \wedge x_2 \qquad \frac{@z}{@x_1} = x_2 < 1$$

$$\text{if } z = x_1 \vee x_2 \qquad \frac{@z}{@x_1} = (1 - x_2) < 1$$

Now, using Equation (3), we can see that the values of the derivatives decrease as we go down the tree since we continuously multi-

ply by a number less than 1 at each level. The derivative of the root with respect to itself is 1 by definition. □

Rule 3: In *infl_read_once*, if the value of the derivative is less than , we do not recurse along the branch. We only recurse along the portion of the tree whose derivative is more than . Since the derivatives are computed in a top-down manner, pruning via this rule can provide several benefits for large data sets.

4.2.2 Top-k queries by probability

The output of a top-k query is a list of output tuples sorted in decreasing order by their probabilities. Therefore, modifying input tuples can cause new tuples to enter the output while simultaneously removing the same number of existing output tuples. Suppose we treat the probability of the k^{th} output tuple (in order) as the threshold . As with probabilistic threshold queries, the only tuples that can enter the result are the set of output tuples with probabilities in the range [;]. Hence, we can apply the pruning rules 1 and 2 work here also. Hence the only input tuples which are influential are the ones that appear in the lineages of these tuples. The algorithm for computing influential variables here is similar to the one for probabilistic threshold queries. The difference between top-k and probabilistic threshold queries is that, for an input tuple t to force the output tuple O_i into the top-k output, it is not enough that its influence value exceed (o_i)= . The reasons are two fold. First, by increasing the probability of an input tuple, we may be increasing the threshold itself, i.e., if the input tuple appears in the lineage of the k^{th} ranked tuple. Second, once a new tuple enters the top-k, the value of needs to be increased. Hence, we have to explicitly check the number of new tuples entering the top-k for each input tuple and then compute the top-ϵ influential input tuples. We plan to develop more efficient algorithms for this purpose in future work.

4.3 How is ϵ assigned?

Until now, we assumed that was provided by the user. However, it is unlikely to expect the user to know the value of . Firstly, a user might know the margin of error that might be present in the input probabilities. For instance, if the application generating the input probability reports that there may be error in the probabilities, then the user can pick between [0;]. Another way to pick a reasonable value might be through a visualization tool. Given an input tuple t with probability p, the output probabilities are linear in p and can be visualized as straight lines. The user can now pick using this visualization, e.g., a region with several intersections.

5. EXPLANATION ANALYSIS

In this section, we provide algorithms for computing the top-ϵ explanations for value queries.

5.1 Boolean Conjunctive Queries

Recall that computing explanations requires us to determine the set of ϵ input tuples, whose probabilities when set to 0, causes the maximum decrease in the output probabilities.

THEOREM 5. *The problem of computing the top-ϵ explanations for boolean conjunctive queries (even without self-joins) is NP-hard.*

PROOF. We use a reduction from vertex cover on 3-uniform 3-partite hypergraph [34] similar to Theorem 4.1 of Meliou et al. [27]. Consider a 3-uniform 3-partite graph G with partitions R, S and T. Every hyperedge contains exactly one vertex from each partition. We construct a database D with 4 relations $R(x)$, $S(y)$, $T(z)$ and $U(x;y;z)$. For each vertex in $R;S$ and T, we introduce a tuple (with tuple-uncertainty) in $R(x)$, $S(y)$ and $T(z)$ respectively. For each hyperedge in G, we introduce a new deterministic tuple in $U(x;y;z)$. Consider the boolean conjunctive query:

$$q() : \quad R(x);S(y);T(z);U(x;y;z)$$

We can easily see there is a vertex cover of size at most ϵ in G, if and only if we can determine an explanation of size ϵ that reduces the probability of the output tuple to zero (maximum possible reduction) in the probability.
□

Although the problem is hard in general, for queries that lead to read-once lineage formulas, there exists an optimal algorithm. We describe this algorithm next.

Read-once formulas

For the case of read-once functions, we use a dynamic programming algorithm to compute the explanation. Consider a lineage formula with two subtrees $_l$ and $_r$. Suppose the function $OPT(;k)$ represents the smallest possible value for probability of by setting k input probabilities to 0 (i.e., maximum possible reduction in the probability). The appropriate recursion for the dynamic program is shown below.

If $= _l \wedge _r$,

$$OPT(;k) = \min_{k_1}(OPT(_l;k_1) \quad OPT(_r;k \quad k_1))$$

If $= _l _ _r$,

$$OPT(;k) = \min_{k_1} \begin{bmatrix} OPT(_l;k_1) + OPT(_r;k \quad k_1)) \\ OPT(_l;k_1) \quad OPT(_r;k \quad k_1) \end{bmatrix}$$

We modify the above program to also include the top-ϵ input tuples at each step. The proof of correctness of the above program can be seen via contradiction. Assume, for the sake of contradiction that there exists a different solution S^0 (set of ϵ variables) which is better than the solution obtained by the algorithm S. We consider two cases. Suppose the root node is an AND node with two children $_l$ and $_r$. Also suppose S^0 has l_1 variables on the left child and l l_1 variables on the right child, which are different from S. Denote $Pr(;S)$ the probability of the boolean formula by setting the probabilities of variables in S to 0. According to our assumption $Pr(;S^0) < Pr(;S)$. This means that $Pr(_l;S^0_l)Pr(_r;S^0_r) < Pr(_l;S_l)Pr(_r;S_r)$. However, this statement is false since the sets S_l and S_r were chosen such that their product was minimum (according to our update rule).

The complexity of the above program is $O(n\epsilon^2)$ since at each step we spend time $O(\epsilon)$ to determine $OPT(;\epsilon)$ and we need to compute OPT for different values from $\{1;2;\ldots;\epsilon\}$.

Non-read-once formulas

We propose two greedy heuristics for this problem.

First Approach: For each input tuple t_i, we compute its influence $_i$ for the output tuple and sort the tuples by the value of $_i p_i$. We select the top-ϵ tuples as the best explanation for our purpose. The motivation for this heuristic is that the product of the influence and the probability corresponds to the "individual contribution" made by the tuple (to a first approximation). Note that this ignores the pairwise and higher order contributions.

Second Approach: Now, we propose a slightly better approximation. In the first step, we select the tuple with the highest contribution, i.e., $_i p_i$ value. Next, we set this tuple probability to zero and recompute the contributions for the remaining tuples. We pick the tuple with the highest contribution and repeat the process ϵ times.

query result, as defined in Meliou et al. [27] relates to the number of possible worlds that are affected by the presence/absence of the tuple. We describe the connection between responsibility as defined here and influence is Section 3.4.

Provenance has been widely studied in the database literature [2, 4, 17, 35] and several models of provenance have been proposed based on boolean formulas (lineage), semirings and so on. In probabilistic databases, the lineage formulas are very large and it is therefore difficult to understand the interesting tuples for a query. Further, the input tuples may have different probabilities and a boolean formula by itself does not capture all the properties of the input tuples. Hence, in our work, we build on top of the lineage provenance by extracting two useful entities, i.e., the most influential/sensitive input tuples and the best explanations from the lineage formula and the associated input tuple probabilities. Re et al. [30] discuss techniques for identifying the most influential tuples for a query by using as input, an approximate lineage formula. We discuss its relationship to our work in Section 1.

Van der Gaag et al. [25] prove that sensitivity of an output variable in a Bayesian network with respect to each input CPT parameter can be computed in $O(n)$ time (n is the size of the network) using techniques similar to junction tree belief propagation. The number of CPT parameters is typically exponential in the size of the largest factor in the Bayesian network. Chan et al. [3] have continued this line of work to compute influences of *pairs* of input parameters. These analyses are more general than our methods since we make assumptions regarding (1) tree structured nature of the Bayesian network , (2) boolean input variables, implying much fewer input parameter and, (3) we are only interested in measuring the sensitivity of one designated child-less output node. Owing to the simplified nature of our problem, we have been able to provide more efficient algorithms in our case.

9. CONCLUSIONS

In recent years, there has been a surge of interest in developing systems for managing and querying uncertain data. While significant advances have been made so far in this effort, most of the systems developed assume query processing over probabilistic database queries as a one-shot process. However, probabilistic databases need to be designed as an interactive application in which users have flexibility to identify relevant input probabilities for a given query and re-evaluate the query with the new values for the probabilities of these tuples. In this paper, we take a first step in this direction by extending a probabilistic database system to support *sensitivity analysis* and *explanation* analysis. Explanations help users understand the cause of an answer while sensitivity analysis considers the stability of the output probabilities, thereby helping users to focus their attention on input tuples that might significantly alter the output. Providing such functionality enables a robust framework for query evaluation in probabilistic databases.

Acknowledgement

This work was supported by NSF Grants IIS-0546136 and IIS-0916736.

10. REFERENCES

[1] L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and simple relational processing of uncertain data. In *ICDE*, 2008.

[2] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *ICDT*, pages 316–330, 2001.

[3] H. Chan and A. Darwiche. Sensitivity analysis in Markov networks. In *IJCAI*, pages 1300–1305, 2005.

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

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

[6] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD*, 2003.

[7] L. Chiticariu, R. Krishnamurthy, Y. Li, S. Raghavan, F. Reiss, and S. Vaithyanathan. Systemt: An algebraic approach to declarative information extraction. In *ACL*, 2010.

[8] D. G. Corneil, Y. Perl, and L. K. Stewart. A linear recognition algorithm for cographs. *SIAM J. Comput.*, 14(4), 1985.

[9] P. Dagum and M. Luby. Approximating probabilistic inference in Bayesian belief networks is np-hard. *Artif. Intell.*, 60(1), 1993.

[10] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, 2004.

[11] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications (Third Edition)*. Springer-Verlag, 2008.

[12] A. Doan, R. Ramakrishnan, F. C. 0002, P. DeRose, Y. Lee, R. McCann, M. Sayyadian, and W. Shen. Community information management. *IEEE Data Eng. Bull.*, 29(1), 2006.

[13] X. L. Dong, A. Y. Halevy, and C. Yu. Data integration with uncertainty. In *VLDB*, 2007.

[14] A. Ferrara, G. Pan, and M. Y. Vardi. Treewidth in verification: Local vs. global. In *LPAR*, 2005.

[15] H. Garcia-Molina. Entity resolution: Overview and challenges. In *ER*, pages 1–2, 2004.

[16] M. C. Golumbic, A. Mintz, and U. Rotics. Factoring and recognition of read-once functions using cographs and normality. In *DAC*, 2001.

[17] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.

[18] R. Gupta and S. Sarawagi. Creating probabilistic databases from information extraction models. In *VLDB*, 2006.

[19] J. Y. Halpern and J. Pearl. Causes and explanations: A structural-model approach - part ii: Explanations. In *IJCAI*, 2001.

[20] J. Y. Halpern and J. Pearl. Causes and explanations: A structural-model approach: Part 1: Causes. In *UAI*, 2001.

[21] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. M. Jermaine, and P. J. Haas. Mcdb: a monte carlo approach to managing uncertain data. In *SIGMOD*, 2008.

[22] T. S. Jayram, S. Kale, and E. Vee. Efficient aggregation algorithms for probabilistic data. In *SODA*, pages 346–355, 2007.

[23] T. S. Jayram, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu. Avatar information extraction system. *IEEE Data Eng. Bull.*, 2006.

[24] B. Kanagal and A. Deshpande. Lineage processing over correlated probabilistic databases. In *SIGMOD*, 2010.

[25] U. Kjærulff and L. C. van der Gaag. Making sensitivity analysis computationally efficient. In *UAI*, 2000.

[26] A. Krause and C. Guestrin. Optimal nonmyopic value of information in graphical models - efficient algorithms and theoretical limits. In *IJCAI*, 2005.

[27] A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu. The complexity of causality and responsibility for query answers and non-answers. In *PVLDB*, 2011.

[28] D. Olteanu, J. Huang, and C. Koch. Approximate confidence computation in probabilistic databases. In *ICDE*, 2010.

[29] C. Re, N. N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, pages 886–895, 2007.

[30] C. Ré and D. Suciu. Approximate lineage for probabilistic databases. *PVLDB*, 2008.

[31] A. D. Sarma, O. Benjelloun, A. Y. Halevy, and J. Widom. Working models for uncertain data. In *ICDE*, 2006.

[32] P. Sen, A. Deshpande, and L. Getoor. Prdb: managing and exploiting rich correlations in probabilistic databases. *VLDB J.*, 18(5):1065–1090, 2009.

[33] P. Sen, A. Deshpande, and L. Getoor. Read-once functions and query evaluation in probabilistic databases. In *PVLDB*, 2010.

[34] P. Senellart and G. Gottlob. On the complexity of deriving schema mappings from database instances. In *PODS*, pages 23–32, 2008.

[35] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, 2005.