Consistent Query Answers in Inconsistent Probabilistic Databases

Xiang Lian, Lei Chen, and Shaoxu Song
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
Hong Kong, China
{xlian, leichen, sshaoxu}@cse.ust.hk

ABSTRACT

Efficient and effective manipulation of probabilistic data has become increasingly important recently due to many real applications that involve the data uncertainty. This is especially crucial when probabilistic data collected from different sources disagree with each other and incur inconsistencies. In order to accommodate such inconsistencies and enable consistent query answering (CQA), in this paper, we propose the *all-possible-repairs* semantics in the context of inconsistent probabilistic databases, which formalize the repairs on the database as *repair worlds* via a graph representation. In turn, the CQA problem can be converted into one in the so-called *repaired possible worlds* (w.r.t. both repair worlds and possible worlds). We investigate a series of consistent queries in inconsistent probabilistic databases, including consistent range queries, join, and top-k queries, which, however, need to deal with an exponential number of the repaired possible worlds at high cost. To tackle the efficiency problem of CQA, in this paper, we propose efficient approaches for retrieving consistent query answers, including effective pruning methods to filter out false positives. Extensive experiments have been conducted to demonstrate the efficiency and effectiveness of our approaches.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query processing*; H.3.3 [**Information Systems**]: Information Storage and retrieval—*Information search and retrieval, Search process*

General Terms

Algorithms, Design, Experimentation, Performance, Theory

Keywords

Inconsistent probabilistic database, consistent query answering, all-possible-repairs semantics, repair world, repaired possible world

1. INTRODUCTION

Recently, a variety of data management applications require data integration [30, 17, 11] from multiple sources, for example, the crawled Web data from Internet, news or rumors from people's personal blogs, and data exchanged or bought from corporations. Due to various reasons such as data expiration, input typos, or subjective comments made by people, data from these sources often exhibit inconsistencies or conflicts. Thus, it has become an increasingly critical yet challenging task for the data integration system to resolve conflicts and present consistent (clean) query answers to users.

Table 1 shows an example of an inconsistent database, which contains records of two restaurants t_1 and t_2 obtained from different sources on the Internet (e.g., the restaurant homepage, comments from customers, or online surveys), with the table schema:

$$\mathsf{restaurant}(\underline{\mathsf{RID}, \mathsf{SID}}, \mathsf{AC}, \mathsf{Zip}, \mathsf{Loc.}, \mathsf{Status}, \mathsf{QoS}),$$

where each restaurant tuple has attributes of restaurant id RID, source id SID, area code AC, zip code Zip, relative location in the area Loc., "Open/Closed" status of the restaurant Status, and the rating of customers about the quality of services QoS.

RID	SID	AC	Zip	Loc.	Status	QoS	$t_{jr}.p$
t_1	t_{11}	44	20000	A	Open	5	.2
t_1	t_{12}	44	20000	B	Closed	4	.4
t_1	t_{13}	44	10000	B	Closed	3	.1
t_2	t_{21}	44	10000	A	Closed	3	.8

Table 1: An inconsistent (probabilistic) database (restaurant).

In the table, the first three tuples correspond to records of the same restaurant t_1, obtained from sources t_{11}, t_{12}, and t_{13}. However, we can see that there are inconsistencies among them. For example, assuming $A \neq B$, tuples t_{11} and t_{12} have completely different locations Loc. ("A" and "B", respectively); tuples t_{12} and t_{13} have distinct zip codes Zip (i.e., "20000" and "10000", respectively); and tuples t_{11} and t_{13} disagree on both Zip and Loc.. Thus, they all violate our common knowledge that the same restaurant can have only one address (i.e., Zip and Loc.).

To incorporate such inconsistent information from multiple sources, probabilistic databases usually associate each tuple with an existence probability, indicating the (relative) confidence of its source compared with others [15]. As an example, to transform Table 1 to a probabilistic database, one can add a new column $t_{jr}.p \in [0, 1]$ for each tuple t_{jr}, which implies the percentage of customers /comments that agree on this tuple. As a result, although different sources may provide inconsistent tuples (or called *alternatives*[1]) such as t_{11}, t_{12}, and t_{13} for the same restaurant t_1 (called *x-tuple*), we can easily interpret them as assertions from sources t_{11}, t_{12}, and t_{13} believed to be true by people with probabilities 0.2 ($= t_{11}.p$), 0.4 ($= t_{12}.p$), and 0.1 ($= t_{13}.p$), respectively; similarly, the assertion from the single alternative t_{21} for restaurant t_2 is believed to be true with probability 0.8 ($= t_{21}.p$).

The manipulation of such probabilistic data usually considers the *possible worlds* semantics [15]. Specifically, a possible world is a

[1]Throughout this paper, we will use terms *tuple* and *alternative* in a probabilistic database interchangeably with the same meaning.

SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA.

materialized instance of the probabilistic database that can appear in the real world, and in each possible world at most one alternative per x-tuple can appear. For example, tuple set $\{t_{11}, t_{21}\}$ forms a possible world, to which each of the two x-tuples t_1 and t_2 contribute one alternative t_{11} and t_{21}, respectively.

Although probabilistic databases can accommodate conflicts by assigning probabilities to different sources for the same data entity (x-tuple), the inconsistency may still exist among different x-tuples. For example, the two tuples in the possible world, $\{t_{11}, t_{21}\}$, of Table 1 come from two distinct restaurants (x-tuples) t_1 and t_2, respectively. However, they are inconsistent with each other, that is, they have the same area code AC (="44") and restaurant location Loc. (="A"), but different zip codes Zip (i.e., "20000" and "10000", respectively), which violates a functional dependency:

$$[\mathsf{AC}, \mathsf{Loc.}] \rightarrow [\mathsf{Zip}].$$

Therefore, it indicates that these probabilistic data can still contain errors or exceptions [20], which might lead to erroneous query answers. Inspired by this, in this paper, we investigate how to resolve the inconsistency among x-tuples and obtain consistent query answers on them, which, to our best knowledge, no prior work has studied in the context of inconsistent probabilistic database.

SID	$\mathcal{D}_1^R$	$\mathcal{D}_2^R$	$\mathcal{D}_3^R$	$\mathcal{D}_4^R$	$\mathcal{D}_5^R$	$\mathcal{D}_6^R$	$\mathcal{D}_7^R$
t_{11}	×	×	√	×	×	×	×
t_{12}	×	√	×	×	√	×	×
t_{13}	√	×	×	√	×	×	×
t_{21}	√	√	×	×	×	√	×

Table 2: Minimal repairs $\mathcal{D}_1^R$ and $\mathcal{D}_2^R$ out of all possible repairs in Table 1.

AC	Zip	Loc.
44	20000	A
44	30000	B

Table 3: One possible ground truth in reality.

Previous works in inconsistent (certain) databases [4] proposed to resolve the inconsistency by repairing the raw database. That is, either data tuples are removed from (inserted into) the database [12, 4], or attribute values of tuples are modified [8, 38, 13], such that a consistent (certain) database is obtained. However, they typically consider *minimal repairs* of the database [4]. That is, the repaired database should have the fewest tuple differences from the original one (or with the smallest repair cost). Taking repairs via tuple deletion [12] as an example, Table 2 shows all the 7 possible repairs, $\mathcal{D}_1^R \sim \mathcal{D}_7^R$, to resolve the inconsistency of the database in Table 1, where "×" represents the tuple deletion, and "√" indicates that tuple is retained in the repaired database. The first two repairs (columns), $\mathcal{D}_1^R$ and $\mathcal{D}_2^R$, correspond to minimal repairs, which delete either tuple set $\{t_{11}, t_{12}\}$ or $\{t_{11}, t_{13}\}$ (i.e., the conflicts cannot be resolved by deleting fewer than 2 tuples). As a result, tuple t_{11} is always deleted under minimal repairs. While one possible ground truth in reality might be given by Table 3, indicating that tuple t_{11} is true with AC = "44", Zip = "20000", and Loc. = "A", the minimal repairs of Table 1 in this example will never restore the original (clean) database (containing t_{11}).

In fact, the main reason for this is due to an implicit assumption made by minimal repairs, that is, true data are provided by much more sources than false ones in the database. Unfortunately, such an assumption does not always hold in many scenarios [16]. In practice, rumors can be copied and propagated on the Internet very fast, which might dominate the truth. Similarly, a newly updated official website might be more reliable (assuming to contain the truth), however, its new contents may not appear in caches of the search engine (database) so fast, due to the delay of the Web crawling. As investigated in a study [16] about restaurants in Manhattan from 1/22/2009 to 3/12/2009, among 467 restaurants that are explicitly marked as "Closed" in the crawled data, only 280 of them were indeed closed, which indicates that data of the rest restaurants were stale and out-of-date. Therefore, in such situations, a minimal repair of an inconsistent database might rule out truths (e.g., t_{11} in Table 1) while keeping rumors, and lead to incorrect inference or query answers [16].

Thus, in this paper, instead of solely considering minimal repairs, we take into account their superset containing all possible repairs (weighted by their repair confidences) in an inconsistent probabilistic database, which have at least one repair that is able to correctly restore a truth database. As shown in Table 2, among all possible repairs (i.e., $\mathcal{D}_1^R \sim \mathcal{D}_7^R$), $\mathcal{D}_3^R$ exactly contains tuple t_{11} which reflects the truth in Table 3. Thus, consider a query that obtains restaurants with attribute QoS within $[4, 5]$. For minimal repairs $\mathcal{D}_1^R$ and $\mathcal{D}_2^R$, we can only obtain either empty (i.e., $\emptyset$) or wrong answer (i.e., $\{t_{12}\}$), respectively. In contrast, by considering all possible repairs (including $\mathcal{D}_3^R$), we are able to report the (true) answer t_{11} associated with some confidence.

Therefore, in the sequel, we will consider a series of important and useful consistent query types under the *all-possible-repairs* semantics in inconsistent probabilistic databases, including *probabilistic consistent range query* (PC-Range), *probabilistic consistent join* (PC-Join), and *probabilistic consistent top-k query* (PC-Top$k$). As in the example of Table 1, PC-Range might report restaurant tuples with the quality of services QoS $\in [4, 5]$, and their *consistent scores* above a threshold, where a consistent score indicates the confidence that a tuple satisfies the query predicates w.r.t. all possible repairs. Similarly, PC-Join computes the joining tuples from two inconsistent probabilistic databases, whose consistent scores are above a threshold. Furthermore, PC-Top$k$ returns k tuples with the highest consistent scores.

Consistent query answering (CQA) in probabilistic databases has many practical applications. GreenOrbs [28] is a project that realizes all-year ecological surveillance in the forest, collecting various sensory data including temperature, humidity, illumination, and carbon dioxide titer. These data can be integrated with other data such as animals' positions (via RFID) to study their habitats and behaviors. Due to environmental factors and inaccuracies of sensing devices, the collected sensor/RFID data are often noisy and unreliable. We can thus consider such integrated data (i.e., environmental parameters around animals) as an inconsistent probabilistic database, where the tuple uncertainty results from uncertain joining attributes (i.e., animal positions), and conduct CQA to obtain consistent query answers with high confidence.

Note that, previous works studied CQA in inconsistent certain databases [4, 20], which returns the intersection of query answer sets on different minimal repairs. In contrast, CQA in an inconsistent probabilistic database considers different repair semantics (i.e., all-possible-repairs rather than minimal repairs), and reports consistent answers from all possible repairs with certain confidences. This is more challenging, since CQA has to consider the consistent confidence, conducted in an exponential number of possible worlds on a possibly exponential number of the repaired databases. Thus, the previously proposed techniques in certain databases cannot be directly applied to our problem. In addition, there are also many studies on queries in probabilistic databases [15, 31, 34, 40, 23, 14, 27]. Nonetheless, they always assume that databases are consistent, and thus cannot provide consistent answers over inconsistent data.

In this paper, we make the following contributions.

1. We formalize the CQA problem in an inconsistent probabilistic database, including a series of queries such as PC-Range, PC-Join, and PC-Top$k$ queries in Section 3.
2. We provide a novel notion of *repair worlds* (i.e., all-possible-repairs) using a graph representation to model the repairs of

an inconsistent probabilistic database, and reduce the CQA problem to the one under the *repaired possible worlds* (w.r.t. possible worlds and repair worlds) in Sections 4.1 and 4.2.

3. We derive a recursive function for computing consistent scores of tuples via the graph in Sections 4.3 and 4.4. Further, we present efficient query processing approaches for obtaining PC-Range, PC-Join, and PC-Top$k$ answers in Section 5.

4. We conduct extensive experiments to confirm the efficiency and effectiveness of our proposed approaches in Section 6.

In addition, Section 2 reviews previous works on consistent query answering in certain databases, as well as query answering in probabilistic databases. Finally, Section 7 concludes this paper.

2. RELATED WORK

2.1 CQA in Certain Database

In inconsistent certain databases that contain precise data [4, 20], inconsistent tuples usually violate some integrity constraints such as key constraints, functional dependencies, and so on. Many previous studies focused on how to manipulate (tuples in) the database to a consistent one with the minimum *repair cost*. The proposed repair models [18] include X-repair [12] allowing tuple deletions only, S-repair [4] with both tuple insertions and deletions, and U-repair [8, 38, 13] considering tuple value updates. In this paper, we adopt X-repair semantics (i.e., tuple deletions only) to probabilistic databases, and we would like to leave consistent query answering (CQA) under other repair semantics as our future work.

Arenas et al. [4] proposed the notion of consistent query answers over the repaired database, which is the intersection of query answer sets on all minimal repairs of the database, where a minimal repair is a repair such that the symmetric difference between the repaired and original databases (i.e., $(\mathcal{D} - \mathcal{D}^R) \cup (\mathcal{D}^R - \mathcal{D})$) is the minimum. Fuxman et al. [20] presented a system, called ConQuer, which uses SQL query rewriting as the CQA strategy, and answers consistent relational queries (e.g., selection-projection-join) on databases that violate the key constraint. Andritsos et al. [2] proposed a probabilistic approach to clean answers on a dirty database with tuples violating the key constraint. Each tuple in the dirty database is associated with a probability and tuples with the same key value have probabilities summed up to 1. The clean answer is defined as tuples that appear in at least one candidate (minimally repaired) database with nonzero probabilities. Wijsen [38] considered the database repair by updating attribute values. The CQA with this repair semantics is NP-hard, and thus becomes intractable. Therefore, only a few query classes that allow nuclei of polynomial size can be computed in polynomial time. Greco and Molinaro [21] proposed probabilistic CQA by returning approximate answers associated with the percentages that answers appear in the repaired databases, where the minimal repair with value updates is considered. Beskales et al. [7] studied the problem of eliminating duplicates in an inconsistent certain database, which models repairs w.r.t. parameter settings of clustering algorithms, and can answer relational operations such as selection, projection, and join, as well as that permits probabilistic constraints.

The aforementioned studies assume that the underlying database contains certain and precise data. To our best knowledge, no previous work has studied the CQA problem in probabilistic databases. Previous techniques (e.g., SQL query rewriting on precise relational data) cannot be directly applicable to our CQA problem. This is because, CQA in probabilistic databases has to additionally consider the possible worlds semantics, which is more challenging due to the exponential size of possible worlds.

Possible World $pw(\mathcal{D})$	**Probability** $Pr\{pw(\mathcal{D})\}$
$pw_1(\mathcal{D}) = \{t_{11}\}$	$.2 \times (1 - .8) = .04$
$pw_2(\mathcal{D}) = \{t_{12}\}$	$.4 \times (1 - .8) = .08$
$pw_3(\mathcal{D}) = \{t_{13}\}$	$.1 \times (1 - .8) = .02$
$pw_4(\mathcal{D}) = \{t_{11}, t_{21}\}$	$.2 \times .8 = .16$
$pw_5(\mathcal{D}) = \{t_{12}, t_{21}\}$	$.4 \times .8 = .32$
$pw_6(\mathcal{D}) = \{t_{13}, t_{21}\}$	$.1 \times .8 = .08$
$pw_7(\mathcal{D}) = \{t_{21}\}$	$(1 - .2 - .4 - .1) \times .8 = .24$
$pw_8(\mathcal{D}) = \emptyset$	$(1 - .2 - .4 - .1) \times (1 - .8) = .06$

Table 4: Possible worlds of restaurant **in Table 1.**

Furthermore, previous works usually define CQA on minimal repairs, which, as mentioned in Section 1, do not guarantee removing all errors and obtaining correct query answers. In contrast, our work considers all-possible-repairs semantics, which cannot directly use CQA techniques with minimal repairs.

2.2 Query Answering in Probabilistic Databases

Probabilistic databases [15] can represent inconsistent or uncertain data in many real applications such as sensor data analysis [1, 23], data integration [30, 17, 11], and data cleaning [22]. Many systems such as MystiQ [9], Orion [10], TRIO [6, 33], MayBMS [3], MCDB [25], and BayesStore [37] have been proposed to manipulate uncertain data.

Query answering in probabilistic databases usually considers the possible worlds semantics [15]. Query processing in probabilistic databases is equivalent to the procedure of first retrieving query answers in all possible worlds (using any query processing techniques on certain data), and then aggregating query results obtained from possible worlds as the final answer. Under different aggregation semantics, the resulting query answer sets differ from each other. For example, top-k queries have been extensively studied in (consistent) probabilistic databases with different aggregation semantics [31, 34, 40, 23, 14, 27]. Recently, Li et al. [27] gave a parameterized ranking definition which can incorporate many top-k query semantics such as U-kRank [34, 40], PT-k [23], and Expected-Rank [14], and also proposed two additional top-k semantics, namely, PRF^w and PRF^e.

The repair-key operation in MayBMS [3] repairs certain table with the violation of key constraints by removing a minimum subset of tuples. Most previous works usually assume queries in a consistent probabilistic database. In contrast, our work focuses on that in an inconsistent one, which involves exponential numbers of possible repairs (possible worlds as well). Thus, it is inefficient (or even infeasible) to directly borrow the existing approaches (proposed for consistent probabilistic database) and apply them on each of the repaired databases (with exponential numbers). This inspires us to carefully design efficient approaches for CQA below.

3. PROBLEM DEFINITION

3.1 Notations

Probabilistic Databases [15]: A probabilistic database $\mathcal{D}$ consists of N x-tuples $t_1, t_2, ...,$ and t_N, where each x-tuple t_j contains one or multiple alternatives (tuples) t_{jr} ($1 \leq r \leq l_j$). Each alternative t_{jr} of an x-tuple $t_j \in \mathcal{D}$ has an existence probability $t_{jr}.p \in [0, 1]$ belonging to the database, satisfying $\sum_{r=1}^{l_j} t_{jr}.p \leq 1$. Any two alternatives in an x-tuple are *mutually exclusive* (i.e., they cannot belong to the database at the same time).

Possible Worlds: Since each alternative t_{jr} can either exist or not exist in a probabilistic database $\mathcal{D}$, in the real world, $\mathcal{D}$ can have several possible materialized instances, called *possible worlds* (denoted as $pw(\mathcal{D})$). In each possible world, at most one alternative appears for any x-tuple.

DEFINITION 3.1. (*Possible Worlds, $pw(\mathcal{D})$*) *A possible world, $pw(\mathcal{D})$, is a subset of probabilistic database $\mathcal{D}$ such that each x-*

tuple $t_j \in \mathcal{D}$ has either zero or only one alternative t_{jr} in $pw(\mathcal{D})$. The appearance probability, $Pr\{pw(\mathcal{D})\}$, that a possible world $pw(\mathcal{D})$ occurs is given by:

$$Pr\{pw(\mathcal{D})\} = \prod_{\forall t_{jr} \in pw(\mathcal{D})} t_{jr}.p \cdot \prod_{\forall t_j \notin pw(\mathcal{D})} (1 - \sum_{r=1}^{l_j} t_{jr}.p),$$

where l_j is the number of alternatives in x-tuple t_j.

Table 4 shows 8 possible worlds $pw(\mathcal{D})$ and their appearance probabilities $Pr\{pw(\mathcal{D})\}$ in the probabilistic database of Table 1.

Repairs on Inconsistent Probabilistic Databases: As mentioned in Section 1, rather than minimal repairs, in the context of inconsistent probabilistic databases, we will consider all possible repairs of the database. Similar to X-repair [12], we repair the database by deleting some possibly erroneous (inconsistent) tuples in $\mathcal{D}$.

Specifically, there are three possible causes of an inconsistency between any two tuples, say t_{11} and t_{21} in the previous example of Table 1, that is, (1) the erroneous input of t_{11}, (2) that of t_{21}, or (3) that of both t_{11} and t_{21}. Thus, our repairs can delete: (1) t_{11}, (2) t_{21}, or (3) both t_{11} and t_{21}, respectively, in order to purge the inconsistency. Formally, we say a repaired probabilistic database, $\mathcal{D}^R$, is the one after removing some inconsistent tuples from $\mathcal{D}$, such that the remaining tuples in $\mathcal{D}^R$ become consistent.

Following the practice of US national statistical agencies [19, 39], we assume that each tuple $t_{jr} \in \mathcal{D}$ is associated with a *repair confidence*, $t_{jr}.rp \in [0, 1]$, which reflects the confidence that t_{jr} might contain errors and should be repaired (deleted). Such a repair confidence can be usually obtained from statistics of historical data. For example, in sensor applications like GreenOrbs project [28], this repair confidence can be defined as the probability that a sensor reports abnormal (erroneous) data in history. Moreover, similar to [32], we assume that each tuple is integrated from one or multiple independent sources (e.g., a source can be an independent sensor in sensor networks), and any two tuples do not share common sources (e.g., sensors). As a result, the repairs (deletions) on these tuples are independent. In the case where repair confidences of tuples are correlated, one can utilize the graphical model [37, 27] to describe the joint distribution of their repair confidences. Nonetheless, we would like to leave this case as our future work.

Accordingly, each repaired probabilistic database $\mathcal{D}^R$ is associated with a repair weight $W_r(\mathcal{D}^R)$, which is the confidence that we repair the database. In particular, let $\Delta(\mathcal{D}^R)$ be the set of deleted tuples for a repair from $\mathcal{D}$ (i.e., $\Delta(\mathcal{D}^R) = \mathcal{D} - \mathcal{D}^R$). Then, the repair weight $W_r(\mathcal{D}^R)$ can be given by:

$$W_r(\mathcal{D}^R) \quad (1)$$
$$= \prod_{\forall t_j \in \mathcal{D}} \left(\sum_{\forall t_{jr} \in \Delta(\mathcal{D}^R)} t_{jr}.rp \right) \cdot \prod_{\forall t_j \in \mathcal{D}} \left(1 - \sum_{\forall t_{jr} \notin \Delta(\mathcal{D}^R)} t_{jr}.rp \right).$$

As in the restaurant example (Table 1), we can add another attribute, the repair confidence $t_{jr}.rp$, for each tuple t_{jr}, indicating the unreliability of this restaurant record t_{jr}. When $t_{11}.rp = t_{12}.rp = 0.8$ and $t_{13}.rp = t_{21}.rp = 0.1$, the repair weight $W_r(\mathcal{D}_1^R)$ of the repaired database $\mathcal{D}_1^R$ (in Table 2) is given by $0.8 \times 0.8 \times (1 - 0.1) \times (1 - 0.1) = 0.5184$.

3.2 CQA in Inconsistent Probabilistic Databases

Next, we formalize the problem of consistent query answering (CQA) in an inconsistent probabilistic database, by considering 3 important query types, PC-Range, PC-Join, and PC-Top$k$. In brief, any tuple is a consistent answer to a query Q, if and only if it satisfies both *query predicates* $\mathcal{P}_Q$ and *consistent score predicates* $\mathcal{P}_S$. Here, a query predicate $\mathcal{P}_Q$ can be a user-specified querying region for PC-Range, a similarity join predicate for PC-Join, or a ranking predicate for PC-Top$k$; a consistent score predicate $\mathcal{P}_S$ requires consistent scores w.r.t. all possible repairs satisfying some constraint. We will illustrate $\mathcal{P}_S$ in detail for each query type below.

PC-Range Queries: We define PC-Range queries as obtaining tuples t_{jr} within a user-specified query region (i.e., range predicates $\mathcal{P}_R$) and having consistent scores greater than a threshold $\alpha_R \in [0, 1)$ (i.e., $\mathcal{P}_R : score_R(t_{jr}) > \alpha_R$) considering all possible repairs. Formally, we have:

DEFINITION 3.2. (*Consistent Range Query in Inconsistent Probabilistic Databases, PC-Range*) *Given an inconsistent probabilistic database $\mathcal{D}$, a range predicate $\mathcal{P}_R$, and a consistent score predicate $\mathcal{P}_S : \alpha_R \in [0, 1)$, a probabilistic consistent range query (PC-Range) retrieves all the tuples, t_{jr}, satisfying $\mathcal{P}_R$ and having consistent scores, $score_R(t_{jr})$, greater than threshold α_R, that is,*

$$score_R(t_{jr}) = t_{jr}.p \cdot \sum_{\forall t_{jr} \in \mathcal{D}^R} W_r(\mathcal{D}^R) > \alpha_R, \quad (2)$$

over all possible repairs, $\mathcal{D}^R$, where $W_r(\mathcal{D}^R)$ is given by Eq. (1).

Intuitively, the consistent score, $score_R(t_{jr})$, in Eq. (2) is the probability that tuple t_{jr} appears in possible worlds and it is not deleted by repairs. Larger consistent score indicates higher confidence that t_{jr} is in the query region in the (consistent) real world.

PC-Join Queries: Assume that we have two distinct inconsistent probabilistic databases, $\mathcal{D}_A$ and $\mathcal{D}_B$, with a common (join) attribute set $\mathcal{A}$. A PC-Join query retrieves pairs of alternatives t_A and t_B from $\mathcal{D}_A$ and $\mathcal{D}_B$, respectively, such that they satisfy a join predicate $\mathcal{P}_J$ on attributes in $\mathcal{A}$ and have consistent scores greater than a threshold α_J (i.e., $\mathcal{P}_S : score_J(t_A \bowtie_{\mathcal{P}_J} t_B) > \alpha_J$), over all their repair combinations.

DEFINITION 3.3. (*Consistent Join Query in Inconsistent Probabilistic Databases, PC-Join*) *Given two inconsistent probabilistic databases $\mathcal{D}_A$ and $\mathcal{D}_B$, a join predicate $\mathcal{P}_J : \varepsilon$, and a consistent score predicate $\mathcal{P}_S : \alpha_J \in [0, 1)$, a probabilistic consistent join query (PC-Join) retrieves all the joining tuple pairs $\langle t_A, t_B \rangle$ satisfying $t_A \bowtie_{\mathcal{P}_J} t_B$ (i.e., $dist(t_A[\mathcal{A}], t_B[\mathcal{A}]) \leq \varepsilon$) and having consistent scores, $score_J(t_A \bowtie_{P_J} t_B)$, greater than threshold α_J over all possible repair combinations, $\mathcal{D}_A^R \times \mathcal{D}_B^R$, given by:*

$$score_J(t_A \bowtie_{P_J} t_B) \quad (3)$$
$$= t_A.p \cdot t_B.p \cdot \sum_{\forall t_A \in \mathcal{D}_A^R} W_r(\mathcal{D}_A^R) \cdot \sum_{\forall t_B \in \mathcal{D}_B^R} W_r(\mathcal{D}_B^R) > \alpha_J,$$

where $t_A \in \mathcal{D}_A$, $t_B \in \mathcal{D}_B$, and $W_r(\mathcal{D}_\cdot^R)$ is given by Eq. (1).

The consistent score $score_J(t_A \bowtie_{P_J} t_B)$ in Eq. (3) indicates the probability that both tuples t_A and t_B appear in some possible worlds of repairs. The larger the consistent score, $score_J(t_A \bowtie_{P_J} t_B)$, is, the higher confidence it has that t_A and t_B join with each other in the (consistent) real world. Similar to α_R in PC-Range, threshold α_J indicates the confidence level, and can be specified by experienced users, based on the quality of historical query answers.

PC-Top$k$ Queries: Top-k queries have been extensively studied in the literature of consistent probabilistic databases [31, 34, 40, 23, 14, 27]. In particular, Li et al. [27] recently proposed a parameterized ranking definition for top-k queries, which incorporates many previously studied top-k semantics [31, 34, 40, 23, 14]. Thus, we will use this unified top-k definition [27] given below[2].

DEFINITION 3.4. (*Top-k Queries in Consistent Probabilistic Databases [27]*) *Given a consistent probabilistic database $\mathcal{D}$ and a preference function $f(\cdot)$, we define the score, $\Upsilon_w(\mathcal{D}, t_{jr})$, of an alternative $t_{jr} \in \mathcal{D}$ as:*

[2]We noted that some probabilistic top-k query semantics such as U-Top$k$ [34] cannot be captured by this unified definition [27], and we would like to leave it as future work.

Symbol	Description
$\mathcal{D}$	a probabilistic database containing N tuples
$\mathcal{D}^R$	a repaired probabilistic database (i.e., $\mathcal{D} - \Delta(\mathcal{D}^R)$)
$\mathcal{PW}(\mathcal{D})$	the set of *possible worlds*, $pw(\mathcal{D})$, in $\mathcal{D}$
$\mathcal{RW}(\mathcal{D})$	the set of *repair worlds*, $rw(\mathcal{D})$, in $\mathcal{D}$
$rpw(\mathcal{D})$	a *repaired possible world* of $\mathcal{D}$, where $rpw(\mathcal{D}) = pw(\mathcal{D})$ - $rw(\mathcal{D})$
t_j	an x-tuple in a probabilistic database
t_{jr}	an alternative (tuple) in x-tuple t_j ($1 \le r \le l_j$)
$t_{jr}.p$	the existent probability of tuple t_{jr}
$t_{jr}.rp$	the repair (deletion) confidence of tuple t_{jr}
$f(\cdot)$	the preference function specified by users
T_j	j alternatives in $\mathcal{D}$ with the highest $f(\cdot)$
$\mathcal{G}^{inc}$	an inconsistency graph of $\mathcal{D}$
$DI(t_{jr})$	a set of tuples that are inconsistent with t_{jr}

Table 5: Symbols and descriptions.

$$\Upsilon_w(\mathcal{D}, t_{jr}) = \sum_{i>0} w(t_{jr}, i) \cdot \sum_{pw(\mathcal{D}): t_{jr} \in pw(\mathcal{D})} Pr\{pw(\mathcal{D}) \wedge r_{pw(\mathcal{D})}(t_{jr}) = i\}, \tag{4}$$

where $w(t_{jr}, i)$ is a weight function of alternative t_{jr} only related to the rank i, and $r_{pw(\mathcal{D})}(t_{jr})$ is the rank of t_{jr} in possible world $pw(\mathcal{D})$. A probabilistic top-k query retrieves k alternatives in $\mathcal{D}$ with the highest scores.

In Eq. (4), with different weight function $w(t_{jr}, i)$, probabilistic top-k queries can have different top-k semantics [31, 34, 40, 23, 14, 27]; the score $\Upsilon_w(\mathcal{D}, t_{jr})$ for each alternative $t_{jr} \in \mathcal{D}$ is the (weighted) summation of probabilities that t_{jr} has rank i in the possible worlds; the top-k query answers are k alternatives with the highest score values, $\Upsilon_w(\mathcal{D}, \cdot)$.

Below, we formally define the PC-Top$k$ query in an inconsistent probabilistic database, which obtains k tuples with the highest consistent scores for all possible repairs (involving both query predicate $\mathcal{P}_T$ and consistent score predicate $\mathcal{P}_S$).

DEFINITION 3.5. (*Consistent Top-k Query in Inconsistent Probabilistic Databases, PC-Top$k$*) *Given an inconsistent probabilistic database $\mathcal{D}$, a top-k predicate $\mathcal{P}_T$: preference function $f(\cdot)$, and a consistent score predicate $\mathcal{P}_S$: k, a probabilistic consistent top-k query (PC-Top$k$) retrieves k alternatives, t_{jr}, such that their consistent scores,*

$$score_T(t_{jr}) = \sum_{\forall \mathcal{D}^R} \left(W_r(\mathcal{D}^R) \cdot \Upsilon_w(\mathcal{D}^R, t_{jr}) \right), \tag{5}$$

over all possible repairs $\mathcal{D}^R$ are the maximum, where $\Upsilon_w(\mathcal{D}^R, t_{jr})$ is given by Eq. (4).

In Definition 3.5, we consider all possible repaired databases, $\mathcal{D}^R$, and sum up the (weighted) scores, $\Upsilon_w(\mathcal{D}^R, t_{jr})$, of tuple t_{jr} under different repairs. Intuitively, higher consistent score, $score_T(t_{jr})$, indicates that tuple t_{jr} is more consistent in the query answers of the repaired databases, with high ranks and confidences. Thus, we will return k tuples with the highest consistent scores.

PC-Top$k$ is a bit different from the previous two queries, PC-Range and PC-Join. That is, PC-Top$k$ needs to compute consistent scores for candidates, involving both predicates $\mathcal{P}_T$ and $\mathcal{P}_S$, before obtaining top-k answers, whereas the checking of $\mathcal{P}_R$ in PC-Range or $\mathcal{P}_J$ in PC-Join does not require calculating any consistent scores. Therefore, as we will mention later, for query answering, while PC-Range and PC-Join can filter out false positives using predicates, say $\mathcal{P}_Q$, without considering $\mathcal{P}_S$, PC-Top$k$ has to do so via consistent scores w.r.t. both predicates $\mathcal{P}_Q$ and $\mathcal{P}_S$ at the same time.

We would like to leave the incorporation of other consistent query types (such as aggregations [26, 29]) into our CQA framework as future work. In particular, similar to the three queries above, we can consider every possible repair and summarize the query answers (e.g., aggregates) in all the repairs by consistent scores.

Challenges: One major challenge of CQA in inconsistent probabilistic databases is related to the efficiency issue. As we can see from Eqs. (2), (3), and (5), the consistent score, $score_.(t_{jr})$, for alternative t_{jr} is computed on all possible repairs of the database with possibly exponential size (i.e., $O(\prod_{j=1}^{N}(l_j + 1))$ in the worst case, where l_j is the number of alternatives for x-tuple t_j), which is quite inefficient for online query processing. Furthermore, for PC-Top$k$ in particular, its consistent score also needs to calculate $\Upsilon_w(\mathcal{D}^R, t_{jr})$ in Eq. (4) considering an exponential number of possible worlds. As a result, we have to conduct queries on combinations of repairs and possible worlds, both with exponential sizes. Thus, simple enumeration is inefficient or infeasible in terms of space and time costs.

Therefore, in the sequel, we propose a notion of repair worlds with a graph representation, with which CQA can be converted into the one on a set difference between possible worlds and repair worlds (called *repaired possible worlds*). Then, we present the detailed computation for each of consistent queries. To improve the CQA efficiency, we present either offline pre-computation (for PC-Range and PC-Join) or online filtering (for PC-Top$k$) techniques for fast retrieval. Table 5 summarizes the commonly used symbols.

4. PROBLEM REDUCTION

In order to efficiently process CQA in an inconsistent probabilistic database, we first propose the notion of *repair worlds* in a graph representation which contains those tuples to be repaired. Then, CQA can be considered as the one in the *repaired possible worlds*.

Note that, for the sake of clear illustration, in the sequel, we will present the solutions to probabilistic databases with single alternative per x-tuple. Thus, for an x-tuple t_j with its single alternative t_{j1}, we will omit the subscript "1" in t_{j1} if the context is clear. Nonetheless, the multi-alternative case can be easily extended by considering it as a special single-alternative case where any pairs of alternatives in the same x-tuple are inconsistent with each other.

4.1 Repair Worlds

Inconsistency Graph: To formalize the notion of repair worlds, we first give the definition of inconsistency graph, $\mathcal{G}^{inc}(\mathcal{D})$, in a probabilistic database.

DEFINITION 4.1. (*Inconsistency Graph, $\mathcal{G}^{inc}(\mathcal{D})$*) *Given an inconsistent probabilistic database $\mathcal{D}$, an inconsistency graph $\mathcal{G}^{inc}(\mathcal{D})$ is given by $\langle V, E \rangle$, where V is a vertex set and E is an edge set. In $\mathcal{G}^{inc}(\mathcal{D})$, each vertex $v_j \in V$ corresponds to an alternative of an x-tuple $t_j \in \mathcal{D}$; any edge $v_j v_l$ is in E iff alternatives t_j and t_l are inconsistent, violating integrity constraints; and any vertex $v_j \in V$ is connected to at least one edge in E. For the multi-alternative case, any two alternatives in the same x-tuple have an edge between them.*

Figure 1 illustrates a simple example of an inconsistency graph $\mathcal{G}^{inc}(\mathcal{D})$ with vertices $v_1 \sim v_4$ corresponding to tuples $t_1 \sim t_4$, respectively, where tuple pairs $\langle t_1, t_2 \rangle$, $\langle t_1, t_3 \rangle$, $\langle t_2, t_3 \rangle$, and $\langle t_2, t_4 \rangle$ are inconsistent (e.g., violating some functional dependencies).

Note that, the inconsistency graph is similar to the notion of *conflict hypergraph* [12]. The difference is that in the conflict hypergraph, each hyper-edge connects a number of tuples (vertices) from different joining tables that violate the integrity constraints. In contrast, each edge of our inconsistency graph pairwisely connects any two inconsistent tuples in the same probabilistic database.

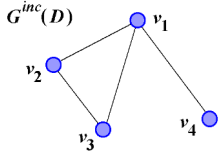


Figure 1: Illustration of inconsistency graph.

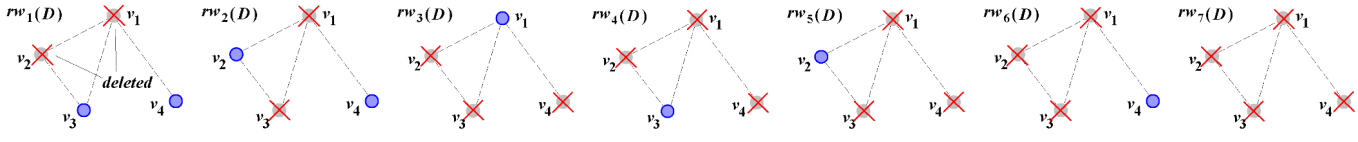


Figure 2: Illustration of 7 repair worlds of the inconsistency graph $\mathcal{G}^{inc}$ in Figure 1. (a) $rw_1(\mathcal{D}) = \{t_1, t_2\}$. (b) $rw_2(\mathcal{D}) = \{t_1, t_3\}$. (c) $rw_3(\mathcal{D}) = \{t_2, t_3, t_4\}$. (d) $rw_4(\mathcal{D}) = \{t_1, t_2, t_4\}$. (e) $rw_5(\mathcal{D}) = \{t_1, t_3, t_4\}$. (f) $rw_6(\mathcal{D}) = \{t_1, t_2, t_3\}$. (g) $rw_7(\mathcal{D}) = \{t_1, t_2, t_3, t_4\}$.

Repair Worlds: Next, we define the repair world, denoted as $rw(\mathcal{D})$, using the graph representation mentioned above, which corresponds to one possible repair of a probabilistic database $\mathcal{D}$. As a result, the repair process in the probabilistic database is exactly equivalent to the removal of vertices (and their connected edges as well) in its inconsistency graph such that no edges exist after vertex deletions.

DEFINITION 4.2. (*Repair World, $rw(\mathcal{D})$*) *Given an inconsistency graph $\mathcal{G}^{inc}(\mathcal{D}) = \langle V, E\rangle$ in an inconsistent probabilistic database $\mathcal{D}$, a repair world $rw(\mathcal{D})$ is a set of tuples, $\{t_j | t_j \in rw(\mathcal{D})\}$, such that: (1) each tuple t_j corresponds to a vertex v_j in V, and (2) by removing all the vertices v_j of $t_j \in rw(\mathcal{D})$ from $\mathcal{G}^{inc}(\mathcal{D})$, as well as edges connected with them, the resulting graph has an empty edge set. The confidence (weight), $Pr\{rw(\mathcal{D})\}$, of a repair world $rw(\mathcal{D})$ is given by $W_r(\mathcal{D}^R)$ in Eq. (1), where $\mathcal{D}^R = \mathcal{D} - rw(\mathcal{D})$.*

Figures 2(a)∼2(g) show seven repair worlds, $rw_1(\mathcal{D}) \sim rw_7(\mathcal{D})$, of the inconsistency graph $\mathcal{G}^{inc}$ in Figure 1, where each repair world $rw_i(\mathcal{D})$ is mapped to the deletion of tuples (vertices) in the probabilistic database (inconsistency graph). As an example in Figure 2(a), by deleting vertices v_1 and v_2 from $\mathcal{G}^{inc}$, the resulting inconsistency graph will not contain any edges. Correspondingly, tuples t_1 and t_2 form a repair world $rw_1(\mathcal{D})$, indicating that by removing t_1 and t_2 from $\mathcal{D}$, the resulting database will be consistent.

4.2 Repaired Possible Worlds

After introducing the notion of repair worlds, we now present the concept of *repaired possible worlds* in which CQA is conducted. We denote $\mathcal{RW}(\mathcal{D})$ as the set of all repair worlds $rw_l(\mathcal{D})$ in an inconsistent probabilistic database $\mathcal{D}$, that is, $\mathcal{RW}(\mathcal{D}) = \{rw_l(\mathcal{D}) \mid \forall l\}$, and $\mathcal{PW}(\mathcal{D})$ as the set of all possible worlds in $\mathcal{D}$. Our CQA problem is actually answered on the Cartesian product $\mathcal{RW}(\mathcal{D}) \times \mathcal{PW}(\mathcal{D})$.

Specifically, for any repair world $rw(\mathcal{D}) \in \mathcal{RW}(\mathcal{D})$ and any possible world $pw(\mathcal{D}) \in \mathcal{PW}(\mathcal{D})$, we can combine them using the operator of set difference, and obtain a *repaired possible world*, denoted as $rpw(\mathcal{D})$, by removing all the tuples in $rw(\mathcal{D})$ from $pw(\mathcal{D})$ (i.e., $rpw(\mathcal{D}) = pw(\mathcal{D}) - rw(\mathcal{D})$). The existence probability $Pr\{rpw(\mathcal{D})\}$ of $rpw(\mathcal{D})$ is given by:

$$Pr\{rpw(\mathcal{D})\} = Pr\{rw(\mathcal{D})\} \cdot Pr\{pw(\mathcal{D})\} = W_r(\mathcal{D}^R) \cdot Pr\{pw(\mathcal{D})\}. \quad (6)$$

Therefore, the CQA problem is equivalent to first conducting queries in each repaired possible world and then combining the query answers from all the repaired possible worlds.

4.3 PC-Range and PC-Join Computations

In this subsection, we discuss the CQA computation of PC-Range and PC-Join queries in the repaired possible worlds. Specifically, as given in Eq. (6), since the confidence of repair worlds $Pr\{rw(\mathcal{D})\}$ is equal to the repair weight $W_r(\mathcal{D}^R)$, we can rewrite Eq. (2) in Definition 3.2 and Eq. (3) in Definition 3.3, respectively, as:

$$\begin{aligned} score_R(t_j) &= t_j.P_{rpw} \quad (7)\\ &= t_j.p \cdot (1 - t_j.P_{rw})\\ &= t_j.p \cdot Pr\{t_j \notin rw(\mathcal{D}) \mid \forall rw(\mathcal{D}) \in \mathcal{RW}(\mathcal{D})\}, \text{ and} \end{aligned}$$

$$score_J(t_A \bowtie_{P_J} t_B) = t_A.P_{rpw} \cdot t_B.P_{rpw}. \quad (8)$$

where $t_j.P_{rpw}$ is the probability that t_j is in the repaired possible worlds, and $t_j.P_{rw}$ is the probability that t_j is in the repair worlds.

In Eq. (7), since the repaired possible worlds are the set differences between possible worlds and repair worlds, $t_j.P_{rpw}$ can be given by the probability that t_j appears in the possible worlds (i.e., $t_j.p = Pr\{t_j \in pw(\mathcal{D}) \mid \forall pw(\mathcal{D}) \in \mathcal{PW}(\mathcal{D})\}$) times the probability that t_j does not appear in the repair worlds (i.e., $1 - t_j.P_{rw} = Pr\{t_j \notin rw(\mathcal{D}) \mid \forall rw(\mathcal{D}) \in \mathcal{RW}(\mathcal{D})\}$).

Therefore, the consistent score of t_j for PC-Range equals to probability $t_j.P_{rpw}$, whereas that for PC-Join equals to the multiplication of $t_A.P_{rpw}$ and $t_B.P_{rpw}$ from two joining tuples t_A and t_B in databases $\mathcal{D}_A$ and $\mathcal{D}_B$, respectively.

Computation of Probability $t_j.P_{rpw}$: We now address the issue about obtaining the probability $t_j.P_{rpw}$ (in Eqs. (7) and (8)), which is the probability that t_j appears in the repaired possible world $rpw(\mathcal{D})$. In turn, we only need to compute the probability that t_j is not in the repair worlds, that is, $(1 - t_j.P_{rw})$. We can derive the formula for probability $t_j.P_{rw}$ with the help of the inconsistency graph $\mathcal{G}^{inc}$ as follows.

Let $N_{\mathcal{RW}}(\mathcal{G}^{inc})$ be the total weight of repair worlds in the inconsistency graph $\mathcal{G}^{inc}$, that is, $N_{\mathcal{RW}}(\mathcal{G}^{inc}) = \sum_{\forall \mathcal{D}^R} W_r(\mathcal{D}^R) = \sum_{\forall rw(\mathcal{D}) \in \mathcal{RW}(\mathcal{D})} W_r(\mathcal{D} - rw(\mathcal{D}))$. Therefore, the probability $t_j.P_{rw}$ is given by the weight summation of repair worlds that does not contain t_j (i.e., $t_j.rp \cdot N_{\mathcal{RW}}(\mathcal{G}^{inc} - \{v_j\})$) divided by the total weight $N_{\mathcal{RW}}(\mathcal{G}^{inc})$ of graph $\mathcal{G}^{inc}$. That is,

$$t_j.P_{rw} = \frac{t_j.rp \cdot N_{\mathcal{RW}}(\mathcal{G}^{inc} - \{v_j\})}{N_{\mathcal{RW}}(\mathcal{G}^{inc})}. \quad (9)$$

Specifically, $N_{\mathcal{RW}}(\mathcal{G}^{inc})$ can be recursively computed by:

$$N_{\mathcal{RW}}(\mathcal{G}^{inc}) = \begin{cases} N_{\mathcal{RW}}(\mathcal{G}^{inc} - \{v_l\}), & \text{if } \forall v_l \in \mathcal{G}^{inc} \wedge DI(t_l) = \emptyset.\\ (1 - t_l.rp) \cdot (\prod_{\forall t_m \in DI(t_l)} t_m.rp) & \\ \quad \cdot N_{\mathcal{RW}}(\mathcal{G}^{inc} - \{v_l\} - DI(v_l)) + t_l.rp & \\ \quad \cdot N_{\mathcal{RW}}(\mathcal{G}^{inc} - \{v_l\}), & \text{if } \forall v_l \in \mathcal{G}^{inc} \wedge DI(t_l) \neq \emptyset.\\ 1, & \text{if } \mathcal{G}^{inc} \text{ contains no edges;} \end{cases} \quad (10)$$

where $DI(t_j)$ (or $DI(v_j)$ for inconsistency graph $\mathcal{G}^{inc}$) is a set of tuples (vertices) that are *directly inconsistent* with t_j (i.e., connecting with vertices v_j). The intuition of Eq. (10) is given below.

- **Case 1:** When a vertex $v_l \in \mathcal{G}^{inc}$ is not connected to any other vertices, vertex v_l can either exist or not exist in repair worlds. Thus, in this case, we have $N_{\mathcal{RW}}(\mathcal{G}^{inc}) = N_{\mathcal{RW}}(\mathcal{G}^{inc} - \{v_l\})$.

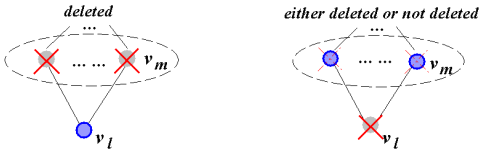


Figure 3: Illustration of $N_{\mathcal{RW}}$ computation in Eq. (10) where $v_l \in \mathcal{G}^{inc} \wedge DI(t_l) \neq \emptyset$. (a) v_l is not deleted. (b) v_l is deleted.

- **Case 2:** When a vertex $v_l \in \mathcal{G}^{inc}$ is connected to some vertices $v_m \in DI(v_l)$, we consider two subcases:
  - **Case 2.1:** If v_l is not deleted, as shown in Figure 3(a), all the vertices $v_m \in DI(v_l)$ must be deleted (based on the repair rule that at least one vertex of an edge should be deleted), thus contributing to weight a factor of $(1 - t_l.rp) \cdot (\prod_{\forall t_m \in DI(t_l)} t_m.rp)$;
  - **Case 2.2:** If t_l is deleted, as shown in Figure 3(b), $t_m \in DI(t_l)$ can either exist or not exist in the repair worlds, thus t_l contributing a factor of $t_l.rp$.
- **Case 3:** In the base case where no edges are left, the remaining vertices can be either deleted or not deleted, thus, the weight is set to 1.

4.4 PC-Top$k$ Computation

In this subsection, we present how to answer PC-Top$k$ queries under the repaired possible worlds semantics. Similar to PC-Range and PC-Join, we can rewrite the PC-Top$k$ query given in Definition 3.5 by considering the repaired possible worlds.

DEFINITION 4.3. (*PC-Top$k$ Query in Repaired Possible Worlds*) *Given an inconsistent probabilistic database $\mathcal{D}$, a PC-Top$k$ query retrieves k tuples, t_j, such that:*

$$score_T(t_j) = \sum_{i>0} w(t_j, i) \cdot \sum_{\forall rpw(\mathcal{D}): t_j \in rpw(\mathcal{D})} Pr\{rpw(\mathcal{D}) \wedge r_{rpw(\mathcal{D})}(t_j) = i\}, \tag{11}$$

are the maximum, where $rpw(\mathcal{D}) = pw(\mathcal{D}) - rw(\mathcal{D})$, $\forall pw(\mathcal{D}) \in \mathcal{PW}(\mathcal{D})$ and $\forall rw(\mathcal{D}) \in \mathcal{RW}(\mathcal{D})$.

We prove below that Definitions 4.3 and 3.5 are equivalent.

LEMMA 4.1. *Definition 4.3 is an equivalent version of Definition 3.5 in repaired possible worlds.*

Proof. It is sufficient to prove that Eq. (11) can be obtained by combining Eqs. (4) and (5). In particular, we substitute $\Upsilon_w(\mathcal{D}, t_j)$ in Eq. (4) into Eq. (5), resulting in:

$$\begin{aligned} score_T(t_j) \quad = \quad & \sum_{\forall \mathcal{D}^R = \mathcal{D} - rw(\mathcal{D})} W_r(\mathcal{D}^R) \cdot \sum_{i>0} w(t_j, i) \\ & \cdot \sum_{pw(\mathcal{D}): t_j \in pw(\mathcal{D})} Pr\{pw(\mathcal{D}) \wedge r_{pw(\mathcal{D})}(t_j) = i\}. \end{aligned} \tag{12}$$

By swapping the orders of summations in Eq. (12), we have:

$$\begin{aligned} score_T(t_j) = & \sum_{\forall rw(\mathcal{D})} W_r(\mathcal{D}^R) \cdot \sum_{pw(\mathcal{D}): t_j \in pw(\mathcal{D})} \sum_{i>0} w(t_j, i) \\ & \cdot Pr\{pw(\mathcal{D}) \wedge r_{pw(\mathcal{D})}(t_j) = i\}. \end{aligned} \tag{13}$$

Then, by considering the pairwise combinations between repair worlds $rw(\mathcal{D})$ and possible worlds $pw(\mathcal{D})$, we obtain:

$$\begin{aligned} & score_T(t_j) \\ = & \sum_{\forall rpw(\mathcal{D}): t_j \in rpw(\mathcal{D})} \sum_{i>0} w(t_j, i) \cdot Pr\{rpw(\mathcal{D}) \wedge r_{rpw(\mathcal{D})}(t_j) = i\}, \end{aligned} \tag{14}$$

which, by swapping the two summations above, is exactly Eq. (11). Thus, the lemma holds. □

Thus, according to Lemma 4.1, the PC-Top$k$ query can be considered to the one in the repaired possible worlds as given in Definition 4.3. As mentioned earlier, to check query predicates $\mathcal{P}_T$ and consistent score predicates $\mathcal{P}_S$ of PC-Top$k$ queries, we have to efficiently compute the consistent scores $score_T(t_j)$ in Eq. (11).

Without loss of generality, assume that we can retrieve tuples t_j from the database $\mathcal{D}$, in descending order of their query preferences $f(t_j)$, for example, $\langle t_1, t_2, ..., t_l \rangle$ where $f(t_1) \geq f(t_2) \geq ... \geq f(t_l)$. Denote T_l as an ordered sequence $\langle t_1, t_2, ..., t_l \rangle$ with the highest preference scores. Then, we can simplify Eq. (11) as:

$$score_T(t_j) = \sum_{i>0} w(t_j, i) \cdot M(t_j, i), \tag{15}$$

where

$$\begin{aligned} & M(t_j, i) \\ = & \sum_{\forall rpw(\mathcal{D}): t_j \in rpw(\mathcal{D})} Pr\{rpw(\mathcal{D}) \wedge r_{rpw(\mathcal{D})}(t_j) = i\}, \\ = \; & t_j.P_{rpw} \cdot \sum_{\forall rpw(\mathcal{D})} Pr\{|rpw(\mathcal{D}) \cap T_{j-1}| = i - 1 \mid \{t_j\} \subseteq rpw(\mathcal{D})\}. \end{aligned} \tag{16}$$

Intuitively, the first term (i.e., $t_j.P_{rpw}$) on RHS of Eq. (16) is the probability that t_j exists in the repaired possible worlds $rpw(\mathcal{D})$, whereas the second term is a conditional probability that there are exactly $(i-1)$ tuples having ranks higher than t_j and appearing in $(rpw(\mathcal{D}) \cap T_{j-1})$, given that t_j exists in $rpw(\mathcal{D})$. Note that, based on the property of our repair rule, if t_j exists in $rpw(\mathcal{D})$, then tuples in $DI(t_j)$ (inconsistent with t_j) cannot appear in $rpw(\mathcal{D})$. Thus, the second term is equivalent to $\sum_{\forall rpw(\mathcal{D})} Pr\{|rpw(\mathcal{D}) \cap (T_{j-1} - DI(t_j))| = i - 1 \mid \{t_j\} \subseteq rpw(\mathcal{D})\}$.

Computation of Function $M(t_j, i)$: In order to calculate function $M(t_j, i)$ in Eq. (16), we can further rewrite it as a function of $R(\cdot)$:

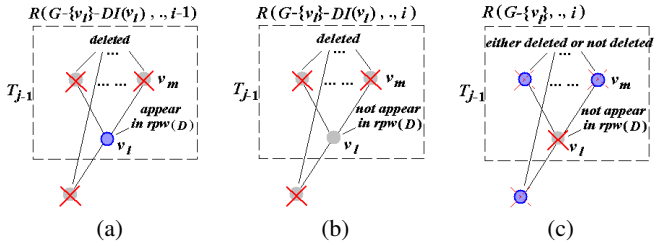
$$\begin{aligned} M(t_j, i) \quad = \quad & t_j.P_{rpw} \cdot R(\mathcal{G}^{inc} - DI(v_j) - \{v_j\}, \\ & T_{j-1} - DI(t_j) - \{t_j\}, i - 1). \end{aligned} \tag{17}$$

where $R(\mathcal{G}, S, i)$ is given by the recursive function below:

$$R(\mathcal{G}, S, i) = \begin{cases} \begin{aligned} & t_l.p \cdot (1 - t_l.rp) \cdot \textstyle\prod_{\forall t_m \in DI(t_l)} t_m.rp \cdot R(\mathcal{G} - \{v_l\} \\ & - DI(v_l), S - \{t_l\} - DI(t_l), i - 1) \\ & + (1 - t_l.p) \cdot (1 - t_l.rp) \cdot \textstyle\prod_{\forall t_m \in DI(t_l)} t_m.rp \\ & \cdot R(\mathcal{G} - \{v_l\} - DI(v_l), S - \{t_l\} - DI(t_l), i) \\ & + t_l.rp \cdot R(\mathcal{G} - \{v_l\}, S - \{t_l\}, i), \end{aligned} & \text{if } t_l \in T_{j-1} \text{ and } |S| \geq i; \\ \begin{aligned} & (1 - t_l.rp) \cdot \textstyle\prod_{\forall t_m \in DI(t_l)} t_m.rp \cdot R(\mathcal{G} - \{v_j\} \\ & - DI(v_j), S - DI(t_l), i) + t_l.rp \cdot R(\mathcal{G} - \{v_j\}, S, i), \end{aligned} & \text{if } t_l \notin T_{j-1} \text{ and } |S| \geq i; \\ \textstyle\prod_{\forall t_m \in S}(1 - t_m.p \cdot (1 - t_m.rp)), & \text{if } S \neq \emptyset \text{ and } i = 0; \\ 0 & \text{otherwise.} \end{cases} \tag{18}$$

Intuitively, a function $R(\mathcal{G}, S, i)$ returns the probability that there are i tuples in S appearing in repaired possible worlds. To compute $R(\cdot)$, we consider 4 cases, as shown in Eq. (18) above.

- **Case 1:** When a tuple t_l in T_{j-1} is found and there are no fewer than i tuples left in S, tuple t_l either appears or does not appear in repaired possible worlds.
  - **Case 1.1:** As illustrated in Figure 4(a), if $t_l \in rpw(\mathcal{D})$, then its appearance probability is given by the probability that t_l exists in possible worlds but not repair worlds (i.e., $t_l.p \cdot (1 - t_l.rp)$). Moreover, based on the repair rule, since tuples in $DI(t_l)$ (inconsistent with t_l) cannot appear in the repaired possible worlds. Thus, tuples in $DI(t_l)$ contribute weights $\prod_{\forall t_m \in DI(t_l)} t_m.rp$. Then, we recursively invoke function $R(\cdot)$, considering the graph G (or set S) excluding vertices in $\{v_l\} \cup$



(a) (b) (c)

Figure 4: Illustration of $R(\cdot)$ computation in Eq. (18) where $t_l \in T_j$ and $|S| \geq i$. (a) $t_l \in rpw(\mathcal{D})$. (b) $t_l \notin rpw(\mathcal{D}) \wedge t_l \notin rw(\mathcal{D})$. (c) $t_l \notin rpw(\mathcal{D}) \wedge t_l \in rw(\mathcal{D})$.

$DI(v_l)$ (or $\{t_l\} \cup DI(t_l)$) and with the parameter of tuple number $(i-1)$.

- **Case 1.2:** If $t_l \notin rpw(\mathcal{D})$, there are two subcases. When t_l is not in repair worlds (Figure 4(b)), it has appearance probability $(1-t_l.p)\cdot(1-t_l.rp)$, and those tuples in $DI(t_l)$ should be deleted (contributing weight $\prod_{\forall t_m \in DI(t_l)} t_m.rp$, similar to Case 1.1); when t_l is in repair worlds (Figure 4(c)), it has probability $t_j.rp$ to appear in repair worlds. Then, we recursively invoke $R(\cdot)$, with the tuple number i.

- **Case 2:** When a tuple t_l not in T_{j-1} is visited and there are no fewer than i tuples left in S, there are also two subcases where t_l is either in or not in the repair worlds. Since their appearances in the repaired worlds do not affect the ranking of tuple t_j, the recursive invocation of $R(\cdot)$ will always take tuple number i as the third parameter.

- **Case 3:** When the set S is not empty and $i = 0$, any tuple in set S should not appear in the repaired possible worlds. Thus, S contributes weight $\prod_{\forall t_m \in S}(1-t_m.p\cdot(1-t_m.rp))$.

- **Case 4:** For other cases, function $R(\cdot)$ outputs 0.

5. CQA PROCESSING

Up to now, we have illustrated the baseline approach for computing consistent scores for PC-Range, PC-Join, and PC-Top$k$. In this section, we illustrate the CQA query processing approach in the inconsistent probabilistic database. Specifically, for PC-Range and PC-Join, since their consistent scores are related to probability $t_j.P_{rpw}$ for any tuple t_j, we can offline pre-compute this probability, which can facilitate online CQA for PC-Range and PC-Join and efficiently return the consistent scores. Moreover, for PC-Top$k$, due to its costly online computation of consistent scores involving ranks, we propose an effective pruning strategy to reduce the search space, as well as an early stopping condition.

Data Pre-Processing: Without loss of generality, we assume that probabilistic data in the database(s) are indexed by an R*-tree [5], on which CQA queries can be processed. Specifically, Figure 5 illustrates the pseudo-code of data pre-processing procedure, namely Pre_Processing. Given an inconsistent probabilistic database $\mathcal{D}$, we offline pre-process the data by constructing an R-tree index (lines 1-5). That is, we first enumerate pairwise tuples in the database and check whether or not they are consistent with each other. If not, then there is an edge connecting their corresponding vertices in an inconsistency graph $\mathcal{G}^{inc}$ (line 1). Then, for each alternative t_j, we pre-compute probabilities such as $t_j.P_{rw}$ and $t_j.P_{rpw}$, which can facilitate fast calculating/outputing consistent scores for online CQA (lines 2-4). Next, we construct an R*-tree [5], $\mathcal{I}$, by inserting probabilistic tuples (associated with pre-computed values) into the index using standard "insert" operator. Each node also stores the maximum consistent score $t_j.P_{rpw}$ for all the tuples under it. This index can help retrieve tuples t_j satisfying the query and consistent score predicates during CQA (line 5).

5.1 PC-Range and PC-Join Query Processing

PC-Range Query Procedure: Figure 6 illustrates the query procedure, namely procedure PC-Range_Processing, for retrieving the PC-Range query answers. Specifically, the query procedure traverses those nodes/tuples in the index $\mathcal{I}$ that may satisfy the range predicate $\mathcal{P}_R$, for example, intersecting with a query region QR, and return the resulting tuples with consistent scores greater than a threshold $\alpha_R \in [0, 1)$. We maintain a candidate set S_{cand} storing PC-Range candidates (line 1). Initially, we traverse the index $\mathcal{I}$ [5] and retrieve all the candidates in S_{cand}. Note that, during the index traversal, we visit those nodes that both intersect with the query region QR in the range predicate $\mathcal{P}_R$ and have their MAX score aggregates in nodes greater than threshold α_R (since all tuples under a node can be safely pruned in case its MAX aggregate $\leq \alpha_R$; line 2). Then, for each tuple $t_j \in S_{cand}$, we retrieve its pre-computed consistent score $score_R(t_j) = t_j.P_{rpw}$, and (if $score_R(t_j) > \alpha_R$) output it to the PC-Range answer set $PC\text{-}Range\_ans$ (lines 3-6). Finally, we return PC-Range answers in $PC\text{-}Range\_ans$ (line 7).

Procedure Pre_Processing {
Input: an inconsistent probabilistic database $\mathcal{D}$ and integrity constraints $\mathcal{IC}$
Output: inconsistency graph $\mathcal{G}^{inc}$ and index $\mathcal{I}$
(1) obtain pairwise inconsistent alternatives in $\mathcal{D}$ and construct an inconsistency graph $\mathcal{G}^{inc}$ from $\mathcal{IC}$
(2) for each inconsistent tuple $t_j \in \mathcal{D}$
(3) pre-compute $t_j.P_{rw}$ and $t_j.P_{rpw}$ // Eqs. (9) and (7)
(4) pre-compute other statistics // e.g., $max\{t_m.p, 1-t_m.p\}$
(5) construct an R-tree index $\mathcal{I}$ over $\mathcal{D}$, where each tuple is associated with pre-computed values and each node has an MAX consistent score aggregate
}

Figure 5: Data pre-processing for CQA.

Procedure PC-Range_Processing {
Input: index $\mathcal{I}$ on an inconsistent probabilistic database $\mathcal{D}$, a range predicate $\mathcal{P}_R$: query region QR, and a consistent score predicate $\mathcal{P}_S : \alpha_R$
Output: the answer to the PC-Range query
(1) $PC\text{-}Range\_ans = \emptyset$; $S_{cand} = \emptyset$;
(2) traverse the index $\mathcal{I}$ and obtain S_{cand} satisfying range predicates $\mathcal{P}_R$ and with MAX consistent score aggregates in nodes $> \alpha_R$
(3) for each $t_j \in S_{cand}$
(4) let $score_R(t_j) = t_j.P_{rpw}$ // Eq. (7)
(5) if $score_R(t_j) > \alpha_R$
(6) add $\langle t_j, score_R(t_j)\rangle$ to $PC\text{-}Range\_ans$
(7) return $PC\text{-}Range\_ans$
}

Figure 6: Procedure of PC-Range query answering.

PC-Join Query Procedure: The PC-Join query is similar to PC-Range. The difference is that, rather than one single database, PC-Join is conducted over two inconsistent probabilistic databases $\mathcal{D}_A$ and $\mathcal{D}_B$ (with indexes $\mathcal{I}_A$ and $\mathcal{I}_B$, respectively) with join predicates $\mathcal{P}_J$, for example, tuple pairs within ε-distance from each other. Thus, we first conduct spatial join [24], where on the node level, the multiplication of MAX aggregates in node pairs should be greater than α_J. Then, we can obtain a candidate set S_{cand} containing pairs, $\langle t_A, t_B\rangle$, satisfying the join predicate $\mathcal{P}_J : dist(t_A[\mathcal{A}], t_B[\mathcal{A}]) \leq \varepsilon$ on join attributes in $\mathcal{A}$, where $t_A \in \mathcal{D}_A$ and $t_B \in \mathcal{D}_B$. The consistent score of $\langle t_A, t_B\rangle$ is given by Eq. (8), using the pre-computed values $t_A.P_{rpw}$ and $t_B.P_{rpw}$. We omit the similar pseudo code here.

5.2 PC-Top$k$ Query Processing

5.2.1 PC-Top$k$ Pruning Rationale

Recall from Section 4.4 that, to answer PC-Top$k$ queries involving both predicates $\mathcal{P}_T$ and $\mathcal{P}_S$, we need to compute the consistent score $score_T(t_j)$ given by Eq. (15), and in turn we only need to calculate the function $M(t_j, i)$. However, the exact computation of function $M(t_j, i)$ for some tuple t_j and rank i may take exponential time complexity in the worst case. Inspired by this, in the sequel, we propose a pruning technique, which computes the lower/upper bounds of $M(t_j, i)$ at a lower cost.

Without loss of generality, we denote lower and upper bounds of $M(t_j, i)$ as $M_{lb}(t_j, i)$ and $M_{ub}(t_j, i)$, respectively. That is, $M_{lb}(t_j, i) \leq M(t_j, i) \leq M_{ub}(t_j, i)$. Then, the consistent score $score_T(t_j)$ in Eq. (15) can be bounded by an interval $[score_{lb}(t_j), score_{ub}(t_j)]$, where

$$score_{lb}(t_j) = \sum_{i>0} w(t_j, i) \cdot M_{lb}(t_j, i), \tag{19}$$

$$score_{ub}(t_j) = \sum_{i>0} w(t_j, i) \cdot M_{ub}(t_j, i). \tag{20}$$

Therefore, we have the pruning rule for PC-Top$k$ as follows.

LEMMA 5.1. *(Pruning Rule for PC-Topk) Assume that we have seen $(j-1)$ tuples in T_{j-1} so far, and let threshold τ be the k-th largest score lower bound among tuples in T_{j-1}. Then, tuple t_j can be safely pruned, if it holds that $score_{ub}(t_j) < \tau$.*

Proof. Since $score_{ub}(t_j)$ is an upper bound of consistent score $score_T(t_j)$, we have $score_T(t_j) \leq score_{ub}(t_j)$. Then, by the lemma assumption ($score_{ub}(t_j) < \tau$) and the inequality transition, it holds that $score_T(t_j) < \tau$. Since threshold τ is the k-th largest score lower bound for tuples we have seen so far, there must exist k tuples in T_{j-1} whose scores are not smaller than t_j. Thus, from Definition 3.5, t_j cannot be the answer to the PC-Top$k$ query. □

Thus, by utilizing Lemma 5.1, we can prune tuples whose consistent scores are definitely lower than those we have seen.

5.2.2 Lower/Upper Score Bounds

Computation of Lower/Upper Bounds for $M(t_j, i)$: We next derive lower/upper bounds for $M(t_j, i)$ in Eq. (17), where $M(t_j, i)$ is the probability that tuple t_j has rank i in some repaired possible worlds $rpw(\mathcal{D})$. That is, we want to obtain the weight that, among $(j-1)$ tuples in T_{j-1}, there are exactly $(i-1)$ tuples appearing in possible worlds $pw(\mathcal{D})$ but not in repair worlds $rw(\mathcal{D})$.

Now recall from Eq. (10) that, we can compute the total weight, $N_{\mathcal{RW}}(\mathcal{G}^{inc})$, of repair worlds in an inconsistency graph $\mathcal{G}^{inc}$, which is a summation of appearance probabilities for all the repair worlds. Thus, $(t_j.P_{rpw} \cdot N_{\mathcal{RW}}(\mathcal{G}^{inc}))$ is the weight that t_j is in repaired possible worlds.

Next, to compute lower/upper bounds for $M(t_j, i)$, we further consider the $(j-1)$ positions in T_{j-1} having higher ranks than t_j. In fact, given any repair world $rw(\mathcal{D})$, there are two possible cases for any tuple (position) $t_m \in T_{j-1}$. The first case is that t_m is not in repair worlds $rw(\mathcal{D})$ holds. In this case, the conditional probability:

$$Pr\{t_m \in rpw(\mathcal{D}) | t_m \notin rw(\mathcal{D})\} = t_m.p.$$

Similarly, we have:

$$Pr\{t_m \notin rpw(\mathcal{D}) | t_m \notin rw(\mathcal{D})\} = 1 - t_m.p.$$

On the other hand, the second case is that $t_m \in rw(\mathcal{D})$ holds (no matter whether or not $t_m \in pw(\mathcal{D})$). In this case, we have the conditional probability:

$$Pr\{t_m \notin rpw(\mathcal{D}) | t_m \in rw(\mathcal{D})\} = 1.$$

Without loss of generality, let n_{min} and n_{max} be the minimum and maximum numbers of deleted tuples in T_{j-1} for any repair worlds. We can infer that, among $(j-1)$ positions in T_{j-1}, there are $n \in [n_{min}, n_{max}]$ positions corresponding to the second case above (i.e., tuples in these positions are in repair worlds). Moreover, there are $(i-1)$ positions corresponding to the first case where tuples appear in the repaired possible worlds. The remaining $(j-n-i)$ tuples corresponding to the first case where tuples do not appear in the repaired possible worlds.

Therefore, assuming W_m is the m-th largest existence probability $t_l.p$ for any $t_l \in T_{j-1}$ (i.e., $W_1 \leq W_2 \leq ... \leq W_{j-1}$), we can obtain the bounds for $M(t_j, i)$ by the following lemma.

LEMMA 5.2. *Lower/upper bounds of $M(T_j, i)$ are given by:*

$$t_j.P_{rpw} \cdot N_{\mathcal{RW}}(\mathcal{G}^{inc}) \cdot \prod_{m=1}^{i-1} W_m \cdot \prod_{m=i+n_{min}}^{j-1} (1 - W_m) \leq M(t_j, i)$$

$$\leq t_j.P_{rpw} \cdot N_{\mathcal{RW}}(\mathcal{G}^{inc}) \cdot \prod_{m=j-i+1}^{j-1} W_m \cdot \prod_{m=1}^{j-i-n_{max}} (1 - W_m). \quad (21)$$

Proof. Derived from the definition of $M(T_j, i)$ in Eq. (16). □

Early Stopping Condition: By utilizing the derived upper bound in Eq. (21), we can give early stopping condition. That is,

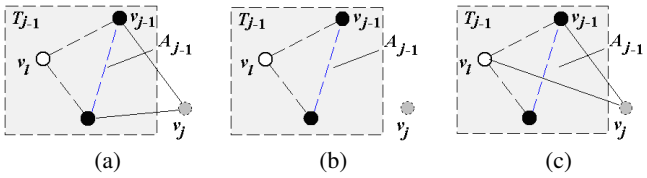


Figure 7: Illustration of incremental A_j computation ("●" represents labeled vertex and "○" denotes unlabeled one). (a) $|A_j| = |A_{j-1}|$, and v_j is unlabeled. (b) $|A_j| = |A_{j-1}|$, and v_j is unlabeled. (c) $|A_j| = |A_{j-1}| + 1$, and v_j and v_l are labeled.

LEMMA 5.3. *Let $\theta_j = max_{\forall t_m}\{t_m.P_{rpw}\} \cdot N_{\mathcal{RW}}(\mathcal{G}^{inc}) \cdot \prod_{m=1}^{j-n_{max}-1} max\{W_m, 1-W_m\}$. Then, if it holds that $\theta_j \leq \tau$, we can terminate the query procedure early.*

Computation of n_{min} and n_{max}: As mentioned earlier, the repair of an inconsistent probabilistic database is equivalent to vertex deletions in its inconsistency graph. Thus, the computation of n_{min} and n_{max} is exactly to find minimum and maximum numbers of deleted vertices in the graph such that no edge exists.

Clearly, the value of n_{max} can be given by the number of inconsistent tuples in T_{j-1} (i.e., we delete all inconsistent tuples in $\mathcal{G}^{inc}$). However, for the problem of computing n_{min}, it is actually equivalent to the one that finds the *minimum vertex cover* in a graph[3]. Since this problem is NP-hard, one can use factor-2 approximation, which repeatedly adds an edge to a set A, then removes its two endpoints from the graph. The resulting edge set A has the property that $|A| \leq |OPT|$, where OPT is the minimum vertex cover and $|\cdot|$ is the cardinality of a set. Thus, we let $n_{min} = |A|$.

Since we need to compute the lower/upper score bounds in Inequality (21) for each tuple t_j we encounter, it is not efficient to online compute n_{min} by the factor-2 approximation algorithm every time from scratch. Therefore, below, we discuss how to incrementally compute a lower bound, $|A_j|$ $(= n_{min})$, of $|OPT|$ for T_j, given T_{j-1} with $|A_{j-1}|$. Specifically, we assume that each vertex in the graph is either labeled or unlabeled. Labeled vertices are endpoints of some edges selected in the set A_j.

Without loss of generality, denote $\mathcal{G}_j^{inc}$ as a subgraph of inconsistency graph $\mathcal{G}^{inc}$ containing vertices v_m, corresponding to tuples $t_m \in T_j$. We consider adding vertex v_j to $\mathcal{G}_{j-1}^{inc}$ and appropriately update $|A_{j-1}|$ to obtain $|A_j|$. There are 2 cases:

- **Case 1:** When v_j is not connected with any unlabeled vertex (i.e., either v_j is all connecting with the labeled vertices $v_r \in \mathcal{G}_{j-1}^{inc}$ or v_j is disconnected, as illustrated in Figures 7(a) and 7(b), respectively), we let $|A_j| = |A_{j-1}|$ and v_j remains unlabeled. This is because according to factor-2 approximation algorithm, v_l are endpoints of edges in A_{j-1}, and after deleting vertices v_l, the edge $v_j v_l$ will also be removed. Thus, the size of edge set $|A_j|$ remains the same.
- **Case 2:** When v_j is connected with some unlabeled vertex $v_l \in \mathcal{G}_{j-1}^{inc}$ (as shown in Figure 7(c)), we let $|A_j| = |A_{j-1}| + 1$, and then both vertices v_j and v_l are labeled. This is because we add an edge $v_j v_l$ to A_{j-1}, and thus the size of A_j is given by $|A_{j-1}| + 1$.

5.2.3 PC-Topk Query Procedure

Next, we illustrate the PC-Top$k$ query procedure, namely PC-Top$k$_Processing, in Figure 8. We use τ to record the score threshold (line 1), which can be used for pruning mentioned in Lemma 5.1. We traverse the index $\mathcal{I}$ and retrieve tuples in non-increasing order of their preference values $f(\cdot)$ [35] (line 2). For the first k alternatives, we can calculate their score lower/upper

[3] *http://en.wikipedia.org/wiki/Vertex_cover*

Procedure PC-Top$k$_Processing {
Input: index $\mathcal{I}$ on a probabilistic database $\mathcal{D}$, query predicates $\mathcal{P}_T$: a preference function $f(\cdot)$, weight function $w(t_j, i)$, and integer k, and consistent score predicates $\mathcal{P}_S$
Output: the answer to the PC-Top$k$ query
(1) $\tau = -\infty$; $PC\text{-}Topk\_ans = \emptyset$; *// initialization*
(2) retrieve tuples t_j in non-increasing order of $f(t_j)$ from index $\mathcal{I}$ until $\theta_j \leq \tau$
(3) for the first k tuples t_j
(4) compute score lower/upper bounds $score_{lb}(t_j)$ and $score_{ub}(t_j)$ and set τ to the k-th largest score lower bound *// Inequality (21)*
(5) add t_j to $PC\text{-}Topk\_ans$
(6) for each of the subsequent tuples t_j
(7) compute score lower/upper bounds $score_{lb}(t_j)$ and $score_{ub}(t_j)$
(8) if $score_{ub}(t_j) > \tau$ *// via pruning rule, Lemma 5.1*
(9) $PC\text{-}Topk\_ans = PC\text{-}Topk\_ans \cup \{t_j\}$
(10) if $score_{lb}(t_j) > \tau$, then update the τ value
(11) refine candidates in $PC\text{-}Topk\_ans$ and return PC-Top$k$ answers
}

Figure 8: Procedure of PC-Top$k$ query answering.

bounds, set threshold τ to the k-th largest score lower bound, and add them to a candidate set $PC\text{-}Topk\_ans$ (lines 3-5). Next, for the subsequent tuples t_j that comes in, we compute their score lower/upper bounds similarly. If the score upper bound $score_{ub}(t_j)$ of t_j is not greater than threshold τ, then t_j is a false alarm and can be removed via our pruning rule (in Lemma 5.1); otherwise, it indicates that t_j is a candidate, and we add it to candidate set $PC\text{-}Topk\_ans$ (lines 6-9). Note that, in case the score lower bound $score_{lb}(t_j)$ of t_j is greater than τ, then we can update the τ value to further improve the pruning power (line 10). Finally, we refine the candidate set $PC\text{-}Topk\_ans$ and return the actual PC-Top$k$ answers (line 11).

Parameters		Settings
$[e_{min}, e_{max}]$	(PC-Range)	$[0, 0.04]$, $[0, 0.08]$, $[\mathbf{0}, \mathbf{0.12}]$, $[0, 0.16]$, $[0, 0.2]$
ε	(PC-Join)	$0.01, 0.02, 0.03, 0.04, 0.05$
k	(PC-Top$k$)	$100, 200, \mathbf{500}, 800, 1000$
γ		$0.04\%, 0.06\%, \mathbf{0.1}\%, 0.16\%, 0.2\%$
d		$\mathbf{2}, 3, 4, 5$
N		$10K, 20K, \mathbf{30K}, 40K, 50K$

Table 6: The experimental settings.

6. EXPERIMENTAL STUDY

In this section, we evaluate the performance of consistent query answering (CQA) in inconsistent probabilistic databases, including PC-Range, PC-Join, and PC-Top$k$ queries. Specifically, we test both real and synthetic data sets. For the real data, we use the International Ice Patrol, IIP, Iceberg Sighting Dataset[4] [27] from 1998 to 2000, which contains the locations ($latitude$, $longitude$) of the iceberg. Similar to [27], we assign each sighting record with a confidence (i.e., the existence probability) based on the source of sighting. That is, we transform sources R/V (radar and visual), VIS (visual only), RAD (radar only), LOW (low earth orbit satellite), MED (medium earth orbit satellite), HIGH (high earth orbit satellite), and EST (estimated) to (random) probability values within $[1, 0.9]$, $(0.9, 0.8]$, $(0.8, 0.7]$, $(0.7, 0.6]$, $(0.6, 0.5]$, $(0.5, 0.4]$, and $(0.4, 0.3]$, respectively. For synthetic data, we first generate alternatives in a probabilistic database with attribute values within $[0, 1]$ following either *Uniform* or *Zipf* (with skewness 0.8) distribution, and obtain data sets $uniform$ and $skew$, respectively. Then, for each alternative t_j, we randomly produce its existence probability $t_j.p$ within $[0, 0.9]$ and repair probability $t_j.rp$ in $[0, 1]$. Similar to [20], we simulate the inconsistency among tuples by randomly selecting pairs of tuples and treating them as inconsistent pairs. This way, a total number of $(\gamma \cdot N)$ tuples are connected to at least one edge in an inconsistency graph of the database. We use an R*-tree [5] with page size $4K$ to index each of the data sets above with pre-computed values (as mentioned in Section 5) to help fast CQA.

We measure the CQA performance in inconsistent probabilistic databases in terms of the *time cost*. Table 6 depicts the parameter settings of our experiments, where the numbers in bold font are *default values*. In each set of subsequent experiments, we will test one parameter while fixing the default values for others. All the experiments were run on a Pentium IV 3.2GHz PC with 1G memory. The experimental results are the average of 50 queries.

[4] *http://nsidc.org/data/g00807.html*

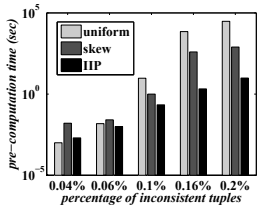
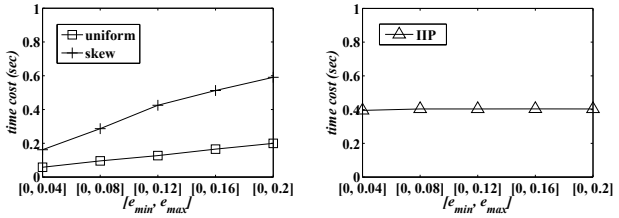


Figure 9: Pre-computation cost vs. percentage of inconsistent tuples.



(a) *time cost (synthetic data)* (b) *time cost (real data)*

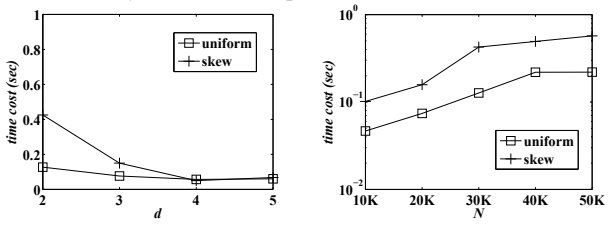
Figure 10: PC-Range vs. query range $[e_{min}, e_{max}]$.

6.1 PC-Range Performance

The first set of experiments test the performance PC-Range queries on both real and synthetic data. Recall that PC-Range returns those tuples satisfying both range predicates and consistent score predicates. In particular, the consistent score given by $t_j.P_{rpw}$ in Eq. (9) for each tuple t_j can be offline pre-computed. In turn, we need to calculate the weight $N_{\mathcal{RW}}(\cdot)$ in Eq. (10). Figure 9 illustrates the total pre-computation time of 3 data sets $uniform$, $skew$, and IIP with default parameter values. When the percentage of inconsistent tuples varies from 0.04% to 0.2%, the pre-computation time increases due to the higher recursive traversal cost in the inconsistency graph. Once the pre-computed data (e.g., $t_j.P_{rpw}$ and W_m) are obtained, we store them with tuples in leaf nodes of the index, which can be returned as consistent scores for PC-Range queries (or help score calculations for PC-Join or PC-Top$k$).

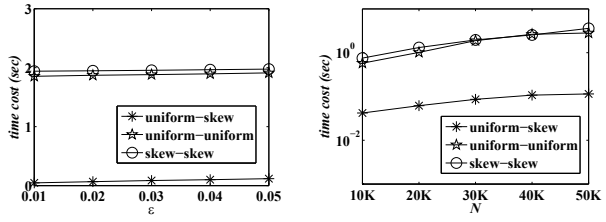
Figure 10 illustrates the time cost of PC-Range answering for query regions of different sizes, which are centered at locations following the data distribution, and with random query extent in $[e_{min}, e_{max}]$ on each dimension, where $[e_{min}, e_{max}]$ varies from $[0, 0.04]$ to $[0, 0.2]$, consistent score predicate $\alpha_R = 0$, and other parameters are set to default values. In figures, larger query region results in higher time cost, since more PC-Range answers (and their consistent scores) need to be accessed and returned (incurring more CPU time and I/O cost).

Moreover, for synthetic data $uniform$ and $skew$, we test the same set of experiments by varying dimensionality d and the data size N. The results are reported in Figures 11(a) and 11(b), respectively. When dimensionality d varies from 2 to 5, the time cost first decreases and then slightly increases. This is because for higher dimensionality d (i.e., more sparse data distribution), the number



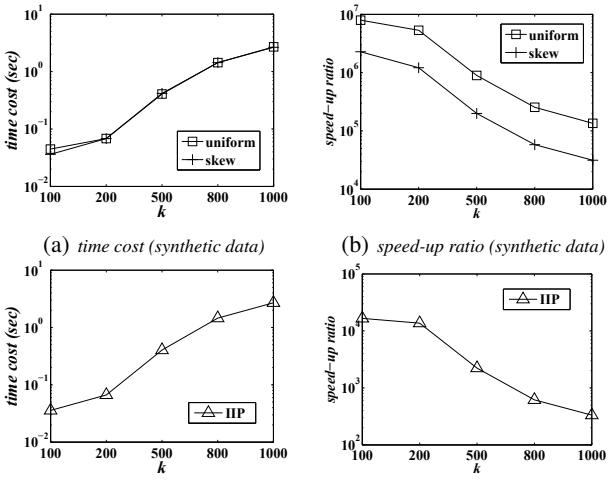
(a) *time cost vs. d* (b) *time cost vs. N*

Figure 11: PC-Range vs. d and N (synthetic data).



(a) *time cost vs.* ε (b) *time cost vs.* N

Figure 12: PC-Join vs. ε and N (synthetic data).



(a) *time cost (synthetic data)* (b) *speed-up ratio (synthetic data)*

(c) *time cost (real data)* (d) *speed-up ratio (real data)*

Figure 13: PC-Top$k$ vs. parameter k.

of the candidates to retrieve becomes smaller (thus, incurring lower time cost). On the other hand, large d will result in worse query performance on the R-tree index [36], which causes the time cost slightly increases for $d = 5$ compared with that for $d = 4$. Further, in Figure 11(b), the time cost smoothly increases with the increase of data size N, indicating good scalability of our approach.

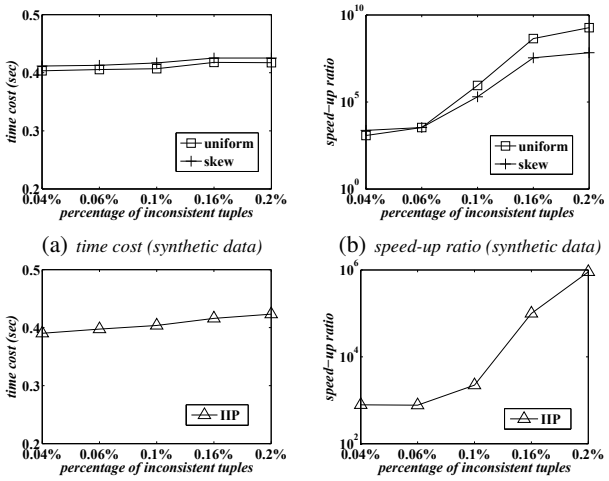
6.2 PC-Join Performance

Figure 12(a) evaluates the PC-Join query performance by varying the joining parameter, similarity threshold ε, from 0.01 to 0.05, on three combinations of data sets with the same size, $uniform$ $-skew$, $uniform$-$uniform$, and $skew$-$skew$, with default parameter settings, where consistent score predicate $\alpha_J = 0$. In particular, after obtaining candidate pairs from indexes, we return those pairs with consistent scores (given by Eq. (8)) above α_J. When threshold ε becomes larger, more PC-Join results will be retrieved, and thus higher time cost is required (nonetheless, below 2 seconds). Due to the data distributions, data set pairs $uniform$ $-uniform$ and $skew$-$skew$ have more joining tuples, thus resulting in higher time cost than $uniform$-$skew$.

Similarly, Figure 12(b) presents the same set of experiments on data sets with different data sizes N. The trends observed from figures are similar to that of PC-Range queries, and PC-Join can be processed efficiently in terms of the time cost. For other data set combinations, that with different α_J, or that with different sizes, the results are similar and thus omitted due to the space limitation.

6.3 PC-Top$k$ Performance

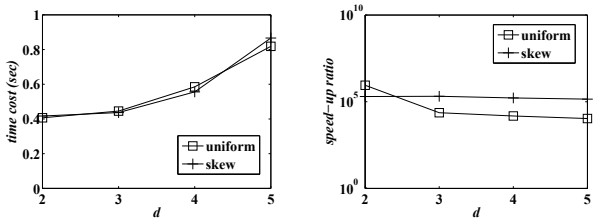
In this subsection, we present the query performance of PC-Top$k$ on both real and synthetic data. Note that, PC-Top$k$ queries can have different top-k semantics [27], by choosing various weight functions $w(t_j, i)$ in score function $\Upsilon_w(\mathcal{D}, t_j)$ (given by Eq. (4)). Due to the similar trend with different weight function, in the subsequent experiments, we will report the experimental results with



(a) *time cost (synthetic data)* (b) *speed-up ratio (synthetic data)*

(c) *time cost (real data)* (d) *speed-up ratio (real data)*

Figure 14: PC-Top$k$ vs. percentage of inconsistent tuples.



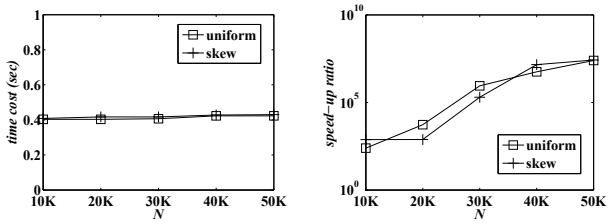
(a) *time cost (synthetic data)* (b) *speed-up ratio (synthetic data)*

Figure 15: PC-Top$k$ vs. dimensionality d.

the PT-k semantics [23, 27], where $w(t_j, i) = 1$ for $i \leq k$ and $w(t_j, i) = 0$ for $i > k$. For the preference function $f(\cdot)$, we test the linear function with random coefficients c_i within $(0, 1)$ on each dimension i (i.e., $f(x) = \sum_{i=1}^{d} c_i \cdot x[i]$). Nonetheless, our proposed approaches are applicable to any monotonic function $f(\cdot)$ and we will not present the similar results due to the space limitation. In the sequel, we will report the time cost of our PC-Top$k$ approach, which applies the early stopping condition (mentioned in Section 5.2.2). Moreover, we compare our approach with a baseline method, which sequentially scans the database and calculates the consistent score for each candidate in the database. We show the *speed-up ratio*, defined as the time cost of the baseline method divided by that of our PC-Top$k$ approach.

Figure 13 illustrates the experimental results of PC-Top$k$ with different k values from 100 to $1,000$ on synthetic and real data sets, where default values are used for other parameters. In figures, when the parameter k increases, the required time cost of our PC-Top$k$ approach also becomes higher. This is because with larger k, more candidates need to be retrieved from the database for the refinement. Nonetheless, the overall time cost remains small (below 1.63 seconds), which indicates the efficiency of our approach. Furthermore, compared with the baseline method, our PC-Top$k$ approach can save the time cost by more than 3-7 orders of magnitude, which indicates the effectiveness of our filtering techniques via score bounds as mentioned in Section 5.2.2.

Figure 14 studies the effect of the percentage of inconsistent tuples ($0.04\% \sim 0.2\%$) in the inconsistent probabilistic database on the PC-Top$k$ query performance, where other parameters are set to their default values. From the experimental results on both real and synthetic data sets, we find that the time cost of our PC-Top$k$ approach slightly increases with more inconsistent tuples, since the candidate refinement needs to compute consistent scores in a larger inconsistency graph, thus taking more time. Further, our approach



(a) *time cost (synthetic data)* (b) *speed-up ratio (synthetic data)*

Figure 16: PC-Top$k$ vs. data size N.

outperforms the baseline method by 3-9 (or 3-6) orders of magnitude for synthetic (real) data. For more inconsistent tuples, the speed-up ratio increases. This is because the baseline method has to traverse the graph with much higher cost for each candidate, whereas our PC-Top$k$ can utilize the effective filtering technique to prune candidates at a low cost. Thus, it indicates the good scalability of our approach against the percentage of inconsistent tuples.

Figure 15 varies the dimensionality, d, of $uniform$ and $skew$ data sets from 2 to 5, where default values are used for other parameters. In figures, the time cost of both data sets increases and speed-up ratio decreases for large d, since the candidate retrieval conducted in higher dimensional space is more costly through the index [36]. Nevertheless, the time cost is below 0.87 second, and the speed-up ratio compared with baseline remains high, that is, about 4-5 orders of magnitude.

Finally, Figure 16 reports the scalability about the PC-Top$k$ query performance against different sizes N of the database. Specifically, we vary N from $10K$ to $50K$ and set other parameters to their default values. The results show that time cost smoothly increases for large data size. Moreover, the speed-up ratio compared with the baseline method increases by around 3-7 orders of magnitude, which indicates the good scalability of our PC-Top$k$ approach.

7. CONCLUSION

This paper studies the important problem of consistent query answering (CQA) in inconsistent probabilistic databases. Specifically, we propose the notion of repair worlds using a graph representation, and reduce the CQA problem to the one in the repaired possible worlds (involving both repair and possible worlds). Under the all-possible-repairs semantics, we consider three probabilistic consistent queries, PC-Range, PC-Join, and PC-Top$k$, in inconsistent probabilistic databases. To enable fast CQA, we propose efficient approaches to process queries, including effective filtering methods to prune false positives. We demonstrate through extensive experiments the efficiency and effectiveness of our approaches.

Acknowledgments

Funding for this work was provided by Hong Kong RGC GRF Grant No. 611608 and NSFC Grant No. 60933011 and 60933012.

8. REFERENCES

[1] D. J. Abadi, S. Madden, and W. Lindner. REED: Robust, efficient filtering and event detection in sensor networks. In *VLDB*, 2005.
[2] P. Andritsos, A. Fuxman, and R. Miller. Clean answers over dirty databases: A probabilistic approach. In *ICDE*, 2006.
[3] L. Antova, C. Koch, and D. Olteanu. MayBMS: Managing incomplete information with probabilistic world-set decompositions. In *ICDE*, 2007.
[4] M. Arenas, L. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS*, 1999.
[5] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *SIGMOD*, 1990.
[6] O. Benjelloun, A. Das Sarma, A. Y. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In *VLDB*, 2006.
[7] G. Beskales, M. A. Soliman, I. F. Ilyas, and S. Ben-David. Modeling and querying possible repairs in duplicate detection. *PVLDB*, 2(1), 2009.
[8] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*, 2005.
[9] J. Boulos, N. N. Dalvi, B. Mandhani, S. Mathur, C. Ré, and D. Suciu. Mystiq: a system for finding more answers by using probabilities. In *SIGMOD*, 2005.
[10] R. Cheng, S. Singh, and S. Prabhakar. U-DBMS: A database system for managing constantly-evolving data. In *VLDB*, 2005.
[11] L. Chiticariu, P. G. Kolaitis, and L. Popa. Interactive generation of integrated schemas. In *SIGMOD*, 2008.
[12] J. Chomicki and J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.*, 197(1/2), 2005.
[13] G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving data quality: consistency and accuracy. In *VLDB*, 2007.
[14] G. Cormode, F. Li, and K. Yi. Semantics of ranking queries for probabilistic data and expected ranks. In *ICDE*, 2009.
[15] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDB J.*, 16(4), 2007.
[16] X. L. Dong, L. Berti-Equille, and D. Srivastava. Truth discovery and copying detection in a dynamic world. *PVLDB*, 2(1), 2009.
[17] X. L. Dong, A. Halevy, and C. Yu. Data integration with uncertainty. *The VLDB Journal*, 18(2), 2009.
[18] W. Fan. Dependencies revisited for improving data quality. In *PODS*, 2008.
[19] I. Fellegi and D. Holt. A systematic approach to automatic edit and imputation. *J. American Statistical Association*, 71(353), 1976.
[20] A. Fuxman, E. Fazli, and R. Miller. ConQuer: efficient management of inconsistent databases. In *SIGMOD*, 2005.
[21] S. Greco and C. Molinaro. Approximate probabilistic query answering over inconsistent databases. In *ER*, 2008.
[22] M. Hernandez, M. A. Hern Andez, S. Stolfo, and U. Fayyad. Real-world data is dirty: Data cleansing and the merge/purge problem. In *Data Min. Knowl. Discov.*, 1998.
[23] M. Hua, J. Pei, W. Zhang, and X. Lin. Ranking queries on uncertain data: a probabilistic threshold approach. In *SIGMOD*, 2008.
[24] Y. W. Huang, N. Jing, and E. A. Rundensteiner. Spatial joins using R-trees: breadth-first traversal with global optimizations. In *VLDB*, 1997.
[25] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. Jermaine, and P. J. Haas. Mcdb: a monte carlo approach to managing uncertain data. In *SIGMOD*, 2008.
[26] R. Kumar, K. Punera, T. Suel, and S. Vassilvitskii. Top-k aggregation using intersections of ranked inputs. In *WSDM*, 2009.
[27] J. Li, B. Saha, and A. Deshpande. A unified approach to ranking in probabilistic databases. *PVLDB*, 2(1), 2009.
[28] L. Mo, Y. He, Y. Liu, J. Zhao, S. Tang, X.-Y. Li, and G. Dai. Canopy closure estimates with greenorbs: Sustainable sensing in the forest. In *ACM Sensys*, 2009. http://greenorbs.org.
[29] T. Neumann, M. Bender, S. Michel, R. Schenkel, P. Triantafillou, and G. Weikum. Distributed top-k aggregation queries at large. *Distributed and Parallel Databases*, 26(1), 2009.
[30] R. A. Pottinger and P. A. Bernstein. Merging models based on given correspondences. In *VLDB*, 2003.
[31] C. Re, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, 2007.
[32] F. Sadri. Reliability of answers to queries in relational databases. *TKDE*, 3(2), 1991.
[33] A. D. Sarma, M. Theobald, and J. Widom. Exploiting lineage for confidence computation in uncertain and probabilistic databases. In *ICDE*, 2008.
[34] M. A. Soliman, I. F. Ilyas, and K. C. Chang. Top-k query processing in uncertain databases. In *ICDE*, 2007.
[35] Y. Tao, V. Hristidis, D. Papadias, and Y. Papakonstantinou. Branch-and-bound processing of ranked queries. *Inf. Syst.*, 32(3), 2007.
[36] Y. Theodoridis and T. Sellis. A model for the prediction of R-tree performance. In *PODS*, 1996.
[37] D. Z. Wang, E. Michelakis, M. Garofalakis, and J. Hellerstein. Bayestore: Managing large, uncertain data repositories with probabilistic graphical models. In *VLDB*, 2008.
[38] J. Wijsen. Database repairing using updates. *ACM Trans. Database Syst.*, 30(3), 2005.
[39] W. E. Winkler. Methods for evaluating and creating data quality. *Inf. Syst.*, 29(7), 2004.
[40] K. Yi, F. Li, G. Kollios, and D. Srivastava. Efficient processing of top-k queries in uncertain databases. In *ICDE*, 2008.