<table>
<thead>
<tr>
<th>Title</th>
<th>Cleaning uncertain data for top-k queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>Mo, L; Cheng, R; Li, X; Cheung, DWL; Yang, XS</td>
</tr>
<tr>
<td>Citation</td>
<td>The 29th International Conference on Data Engineering (ICDE 2013), Brisbane, Australia, 8-11 April 2013. In International Conference on Data Engineering Proceedings, 2013, p. 134-145</td>
</tr>
<tr>
<td>Issued Date</td>
<td>2013</td>
</tr>
<tr>
<td>URL</td>
<td><a href="http://hdl.handle.net/10722/189637">http://hdl.handle.net/10722/189637</a></td>
</tr>
</tbody>
</table>
Cleaning Uncertain Data for Top-\(k\) Queries

Luyi Mo, Reynold Cheng, Xiang Li, David W. Cheung, Xuan S. Yang

Department of Computer Science, University of Hong Kong, Pokfulam Road, Hong Kong
{lmyo, cckcheng, xli, dcheung, xyang2}@cs.hku.hk

Abstract—The information managed in emerging applications, such as sensor networks, location-based services, and data integration, is inherently imprecise. To handle data uncertainty, probabilistic databases have been recently developed. In this paper, we study how to quantify the ambiguity of answers returned by a probabilistic top-\(k\) query. We develop efficient algorithms to compute the quality of this query under the possible world semantics. We further address the cleaning of a probabilistic database, in order to improve top-\(k\) query quality. Cleaning involves the reduction of ambiguity associated with the database entities. For example, the uncertainty of a temperature value acquired from a sensor can be reduced, or cleaned, by requesting its newest value from the sensor. While this “cleaning operation” may produce a better query result, it may involve a cost and fail. We investigate the problem of selecting entities to be cleaned under a limited budget. Particularly, we propose an optimal solution and several heuristics. Experiments show that the greedy algorithm is efficient and close to optimal.

I. INTRODUCTION

The information handled in many emerging application is often uncertain. In the Global-Positioning System (GPS) [1] and natural habitat monitoring applications [2], data collected by sensors (e.g., location, humidity, and wind speed) may be stale and contaminated with measurement error. In data integration, record linkage tools may assign confidence values to data to indicate matching quality [3]. For example, in a movie rating system, whose information is integrated from Netflix challenge and IMDB, a user’s rating is a set of values with probabilistic guarantees [4]. Machine learning algorithms also generate segments of a given piece of text, the output of which can be incorrect [5]. To handle the increasing need of managing imprecise data, probabilistic databases have been recently developed (e.g., [6]–[9]).

Table I illustrates a probabilistic database, called udb1, where the uncertainty of an entity is captured by the \(x\)-tuple [6]. Here, an \(x\)-tuple represents current temperature values recorded by a sensor deployed in a geographical region. The tuples that exist in an \(x\)-tuple are mutually exclusive, and the existential probability (or Prob.) of a tuple is the chance that the reading is correct. For example, the reading of sensor \(S_1\) is \(21^\circ\text{C}\) with a probability of 0.6.

A probabilistic database enables the evaluation of a probabilistic query, which produces answers with statistical guarantees. Consider a probabilistic top-\(k\) query, returns information about objects that yield the \(k\) highest scores [10]–[15]. A Probabilistic Threshold top-\(k\) (or PT-\(k\)) query, for instance, returns tuples whose probabilities of being ranked \(k\)-th or higher are not smaller than some threshold \(T\) [11]. Let us suppose that in Table I, a tuple is given a higher rank if it has a higher temperature. If \(k = 2\) and \(T = 0.4\), then the answer of the PT-\(k\) query is \(\{t_1, t_2, t_3\}\), since the probabilities of these tuples that rank second or higher exceed 0.4.

In this paper, we study how to interpret the probabilistic result of a top-\(k\) query. Given the data uncertainty, a query answer is naturally inexact. In the previous example, we know that three tuples have probabilities larger than 0.4 for satisfying the PT-\(k\) query. However, are these the only tuples in the query answer? Is there any tuple that satisfies the query with a large probability (e.g., 0.39), but does not appear in the answer because \(T\) is set to 0.4? More generally, how should the ambiguity of a query answer be measured, so that we can know how much trust should be placed on an answer?

To address these issues, [16] proposed the PWS-quality metric for a probabilistic query. This score systematically quantifies the ambiguity of a query answer based on the Possible World Semantics (PWS) [3]. The PWS is a formal interpretation of a probabilistic database for supporting query evaluation. As illustrated in Figure 1(a) (Step 1), a probabilistic database can be viewed as a set of possible worlds. For instance, in Table I, a possible world \(W = \{t_0, t_3, t_4, t_6\}\) exists with probability \(0.6 \times 0.3 \times 0.4 \times 1 = 0.072\). A probabilistic query \(q\) can be conceptually answered by evaluating it on every possible world, yielding an answer known as \(\text{pw-result}\) (Step 2). For example, to answer a PT-2 query, we evaluate a deterministic top-2 query on every possible world. For \(W\), this top-2 query returns \(\text{pw-result}\) \(\{t_6, t_4\}\). In Step 3, the \(\text{pw-results}\) are aggregated according to the query semantics to form \(q\)’s answer. The PWS-quality measures the entropy of the occurrence of \(\text{pw-results}\) (Step A). Figure 2 shows all the \(\text{pw-results}\) of a PT-2 query on udb1; the PWS-quality of this query is \(−2.55\). Let us consider another database udb2 (Table II), where the \(x\)-tuple \(S_3\) of udb1 is modified. The PWS-quality of the PT-2 query for this database is \(−1.85\), which is higher than that of udb1. Figure 3 lists all the \(\text{pw-results}\) of udb2, the number of which is less than that of udb1. Intuitively, udb2 is

<table>
<thead>
<tr>
<th>Sensor ID</th>
<th>Tuple ID</th>
<th>Temp. (°C)</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>t0</td>
<td>21</td>
<td>0.6</td>
</tr>
<tr>
<td>S2</td>
<td>t2</td>
<td>30</td>
<td>0.7</td>
</tr>
<tr>
<td>S3</td>
<td>t4</td>
<td>25</td>
<td>0.4</td>
</tr>
<tr>
<td>S4</td>
<td>t6</td>
<td>26</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sensor ID</th>
<th>Tuple ID</th>
<th>Temp. (°C)</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>t0</td>
<td>21</td>
<td>0.6</td>
</tr>
<tr>
<td>S2</td>
<td>t2</td>
<td>30</td>
<td>0.7</td>
</tr>
<tr>
<td>S3</td>
<td>t4</td>
<td>25</td>
<td>0.4</td>
</tr>
<tr>
<td>S4</td>
<td>t6</td>
<td>26</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sensor ID</th>
<th>Tuple ID</th>
<th>Temp. (°C)</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>t0</td>
<td>21</td>
<td>0.6</td>
</tr>
<tr>
<td>S2</td>
<td>t2</td>
<td>30</td>
<td>0.7</td>
</tr>
<tr>
<td>S3</td>
<td>t4</td>
<td>25</td>
<td>0.4</td>
</tr>
<tr>
<td>S4</td>
<td>t6</td>
<td>26</td>
<td>1</td>
</tr>
</tbody>
</table>
less ambiguous than \textit{udb1}, yielding a higher quality score. \footnote{In the sequel, we use “quality” and “PWS-quality” interchangeably.}

Our first objective is to study the challenging problem of computing the quality score for a top-$k$ query. A naive solution, called \textit{PW}, computes the score by using the definition of the PWS-quality directly (Figure 1 Step A). Unfortunately, \textit{PW} is extremely expensive, since the number of possible worlds is exponentially large. We thus develop two efficient solutions to compute the quality of three common probabilistic top-$k$ queries, namely \textit{U-$k$Ranks} [10], \textit{PT-$k$} [11], and Global-top$k$ [12]. The first algorithm, called \textit{PWR}, computes the pw-results of these queries without performing Steps (1) and (2) (c.f. the dotted arrow in Figure 1). This saves the cost of handling possible worlds. However, \textit{PWR} depends on the number of pw-results, it may not work very well if the number of pw-results is large. Our second solution, called \textit{TP}, does not generate pw-results (Figure 1(b)). Instead, it uses tuple ranking information, computed by the \textit{PSR} algorithm \footnote{The \textit{PSR} algorithm was originally developed to evaluate probabilistic top-$k$ queries. We use it to compute PWS-quality here.}, to compute the PWS-quality efficiently (Step B). \footnote{The \textit{PSR} algorithm was originally developed to evaluate probabilistic top-$k$ queries. We use it to compute PWS-quality here.}

Another advantage of \textit{TP} is that it allows the computation effort of a top-$k$ query to be \textit{shared} by quality evaluation. As shown in Figure 1(b), the rank probability information used to obtain query answers can also be used to evaluate quality scores. Therefore, if a user wants to obtain a query answer and quality score at the same time, computing the score incurs little overhead. In some of our experimental results, the quality computation time is only 6\% of the query evaluation time.

Cleaning uncertain data. The use of the PWS-quality metric allows us to address an important issue: if a user is not satisfied with the quality of a query answer, can we \textit{improve its quality}? Intuitively, this can be done by removing, or \textit{cleaning} the uncertainty of the probabilistic database, so that less ambiguous results can be produced. The uncertainty of a movie rating, for instance, may be removed by asking the movie viewer to confirm her score. In the sensor monitoring application, it is possible to reduce data uncertainty by requesting (or \textit{probing}) the sensor to report the latest reading. Let us consider database \textit{udb1}, and suppose that after a probing operation, the reading of $S_3$ should be $27^\circ$C as indicated by tuple \textit{t5}. The new database, \textit{udb2} (Table II), shows that \textit{t4} was removed as a result. We say that $S_3$ is \textit{cleaned}. More importantly, this \textit{udb2} can yield a \textit{higher} quality than \textit{udb1}.

This is because the number of pw-results generated from \textit{udb2} is only four (Figure 3), which is smaller than the seven pw-results obtained from \textit{udb1} (Figure 2).

Ideally, all the x-tuples should be cleaned, so that we can attain the best query quality. However, this cleaning process is complicated by the following factors:

1. \textbf{Cost.} A cleaning operation, such as calling a movie viewer to confirm her rating, or querying a sensor about the latest reading, often involves a cost. This cost, in the movie rating system, may refer to the amount of money paid to officers to make phone calls to the movie viewers, or the airtime charge per minute. In the sensor monitoring system, it may be the battery power or the bandwidth to transfer a value from the sensor to the system.

2. \textbf{Limited budget.} The maximum amount of money allowed to make phone calls to movie viewers, or the maximum amount of resources that can be used to probe sensors, is often limited in practice. Hence, the number of cleaning operations that can be performed is constrained.

3. \textbf{Successfulness.} A cleaning operation may also fail to remove the uncertainty of an entity. The movie viewer may be unreachable at some time, and so the actual rating is not confirmed in this phone call. The transmission between two nodes in the network has some probability to fail due to the packet loss or other reliability problems [17]. Thus the system fails to get the latest reading from the sensor even a probing action is performed.

Under these conditions, we have to carefully control which x-tuples to be cleaned, in order to attain a better query quality. Moreover, the real value of the x-tuple is unknown before the cleaning operation is performed successfully. Hence, our second goal in this paper is to efficiently decide a set of x-tuples to be cleaned, and the number of cleaning operations performed on each selected x-tuple, in order to maximize the expected quality improvement. To address this problem, we first propose an efficient method to compute the expected quality improvement, given the selected x-tuples and the number of cleaning operations applied on each of them. We then design an optimal solution, as well as some fast heuristics, to tackle the cleaning problem.

We have performed detailed experiments on real and synthetic datasets. The results show that our approaches can efficiently compute the PWS-quality for three probabilistic top-$k$ queries. Among our cleaning algorithms, the greedy algorithm is efficient and gives a close-to-optimal quality.
improvement.

The rest of the paper is described as follows. Section II discusses the related work. Section III and IV introduces the probabilistic database model and how to compute PWS-Quality for top-\(k\) queries. Section V presents the cleaning problem and the solutions. Experimental results will be shown in Section VI. Section VII concludes.

II. RELATED WORK

The topic of probabilistic top-\(k\) queries has recently attracted a lot of research interest. Several semantics have been proposed, including U-Top\(k\) [10], U-\(k\)Ranks [10], PT-\(k\) [11], Global-top\(k\) [13], and Expected Rank queries [14]. In [18], a unified ranking framework was used to describe different variants of top-\(k\) queries. In [10]–[15], [19], researchers studied efficient evaluation algorithms for these queries.

We develop efficient quality computation algorithms for three well-studied top-\(k\) queries – U-\(k\)Ranks, PT-\(k\), and Global-top\(k\). These three queries share two common properties: (1) they conceptually evaluate a deterministic top-\(k\) query in every possible world (Figure 1(a) Step (2)); and (2) their query answers can be produced by rank probability information (Figure 1(b)). As we will explain, these two properties allow us to evaluate PWS-quality scores efficiently. We plan to study efficient quality computation algorithms for other top-\(k\) queries in the future.

Few works address the important issue of quantifying the ambiguity of answers produced by top-\(k\) queries. The closest work to ours is [16], which developed efficient algorithms to compute PWS-quality scores for range and max queries. The semantics of these queries are simple; both of them return a set of tuples and their probabilities of satisfying the queries. These query results are then used to derive the quality score [16]. However, that approach cannot be trivially extended to support top-\(k\) queries. First, a top-\(k\) query is more difficult to handle than range and max queries, even in deterministic databases. Second, as discussed before, a probabilistic top-\(k\) query possesses different semantics. In this paper, we study new solutions to compute the quality score of three common top-\(k\) queries.

In recent years, a number of researchers have studied the issues of cleaning uncertain data. In [20], user feedback was used to improve the the quality of a probabilistic database integrated from different sources. [21] analyzed the sensitivity of a probabilistic query answer to input data. In [16], the problem of cleaning a probabilistic database under a limited budget for maximizing the quality of range and max queries is studied. That paper assumes an idealistic model, where a cleaning operation always successfully removes ambiguous information about an object. In this paper, we study a more practical cleaning operation, where the result of cleaning is only successful with a probability. Moreover, we tackle this problem for top-\(k\) queries which, as discussed before, is more difficult to handle than range/max queries. A more sophisticated cleaning model is studied in [22], where the probability that a cleaning operation is successful is a probability distribution, and (2) the ambiguous information of an object may only be partially removed. It would be interesting to study how to extend our solutions to support this cleaning model. However, we remark that [22] neither handles probabilistic databases nor top-\(k\) queries.

We now summarize other works related to disambiguating a database. In sensor networks and data streams, researchers have studied how to request or probe data to reduce data ambiguity [23]–[25]. Data duplicate elimination techniques [26] can also be used to improve data quality. [27] used duplicate tuple merging techniques to provide possible answers. In [28], “possible repairs” are proposed to manage different versions of resulting databases due to duplicate removal. In [29], integration constraints were used to remove inconsistent data.

III. DATA MODELS AND TOP-\(k\) QUALITY

We now describe the probabilistic data model in Section III-A, and variants of probabilistic top-\(k\) query in Section III-B. We then review the PWS-quality in Section III-C and describe how it can be used to measure the top-\(k\) query quality.

A. Probabilistic Database Model

We adopt the x-tuple model [6] in this paper. Generally, a probabilistic database is denoted by \(D\), which contains \(m\) x-tuples where the \(l\)-th x-tuple is represented by \(\tau_l\). Each x-tuple is a subset of tuples \(t_i (i=1,\ldots,n)\), and is represented by \((ID_1,x_1,v_1,e_1)\). \(ID_1\) is the key of tuple \(t_1\). \(x_1 = \{l|l=1,\ldots,m\}\) indicates which x-tuple \(t_l\) belongs to. \(v_1\) is a set of attribute values of \(t_1\), and \(e_1\) is the existential probability of \(t_1\) (i.e., the probability that \(t_1\) exists in the real world). The existence of tuples in the same x-tuple is mutually exclusive, while the existence of tuples from different x-tuples is independent. Hence, the sum of \(e_1\) in the x-tuple \(\tau_l\), denoted by \(s_l\), is no larger than 1. If \(s_l\) is smaller than 1, we conceptually insert a “null” tuple to \(\tau_l\), whose existential probability equals to 1 – \(s_l\) and value equals to “null”. For example, in Table I, the “Temp.” and “Prob.” columns respectively correspond to the value and existential probability. There are 4 x-tuples, and in the first x-tuple, there are two tuples \(t_0\) and \(t_1\), i.e., \(\tau_1 = \{t_0,t_1\}\).

We also follow the PWS, where a possible world \(W\) contains exactly one tuple from each x-tuple in \(D\) (i.e., \(|W \cap \tau_l| = 1\)). The probability of a possible world \(W\) is the product of the existential probability of tuples in \(W\). Notice that \(\sum_{W \in W(D)} \Pr(W) = 1\).

B. Probabilistic Top-\(k\) Queries

In a deterministic database, the result of a top-\(k\) query contains tuples with the \(k\) highest ranks according to a ranking function \(f\). We assume that a tuple with value equals to “null” is always ranked lower than other tuples whose value is not “null”. We also assume that the ranking function assigns a unique rank for each tuple. This uniqueness can be obtained by employing any tie breaking criteria. Therefore, for two tuples \(t_1\) and \(t_2\), \(t_1 = f t_2\), if and only if \(t_1\) is identical to
of $p$-result.

denote probabilistic top-$k$ variations. To perform this query process, we leave them for future study. In the rest
of the paper, we restrict our discussion to these three variants.

Three commonly used probabilistic top-$k$ queries exactly follow this query process to produce query answers.

- **U-kRanks** [10]: for each rank $h (1 \leq h \leq k)$, return the tuple whose rank equals to $h$ has the highest probability among all tuples in the database.

- **PT-k** [11]: return the tuples with a top-$k$ probability not smaller than a given threshold.

- **Global-topk** [13]: tuples are sorted according to their top-$k$ probabilities, and only tuples with the $k$ highest top-$k$ probabilities are returned as the query answers.

Since other variants of top-$k$ query do not exactly follow this query process, we leave them for future study. In the rest of the paper, we restrict our discussion to these three variants of probabilistic top-$k$ query. In the sequel, we use “query” to denote probabilistic top-$k$ query. Next is the formal definition of pw-result.

**Definition 1:** The set of all pw-results of $Q$ on $D$, denoted by $\mathcal{R}(D, Q)$, is given by:

$$\mathcal{R}(D, Q) = \{r | \exists W \in \mathcal{W}(D) \text{ s.t. } r = Q^{k,f}(W)\},$$

where $Q^{k,f}(W)$ is the top-$k$ result according to the ranking function $f$ in possible world $W$. The probability of a pw-result $r \in \mathcal{R}(D, Q)$ is $Pr(r) = \sum_{W \in \mathcal{W}(D) \cap r = Q^{k,f}(W)} Pr(W)$.

In fact, a pw-result $r$ is an ordered list of $k$ tuples according to the ranking function, and $\sum_{r \in \mathcal{R}(D, Q)} Pr(r) = 1$. Consider $\mathit{udb1}$ in Table I, and a top-2 query on it. Suppose the ranking function assigns a higher rank to a tuple with a higher temperature reading. An example pw-result is $r = (t_1, t_2)$, since it is a top-2 result in $W_1 = \{t_1, t_2, t_4, t_6\}$ and $W_2 = \{t_1, t_2, t_5, t_8\}$. The probability of $r$ equals to $0.112 + 0.168 = 0.28$. In fact, $Pr(r)$ can be computed without examining all possible worlds.

**Lemma 1:** Given a pw-result $r$, let $r,t$ be the tuple which is ranked the lowest in $r$, the probability of $r$ is then given by

$$Pr(r) = \prod_{t \in r} e_t \prod_{t \in \mathcal{R}(D, Q) \cap r = \emptyset} (1 - \sum_{t \in \mathcal{R}(D, Q) \cap r = r,t} e_t).$$

**Proof:** The probability of $r$ is the product of: (1) the existential probabilities of tuples in $r$ (i.e., $\prod_{t \in r} e_t$), and (2) the probability that other tuples with higher rank than $r,t$ do not exist, otherwise $r$ is not a top-$k$ result in this possible world. The probability of (2) is $\prod_{t \in \mathcal{R}(D, Q) \cap r = \emptyset} (1 - \sum_{t \in \mathcal{R}(D, Q) \cap r = r,t} e_t)$.

We can obtain the rank-$h$ ($h = 1, \ldots, k$) probability of a tuple based on pw-results, and derive the top-k probability of a tuple accordingly.

**Definition 2:** The rank-$h$ probability ($h = 1, \ldots, k$) of $t_i$, denoted by $\rho_i(h)$, is the probability that $t_i$’s rank equals to $h$ in a pw-result.

**Definition 3:** The top-k probability of $t_i$, denoted by $p_i$, is the probability that $t_i$ is in top-k ranks in a pw-result, i.e.,

$$p_i = \sum_{h=1}^{k} \rho_i(h) = \sum_{t_i \in r \cap r \in \mathcal{R}(D, Q)} Pr(r).$$

**C. PWS-quality for Probabilistic Top-k Query**

PWS-quality is first proposed in [16]. It is defined as the negated value of the entropy of the pw-results. According to information theory, entropy can measure the uncertainty of information [30], which is used to quantify the ambiguity of query results in [16]. It’s shown that this quality metric is an effective and universal measure for query results derived following PWS. Thus, we adopt this quality metric to measure the uncertainty of probabilistic top-$k$ query result.

**Definition 4:** Given a probabilistic database $D$, a ranking function $f$, and a probabilistic top-$k$ query $Q$, the PWS-quality, denoted by $S(D, Q)$, is given by:

$$S(D, Q) = \sum_{r \in \mathcal{R}(D, Q)} Pr(r) \log Pr(r),$$

where the base of the log function is 2.

A higher entropy indicates the pw-results are more uncertain, and so PWS-quality is lower. PWS-quality achieves its maximum value zero when $\mathcal{R}(D, Q)$ contains a single result. If $|\mathcal{R}(D, Q)|$ is fixed, the minimum value is $\log(\frac{1}{|\mathcal{R}(D, Q)|})$ at which each pw-result has the same probability to exist.

To compute the quality score of a query, the naïve way, denoted by PW, derives all pw-results by evaluating a top-$k$ query in every possible world, and computes the quality score based on Definition 4. However, the number of possible worlds may be exponentially large. We next present two efficient algorithms for computing query quality score.

**IV. Top-k Quality Computation**

We present two efficient quality computation algorithms PWR and TP in Section IV-A and IV-B, respectively. We then discuss how to share the computation effort between query and quality evaluation (Section IV-C). Prior to the quality computation, we assume that tuples in $D$ are arranged in descending order of ranks.

**A. The PWR Algorithm**

We propose an algorithm, denoted by PWR, which directly derives all pw-results without expanding all possible worlds (c.f. the dotted arrow in Figure 1(a)), and then computes the quality score according to the Definition 4. Compared with PW, PWR avoids examining all possible worlds. Since a pw-result contains at most $k$ tuples in $D$, the number of pw-results is bounded by $n^k$. Hence, compared with PW, PWR can reduce the time complexity from exponential to polynomial to the size of $D$. Besides, PWR can avoid scanning the whole database. We next discuss the details of PWR.

Recall the query semantic in deterministic database, the result only contains tuples in top-$k$ ranks. If tuples are pre-sorted, the result is the first $k$ tuples. Thus, there is no need to distinguish tuples ranked lower than $k$. PWR makes use of this observation. We scan the tuples one by one and enumerate the existence of each tuple. Once it exists $k$ tuples, we get a
Algorithm 1 PWR

Require: Pre-sorted Probabilistic Database $D$, $k$
1: $\mathcal{R} \leftarrow \emptyset$, $r \leftarrow \emptyset$
2: DFS(1, $r$)
3: Compute PWS-quality score by Equation 4
4: return Scores of PWS-quality
5: procedure DFS($i$, $r$)
6: if $|r| = k$ then
7: $\mathcal{R} \leftarrow \mathcal{R} \cup \{r\}$, Compute Pr(r) by Equation 2
8: else if $\tau_x \cap r \neq \emptyset$ then
9: DFS($i+1$, $r$)
10: else if $\forall t \not\in \tau_x$, $t \not\in r$ then
11: DFS($i+1$, $r \cup \{t_i\}$)
12: else
13: DFS($i+1$, $r \cup \{t\}$), DFS($i+1$, $r$)

The number of pw-results is bounded by $n^k$, while finding a specific pw-result and computing its probability both require $O(n)$ time to scan the database. Therefore, the time complexity of PWR is $O(n^{k+1})$, which is polynomial to the size of $D$. However, the time complexity is exponential to the value of $k$, which makes this algorithm inefficient when $k$ is large. We next propose a more efficient algorithm to compute quality scores without enumerating all pw-results.

B. The TP Algorithm

Before we discuss the TP algorithm, we first present an important theorem which states that the PWS-quality for a top-$k$ query can be expressed by some probability of tuples’ existential probability and top-$k$ probability. We call it the tuple form expression of PWS-quality. For notational convenience, we use $Y(x)$ to denote the function of $x \log x$.

**Theorem 1**: The tuple form of PWS-quality is:

$$S(D, Q) = \sum_{t_i \in D} \omega_i \ p_{i},$$

where $\omega_i$ equals to

$$\log e_i + \frac{1}{e_i} (Y(1 - \sum_{x_i = t_i \land \lambda \geq t_i} e_{i'}) - Y(1 - \sum_{x_i = t_i \land \lambda > t_i} e_{i'})).$$

(6)

The derivation of Theorem 1 is outlined in Section IV-D. In fact, $\sum_{x_i = t_i \land \lambda \geq t_i} e_{i'}$ is the probability that tuples in $\tau_{x_i}$ ranked not less than $t_i$. And $\sum_{x_i = t_i \land \lambda > t_i} e_{i'}$ is the probability that tuples in $\tau_{x_i}$ ranked higher than $t_i$.

Theorem 1 illustrates that the quality score is essentially a weighted sum of tuples’ top-$k$ probabilities, and so we may avoid examining all pw-results in the quality evaluation. TP algorithm employs this theorem to compute the quality score. However, to directly use this theorem, it requires to compute the values of $p_i$ and $\omega_i$ first. We next show how $p_i$ and $\omega_i$ can be computed efficiently to complete the discussion of TP.

**Evaluation of $p_i$** To compute $p_i$, we adopt the PSR algorithm proposed in [15], which is the best approach currently known for calculating the rank-$h$ probability ($h = 1 \ldots k$) of all tuples in $D$. We slightly modify PSR in order to make it also return the top-$k$ probability of $t_i$, i.e., we additionally compute $p_i = \sum_{h=1}^k p_i(h)$. In fact, PSR also enables evaluation of $U$-kRanks, PT-k, and Global-topk. We will further discuss this in Section IV-C. The time complexity of PSR is $O(kn)$.

**Evaluation of $\omega_i$** To compute $\omega_i$ for $t_i$, the straightforward way is to directly employ Equation 6. However, it requires to examine all tuples $t_i'$ in $D$, which yields $O(n^2)$ time to compute all $\omega_i$s'. Observed that tuples are pre-sorted, and $\omega_i$ only depends on the existential probabilities of tuples which are ranked not lower than $t_i$ and in the same x-tuple with $t_i$, we can develop an incremental method to compute all $\omega_i$s'.

Let $E_{i,t} (1 \leq i \leq n, 1 \leq l \leq m)$ be the probability that tuples in $\tau_l$ ranked not lower than $t_i$, i.e.,

$$E_{i,t} = \sum_{t_i' \in \tau_l \land t_i' \geq t_i} e_{i'}.$$  

Thus, $\omega_i$ can be written as

$$\log e_i + \frac{1}{e_i} (Y(1 - E_{i,x_i}) - Y(1 - E_{i,x_i} + e_i)).$$

(8)

Since the tuples are pre-sorted, we can get the relation between $E_{i,t}$ and $E_{i-1,t}$ as follows:

$$E_{i,t} = \begin{cases} e_i + E_{i-1,t} & \text{if } l = x_i, \\ E_{i-1,t} & \text{otherwise.} \end{cases}$$

(9)

The boundary case is $E_{i,t} = 0$. Hence, we can incrementally compute all $E_{i,t}$ and $\omega_i$ in $O(n)$ time.

The efficiency of the evaluation of $\omega_i$ can be further improved by using the following fact. If $p_i = 0$, then $\omega_i = 0$. Hence, if we find that no other tuple has nonzero top-$k$ probability, we can stop the evaluation of $\omega_i$. The following lemma shows how to judge whether there is any tuple with nonzero top-$k$ probability.

**Lemma 2**: If $\{l | E_{i,l} = 1, 1 \leq l \leq m\} \geq k$, then for all $i' > i$, we have $p_{i'} = 0$.

**Proof**: The condition indicates that in any possible world there are at least $k$ tuples ranked higher than $t_i$, and so $t_i'$ has no chance to be in a pw-result.

With Lemma 2, we can stop the evaluation of $\omega_i$ once we find that no other tuple has nonzero top-$k$ probability.

To summarize, as shown in Figure 1(b), TP first employs PSR to obtain the values of $p_i$. It then calculates $\omega_i$ incrementally as discussed above, and get the quality score using its tuple form stated in Theorem 1 (Step B). The total time complexity of TP is $O(kn)$. 

138
C. Query Evaluation and Quality Computation

To obtain the query answer as well as the quality score, one can first run a query evaluation algorithm and then run one of the quality computation algorithms proposed in this paper. In this section, we show that TP enables computation effort sharing between query and quality evaluation, and so the total evaluation time of quality and query can be reduced. As discussed above, U-kRanks, PT-k and Global-top-k produce query answers based on the rank probability information. Using PSR, we can obtain the rank probability information efficiently. In fact, it has been shown that PSR can accelerate the evaluation process of these three queries [15]. For example, the query result of PT-k can be obtained by scanning the tuples with nonzero top-k probability once and output the ones whose top-k probability is not smaller than the threshold [15] (See [15] for more details).

Based on this observation, we can first employ PSR to compute the rank probabilities. This probability information is then used to derive the query answers according to the query semantics. Next we reuse this probability information to compute the PWS-quality. This information sharing is shown in Figure 1(b).

D. Proof Outline of Theorem 1

We outline the proof of Theorem 1 in this section. The detailed derivation can be found in our technical report [31].

By using Lemma 1 and the property that \( \log(ab) = \log a + \log b \), PWS-quality can be rewritten as:

\[
S(D,Q) = \sum_{r \in \mathcal{R}(D,Q)} \Pr(r) \sum_{t_i \in r} \log e_i + \sum_{r \in \mathcal{R}(D,Q)} \Pr(r) \sum_{t_i \in \mathcal{R}(D,Q)} \log (1 - \sum_{t_j \in \mathcal{R}(D,Q)} e_j). \tag{10}
\]

Recall the definition of top-k probability (Definition 3), \( p_i \) is obtained by summing up the pw-results’ probabilities (\( \Pr(r) \)'s). We can make use of this fact to replace all \( \Pr(r) \)'s by \( p_i \)'s in the first part of Equation 10.

For the second part of Equation 10, \( \sum_{t_i \in \mathcal{T}_i \cap d > r.t} e_i \) is essentially the probability that \( \mathcal{T}_i \) has at least one tuple ranked higher than \( r.t \). It can be obtained by summing up the existential probabilities of tuples in \( \mathcal{T}_i \) whose ranks are higher than \( r.t \). Thus,

\[
\sum_{t_i \in \mathcal{T}_i \cap d > r.t} e_i = \sum_{t_i \in \mathcal{T}_i \cap d > t(l.r.t)} e_i, \tag{11}
\]

where \( l(l,r,t) \) is the smallest tuple in \( \mathcal{T}_i \) whose rank is higher than \( r.t \). Then the second part of Equation 10 is equivalent to:

\[
\sum_{\mathcal{T}_i \in D} \sum_{t_i \in \mathcal{T}_i} \log (1 - \sum_{t_j \in \mathcal{T}_i \cap d > t_l(l,r,t)} e_j) \sum_{t_i \in \mathcal{T}_i \cap d > t_l(l,r,t)} \Pr(r). \tag{12}
\]

The summation of \( \Pr(r) \)'s in Equation 12 is the probability that a pw-result \( r \) contains no tuple \( t_i \) and the smallest tuple in \( \mathcal{T}_i \) whose rank is higher than \( r.t \) is tuple \( t_l \). This probability can be further replaced by the tuples’ top-k probabilities and existential probabilities in \( \mathcal{T}_i \) using the following facts.

First, the condition that \( t(l,r,t) = t_i \) can be rewritten as \( t_i > r.t \geq t_{\xi(t_i)} \), where \( \xi(t_i) \) is the index of the largest tuple in \( \mathcal{T}_i \) whose rank is lower than \( t_i \).

Second, let \( A_i \) be the event that (1) the rank of the smallest tuple in a pw-result is lower than \( t_i \), and (2) tuples in \( \mathcal{T}_i \) whose rank is not smaller than \( t_i \) does not appear. \( \sum_{t_{\xi(r) = \emptyset} \mathcal{A}_r \cap d > r.t} \Pr(r) \) equals to the probability that \( A_i \) is true (\( \Pr(A_i) \)). Also,

\[
p_i = e_i \Pr(A_i)/(1 - \sum_{t_{\xi(r) \cap d \geq t_i} e_i}). \tag{13}
\]

In fact, \( \Pr(A_i)/(1 - \sum_{t_{\xi(r) \cap d \geq t_i} e_i}) \) is the probability that for tuples not in \( \mathcal{T}_i \), at most \( k - 1 \) tuples whose ranks are higher than \( t_i \) exist. Hence, if \( t_i \) exists, \( t_i \) is in the pw-result.

Using these facts to replace all \( \Pr(r) \)'s in Equation 10 by tuples’ top-k probabilities and existential probabilities, we can derive the final result (Equation 5 and 6).

V. CLEANING UNCERTAIN DATA

We describe the cleaning model and the cleaning problem in Section V-A. We then discuss the efficient solutions in the rest of sections.

A. Cleaning Model and Problem Definition

As mentioned in Section I, after the user obtains the quality score, they may not be satisfied, which arouses the requirement of removing the uncertainty in the database, or clean the database. However, a cleaning operation, such as contacting the movie-viewer to confirm their rating or probing a sensor to obtain the latest reading, may fail and involve a cost.

To model the successfulness of a cleaning operation, we associate a successful cleaning probability or sc-probability to each x-tuple, which is the probability that an x-tuple is cleaned successfully. In the movie rating system, the sc-probability can be estimated by the historical statistics of the cleaning agent’s performance. In the sensor monitoring application, it can be the reliability of the data transmission between the sensor and the base station [32]. Let us first define the cleaning operation \( pclean(\mathcal{T}_i) \) with sc-probability.

**Definition 5:** Given an x-tuple \( \mathcal{T}_i \), and the sc-probability on this x-tuple \( P_i \), 0 \( \leq P_i \leq 1 \), the cleaning operation \( pclean(\mathcal{T}_i) \) is performed successfully with probability \( P_i \). If \( pclean(\mathcal{T}_i) \) is successful, \( \mathcal{T}_i \) is replaced by an x-tuple that contains a single tuple: \( \{ID_i, t_{v_i}, 1 \} \), such that \( ID_i \) and \( t_{v_i} \) are the corresponding identifier and value of some tuple \( t_i \) that belongs to \( \mathcal{T}_i \). If \( pclean(\mathcal{T}_i) \) fails, there is no change on \( \mathcal{T}_i \).

According to this definition, if \( pclean(\mathcal{T}_i) \) is performed, it has probability \( P_i \) to be successful. If \( pclean(\mathcal{T}_i) \) is successful, \( \mathcal{T}_i \) becomes “certain”. For example, in \( ud\delta \) (Table I), after \( pclean(S_{k3}) \) is successfully performed, \( S_{k3} \) in the new database, \( ud\delta \) (Table II), only contains a single tuple \( \{t_{5}, S_{k3}, 27°C, 1\} \), derived from \( t_{5} \), with the existential probability equals to 1.

We use \( c_1 \), a natural number, to model the cost of performing the cleaning operation \( pclean(\mathcal{T}_i) \) once. We assume that the current query \( Q \) is associated with a budget of \( C \) units, where \( C \) is a natural number and limits the maximum amount of resources that can be used to improve query quality.
Ideally, all x-tuples should be cleaned successfully. However, due to the cost and the limited budget, we should carefully select the set of x-tuples to clean. Moreover, since a cleaning operation may fail, we may need to clean an x-tuple for several times to increase its chance to be cleaned successfully. Hence, our goal is to select an appropriate set of x-tuples to clean, and determine the number of cleaning operations to be performed on the selected x-tuples such that the expected improvement in quality is maximized under a limited budget.

Formally, let X be any set of x-tuples chosen from the probabilistic database D. Without loss of generality, let X = \{τ_1, ..., τ_X\}. Let M = \{M_1, ..., M_X\}, where M_i represents the number of times pelean(τ_i) to be performed.

We use z_i to denote the set of all possible cleaned results of τ_i. Thus, z_i contains (|τ_i| + 1) x-tuples. Among them, one is τ_i, and others are new x-tuples which contain a single tuple derived from some tuple t_i that belongs to τ_i. Also, let \vec{x} be an “x-tuple vector” of |X| dimensions, which is the possible result after pelean(τ_i) is performed for M_i times for all τ_i ∈ X. We have \vec{x} ∈ z_1 × ... × z_X.

In Table 1, for example, if X = \{τ_1, τ_2\}, we have z_1 = \{τ_1, \{t_1\}, \{t_1\}\} and z_2 = \{τ_2, \{t_2\}, \{t_3\}\}. Since \vec{x} ∈ z_1 × z_2, there are 9 possible values of \vec{x}, including \{τ_1, \{t_2\}\}, \{\{t_0\}, \{t_3\}\}, and \{\{t_1\}, \{t_2\}\}.

The existential probability of a tuple in an x-tuple represents the probability that this tuple exists in the real world. Therefore, if t_i ∈ τ_i and pelean(τ_i) is successful, the probability that \vec{x}(l) = \{t_i\} = e_i.\footnote{\vec{x}(l) is the l-th element of \vec{x}, which is corresponding to τ_i.}

The new database after pelean(τ_i) is performed for M_i times for all τ_i ∈ X is denoted by D’ (i.e., in D’, τ_i is replaced by the corresponding x-tuple in \vec{x}). The expected quality of the cleaned database D’ is equal to:

\[
E(S(D’, Q)) = \sum_{\vec{x}0 \in z_1 \times ... \times z_X} Pr(\vec{x} = \vec{x}_0) \cdot S(D’, Q)
\]

Next we define the expected quality improvement of the cleaning, and the cleaning problem we study in this paper.

**Definition 6:** Given X = \{τ_1, ..., τ_X\}, and M = \{M_1, ..., M_X\}, the **expected quality improvement** after the cleaning is:

\[
I(X, M, D, Q) = E(S(D’, Q)) - S(D, Q)
\]

\[
\text{Definition 7: Given a budget of } C \text{ units, the Problem of cleaning uncertain data is to return a set of x-tuples } X \text{ from } D, \text{ and determine the number of cleaning operations to be performed on each selected x-tuple } M, \text{ such that } I(X, M, D, Q) \text{ is maximized, and the total cost of all cleaning operations does not exceed } C.\]

In practice, after we decide which x-tuples are selected to clean, and the number of cleaning operations to be performed on the selected x-tuples, we pass this information to the cleaning agent. The cleaning agent then uses this list of cleaning tasks and their frequencies to do cleaning. It is possible that an x-tuple is cleaned successfully before performing the assigned number of cleaning operations. In this case, the cleaning agent will not perform more cleaning operations on this x-tuple, and so some resources may be left. In this paper, we decide X and M to maximize the expected quality improvement before the cleaning. The interesting problem about how to update the list so that the rest of resources can be used to further improve the quality will be studied in future work.

We note that the total cost of cleaning τ_i for M_i times for all τ_i ∈ X is equal to \sum_{τ_i ∈ X} p_i M_i. A naive way to solve this problem is to enumerate all possible combinations of X and M, such that the total cost will not exceed the given budget, and find the one that can achieve the highest value of I(X, M, D, Q). However, this solution is inefficient in practice. The first reason is the computation of I(X, M, D, Q) may consume a lot of time if we calculate it by directly employing Equation 17, which requires to consider all possible x-tuple vectors. Another reason is, despite of M, the number of possible values of X may be exponentially large. We address these problems in Section V-B and V-C, and propose some efficient data cleaning algorithms in Section V-D.

**B. Evaluating Quality Improvement**

In this section, we discuss some principles about the expected quality improvement in order to build up the background knowledge for Section V-C and V-D. The main problem we investigate in this section is how to compute the expected quality improvement without evaluating the quality in every possible cleaned database.\footnote{The detailed proofs of lemmas and theorems used in this section can be found in our technical report [31].}

We first use e_i, \omega_i and p_i to denote the existential probability, weight and top-k probability of t_i in the cleaned database D’, respectively. If t_i is removed from D’ by the cleaning operation, we have e_i = \omega_i = p_i = 0.

We also let g(l, D) = \sum_{t_i \in \tau_i} \omega_i p_i, which is the weighted sum of top-k probabilities for tuples in τ_i and D. Thus, the quality of Q in D, i.e. S(D, Q), equals to \sum_{\tau_i \in D'} g(l, D). Accordingly, we let g(l, D’) = \sum_{t_i \in \tau_i} \omega_i p_i’ and S(D’, Q) = \sum_{\tau_i \in D'} g(l, D’).

In one possible cleaned database D’, we can compute t_i’s top-k probability in D’. We first study the relation between t_i’s expected top-k probability over all possible cleaned databases and t_i’s top-k probability in the original database D.
Lemma 3: If \( t_i \in \tau \) and \( \tau \) not in \( X \), the expected top-k probability of \( t_i \) over all possible cleaned databases, denoted by \( E(p'_l) \), equals to \( p_i \).

Proof: (Sketch) To evaluate \( E(p'_l) \) directly, we should examine all possible cleaned databases. Furthermore, to compute \( p'_l \) in \( D' \), it requires querying on every possible world. Thus, we can first aggregate the same possible world from different cleaned databases, and evaluate \( E(p'_l) \) on the aggregated possible worlds. We can further prove that the probability of a possible world over all possible cleaned databases is the same with the probability of this possible world in \( D \). Thus, the function of \( E(p'_l) \) is the same with \( p_i \).

In fact, Lemma 3 is used to support the following theorem.

Theorem 2: The expected quality improvement is given by:

\[
I(X, M, D, Q) = -\sum_{i=1}^{X} (1 - (1 - P)^{M_i}) g(l, D). \tag{19}
\]

Proof: (Sketch) Observed that \( E(S(D', Q)) \) equals to

\[
\sum_{i=1}^{X} E(g(l, D')) + \sum_{l=1}^{m} E(g(l, D)) \tag{20}
\]

We first claim that if \( 1 \leq l \leq X \), \( E(g(l, D')) = (1 - P)^{M_l} g(l, D) \). Consider two cases. (1) if \( \tau \) is cleaned, for all \( t_i \in \tau \), \( \omega' = 0 \), and so \( g(l, D') = 0 \); (2) if \( \tau \) is not cleaned, for all \( t_i \in \tau \), \( \omega' = \omega_i \). And the expected top-k probability of \( t_i \) over all possible cleaned databases where \( \tau \) is not cleaned equals to \( p_i \). Since the probability that \( \tau \) is not cleaned is \( (1 - P)^{M_l} \), we have \( E(g(l, D')) = (1 - P)^{M_l} \sum_{t_i \in \tau} \omega_i p_i = (1 - P)^{M_l} g(l, D) \).

We next claim that if \( X + 1 \leq l \leq m \), \( E(g(l, D')) = g(l, D) \). Since \( \tau \) is not selected to be cleaned, the existential probability of tuples in \( \tau \) keeps unchanged. Thus, for all \( t_i \in \tau \), we have \( \omega'_i = \omega_i \). Furthermore, \( E(\omega'_i p_i) = \omega_i E(p'_i) \), which is equivalent to \( \omega_i p_i \) according to Lemma 3.

By substituting these two facts into Equation 18, we get the final result (Equation 19).

Theorem 2 illustrates that when \( \omega_i \) and \( p_i \) have been computed during the quality evaluation, \( g(l, D) \) can be calculated in linear time, and so \( I(X, M, D, Q) \) can be computed in polynomial time. This computation method is much more efficient than the original definition which requires to examine all possible cleaned databases.

Because the value of the term \( (1 - (1 - P)^{M_i}) g(l, D) \) in Equation 19 equals to the value of \( I(X, M, D, Q) \) when \( X = \{\tau\} \), this term can be regarded as the expected quality improvement caused by performing \( pclean(\tau_1) \) for \( M_1 \) times. For convenience, we use \( G(l, D, j)(j \geq 0) \) to denote this term where \( M_1 \).

We then let \( b(l, D, j) \) be the amount of increment on the expected improvement if the number of times \( pclean(\tau_1) \) to be performed increases from \( j - 1 \) to \( j \). We have

\[
b(l, D, j) = G(l, D, j) - G(l, D, j - 1) = -((1 - P)^{j - 1} P) \cdot g(l, D). \tag{21}
\]

Furthermore, we assume that \( b(l, D, 0) = 0 \). With Equation 21, if \( g(l, D) \) is given, \( b(l, D, j) \) can be computed in \( O(1) \) time.

And \( I(X, M, D, Q) \) can be rewritten as the summation of \( b(l, D, j) \) as follows:

\[
I(X, M, D, Q) = \sum_{\tau_1 \in X} \sum_{j=1}^{M_1} b(l, D, j) \tag{22}
\]

Equation 22 can be used to derive the equivalent optimization problem to the cleaning problem. We will further discuss this in Section V-C.

Finally, we claim that \( b(l, D, j) \) has a monotonic property.

\[
\text{Lemma 4: } b(l, D, j) \text{ monotonically decreases with } j, \text{ which is the number of times } pclean(\tau_1) \text{ to be performed.}
\]

This nice property enables our efficient solutions proposed in the next two sections. Let us discuss how to make use of facts described in this section to solve our cleaning problem.

C. An Optimization Problem

We first state that not all x-tuples need to be considered in the cleaning process.

Lemma 5: For an x-tuple \( \tau_i \), if \( \forall t_i \in \tau \), \( p_i = 0 \), there is no need to clean \( \tau_i \).

Proof: Since \( g(l, D) = \sum_{t_i \in \tau} \omega_i p_i = 0 \), no matter how the cleaning operation is performed on this x-tuple, it has no effect on the expected quality improvement.

By using this fact, we can first exclude the x-tuples that cannot contribute to the expected quality improvement. We next reformulate the cleaning problem as an optimization problem.

We set \( Z \) be the set of all x-tuples whose \( g(l, D) \neq 0 \). Without loss of generality, let \( Z = \{\tau_1, \tau_2, \ldots, \tau_{|Z|}\} \). Let \( c_l \) be the cost to perform \( pclean(\tau_1) \) once, and \( C \) be the budget assign to the cleaning. Under the limited budget, we can derive that the number of times \( pclean(\tau_1) \) to be performed cannot exceed \( J_l = \lfloor \frac{C}{c_l} \rfloor \). We also assume that the value of \( g(l, D) \) has been obtained and stored in a lookup table, thus \( b(l, D, j) \) can be computed by Equation 21 in \( O(1) \) time. For notational convenience, we use \( b_{l,j} \) to represent \( b(l, D, j) \).

Consider the following optimization problem \( P(C, Z) \):

\[
\begin{align*}
\text{maximize} & \quad \sum_{l=1}^{X} \sum_{j=1}^{J_l} y_{l,j} \cdot b_{l,j} \\
\text{subject to} & \quad \sum_{l=1}^{X} \sum_{j=1}^{J_l} y_{l,j} \cdot c_l \leq C \\
& \quad y_{l,j} = 0 \text{ or } 1
\end{align*}
\tag{23}
\tag{24}
\tag{25}
\]

We claim that getting the optimal solution to \( P(C, Z) \) is equivalent to solving our cleaning problem (Definition 7).

\[
\text{Theorem 3: The optimal solution to the cleaning problem is given by:}
\]

\[
X^* = \{ \tau_i | \sum_{j=1}^{J_i} y^*_i, j > 0 \} \tag{26}
\]

\[
M^* = \{ M_i | \tau_i \in X^* \land M_i = \sum_{j=1}^{J_i} y^*_i \} \tag{27}
\]

where \( y^*_i \) is the optimal solution to \( P(C, Z) \).

Proof: We first prove that for an x-tuple \( \tau_i \in Z \), if \( \exists j_0 \) s.t. \( y^*_{i,j_0} = 0 \), then \( \forall j > j_0, y^*_{i,j} = 0 \).

Suppose by contradiction that for an x-tuple \( \tau_i \in Z \), \( \exists j_1 > j_0 \) and \( y_{i,j_1} = 1 \). Consider \( y_{l,j} = y^*_{i,j} \) except \( y_{l,j_1} = 1 \), \( y_{l,j_0} = 0 \). We have \( \sum_{j} y_{l,j} \cdot c_l = \sum_{j} y^*_{i,j} \cdot c_l \leq C \). Thus \( y^*_{i,j} \) is a feasible solution to \( P(C, Z) \). As discussed in Lemma 4,
$b_{i_0,j_0} \geq b_{i_0,j_1}$, thus, $\sum_j y_{i,j} \cdot b_{i,j} \geq \sum_j y_{i,j} \cdot b_{i,j}$, which violates the assumption that $y_{i,j}$ is the optimal solution to $P(C, Z)$.

Therefore, adding the constraint $y_{i,j+1} < y_{i,j}$ ($1 \leq l \leq |Z| \land 1 \leq j \leq J_l - 1$) into $P(C, Z)$ will not affect the optimal solution to $P(C, Z)$. Let $j_l$ be the largest index in $\gamma_l$ such that $y_{i,j_l} = 1$. We have $\sum_{j=1}^{j_l} y_{i,j} \cdot b_{i,j} = \sum_{j=1}^{j_l} b_{i,j}$. Besides, $\sum_{j=1}^{j_l} y_{i,j} \cdot c_l = j_l \cdot c_l$, which is the cost to perform $pe_{clean}(\gamma_l)$ for $j_l$ times. So the optimal solution of the cleaning optimization problem is equivalent to the optimal solution of $P(C, Z)$:

$$\text{maximize} \quad \sum_{l=1}^{|Z|} \sum_{j=1}^{j_l} b_{i,j}$$

subject to \quad $\sum_{l=1}^{|Z|} J_l \cdot c_l \leq C, j_l = 0, 1, \ldots, J_l$

Furthermore, this optimization problem is essentially the one corresponding to Def. 7, with $M_l = j_l$. We can then derive the optimal solution to the cleaning problem based on $y_{i,j}$, and obtain Equation 26 and 27.

D. Efficient Data Cleaning Algorithms

In this section, we first propose an optimal algorithm to the cleaning problem. We further develop several heuristics to improve the efficiency of the data cleaning. The implementation details and complexity analysis of all algorithms can be found in our technical report [31].

1) DP: This is a dynamic programming algorithm which can achieve the optimal solution to the cleaning problem. We note that $P(C, Z)$ is essentially the 0-1 knapsack problem in [33]. There are $N = \sum_{l=1}^{|Z|} J_l$ items in total, while each item has the value of $b_{i,j}$ and cost of $c_l$. The budget of the knapsack problem is $C$. This problem can be solved by a dynamic programming solution that runs in $O(N \times C) = O(C^2 |Z|)$ time. After finding the optimal solution to $P(C, Z)$, we can further derive the optimal solution to the cleaning problem according to Equation 26 and 27. Thus, the time complexity of $DP$ is $O(C^2 |Z|)$.

2) RandU: This is the simplest heuristic, where x-tuples are randomly selected to be cleaned with the same probability and with replacement until the budget is exhausted. This heuristic is designed based on the fairness principle, thus each cleaning operation gets a nearly fair chance to be selected to perform. The time complexity of this heuristic is $O(C)$.

3) RandP: This heuristic is similar to RandU. But the probability that an x-tuple is selected to be cleaned is derived based on the top-k probability of each x-tuple. We notice that $\sum_{i=1}^n p_i = k$, thus the probability that $\tau_l$ is selected to be cleaned equals to $\sum_{\tau_l \in \tau_l} p_i / k$. The selection process is the same as RandU’s, where x-tuples are randomly selected based on the derived probability and with replacement until the budget is exhausted. In fact, this heuristic is designed based on the intuition that cleaning an x-tuple with a larger top-k probability may have a better effect on the quality score, thus this x-tuple should have a higher probability to be cleaned. The time complexity of this heuristic is $O(C)$.

4) Greedy: Consider $N$ items described in V-D.1, let $\gamma_{i,j} = b_{i,j} / c_l$ be the score of an item with value of $b_{i,j}$ and cost of $c_l$. Conceptually, Greedy selects the item with the highest values of $\gamma_{i,j}$ such that the total cost does not exceed the budget. We denote the solution obtained by Greedy as the approximate solution to $P(C, Z)$. Greedy then employs Equation 26 and 27 to derive the solution to the cleaning problem based on this approximate solution. Intuitively, $\gamma_{i,j}$ is the expected quality improvement of $pe_{clean}(\tau_l)$ per unit cost. Thus, the choice of Greedy is decided by the amount of expected quality improvement and the cost required by each cleaning operation. By using the fact that $\gamma_{i,j+1} < \gamma_{i,j}$, and a heap to maintain items’ scores, the time complexity of this heuristic is $O(N \log |Z|) = O(C |Z| \log |Z|)$.

VI. EXPERIMENTS

We conducted an empirical study on both real datasets and some synthetic datasets. All our experiments are written in C++ and run on a machine with 8G memory, i5 CPU, 64-bit Linux system.

Synthetic Datasets. To follow the way in [16], we generate the synthetic dataset, we follow the way in [16]. The default dataset contains 5K x-tuples. Each x-tuple has 1D attribute $y$, in the domain $[0, 10000]$. $y$ has two components “uncertainty interval” $y.L$ and the “uncertainty pdf” $y.U$, where $y.U$ is a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$. The default parameter is $\sigma = 100$ and the mean value $\mu$ is uniformly distributed in the domain. The range of $y.L$ is uniformly distributed in [60, 100] and the center of “y.L” equal to $\mu$. Each x-tuple contains 10 tuples. We also discretize the $y.U$ by its histogram representation, where the existential probabilities are computed based on the “equal histogram bars” (the number of bars is 10), and values are the mean values of the histogram bars. Our default synthetic dataset thus has 5K x-tuples, and 50K tuples. In the sequel, when we mention the database size, it refers to the number of tuples in the database. Our ranking function $f$ gives a higher rank to a tuple with a larger value. For two tuples with the same value, the tuple with a smaller index is ranked higher.

Real Datasets (MOV). We also perform experiments on a real-world probabilistic dataset [4], which stores movie-viewer ratings from Netflix and synthetic uncertainty of the actual ratings. This dataset contains 4999 x-tuples, and each x-tuple consists of 2 tuples in average. Each tuple has five attributes: movie-id, viewer-id, date, rating, and confidence, where (movie-id, viewer-id) is the key for the corresponding x-tuple. The date and rating are the value attributes. The confidence represents the existential probability of this tuple. The date varies from 2000-01-01 to 2005-12-31, while the rating varies from 1 to 5. We normalized date and rating within the interval [0, 1] to give each attribute the same weight. The ranking function $f$ on this dataset gives a higher rank to a tuple with the larger score on date + rating. The top-k query on each possible world can be viewed as to find the top-k movie ratings with a high rating and recent date.

Top-k Queries. The tuples in all datasets are pre-sorted in the descending order of rank. For top-k queries, by default,
Effect of Uncertainty pdf. We also test the effect of the uncertainty pdfs of the attribute $y (y.U)$ in our datasets. Figure 4(b) presents the quality scores for five different pdfs. We test the effect of variance of the Gaussian distribution ($y.U$), where $G_X$ denotes that the variance is $X$, and also test a dataset where the uncertainty pdf is a uniform distribution. Observed that a Gaussian pdf with a smaller variance yields a higher quality score, and the uniform pdf has a lower quality score than other Gaussian pdfs. This shows that a uniform pdf leads to a more ambiguous result than a Gaussian pdf, and a Gaussian pdf with a larger variance renders a more ambiguous result than the one with a smaller variance.

Results on MOV. Figure 4(c) shows the quality score of MOV under different values of $k$. Similar to Figure 4(a), we observe a degradation in quality when $k$ increases. We also observe that although the number of x-tuples in MOV (4999) is similar to the number of x-tuples in the default synthetic dataset (5000), the quality scores in MOV is generally higher than those obtained in synthetic dataset. This is because each x-tuple in MOV has 2 tuples on average, while each x-tuple in synthetic dataset has 10 tuples. Thus, MOV is generally less ambiguous than the synthetic dataset, which may lead to a higher query quality.

B. Evaluation of Quality

Evaluation Time. To compare the efficiency of three quality computation algorithms, we first examine small datasets and with $k = 5$. We present the results in Figure 4(d). The amount of time required by $PW$ and $PWR$ increases with the database size, since more possible worlds (or pw-results) have to be considered. For $PW$, its runtime increases rapidly with the database size, and it takes 36.2 minutes to compute the quality even for a database with only 10 x-tuples. $PWR$ is much faster than $PW$, since the number of pw-results (e.g., $1.1 \times 10^3$ for database size = 100 and $k = 5$) is much fewer than the number of possible worlds (e.g., $10^{10}$ for database size = 100). On the other hand, $TP$ runs very fast since its runtime is linear to the database size and $k$.

We further evaluate the efficiency of $PWR$ and $TP$ in some large datasets. Figures 4(e) and 4(f) both show that if the database size is large, or the value of $k$ is large, $PWR$ cannot return the quality score in a reasonable time. In Figure 4(e), when $k = 15$, runtime of $PWR$ increases very fast since the number of pw-results increases rapidly. However, $TP$ needs much shorter time to evaluate the quality score than $PWR$. In the sequel, we use $TP$ to compute the quality.

Query vs. Quality Evaluation. We next evaluate the effect of probability information sharing discussed in Section IV-C. Figure 5(a) shows that with the probability information sharing, the total evaluation time of query and quality can be reduced to 52% of the time required by a solution without sharing at $k = 100$. Figure 5(b) further shows the time required to evaluate PT-$k$ and the extra time required to compute the quality when sharing is employed. Observed that when $k$ increases, the percentage of time spent on quality computation decreases. For example, the percentage is reduced from 33.3%
performance [34]. For two random algorithms $RandU$ and $RandP$, $RandP$ performs better since it gives priority to x-tuples with a higher top-$k$ probability, that may be beneficial to the quality improvement. $RandU$ does not consider any factor which may affect the quality improvement, and so it performs the worst among all cleaning algorithms.

**Effect of sc-pdf.** We also evaluate how the sc-pdf affects the cleaning effectiveness in Figure 6(b). In addition to the default uniform distribution, we also test three sc-pdfs that follow normal distributions with mean 0.5 and variance $\sigma = 0.13, 0.167,$ and 0.3, respectively. We observe that the effectiveness of $DP$ and $Greedy$ both increases with $\sigma$, and the effectiveness with a uniform sc-pdf is maximum. Conceptually, a larger variance indicates that there are more x-tuples have a large sc-probability. $DP$ and $Greedy$ take the sc-probability into consideration when determine the solution to the cleaning problem, and so they both benefit by a large variation of sc-pdf. However, $RandU$ and $RandP$ do not consider the sc-probability during the selection process. Since the average value of all sc-pdfs is identical, their effectiveness does not change a lot in different sc-pdfs.

**Effect of Average sc-probability.** Figure 6(c) shows how the change of the average sc-probability affects the cleaning effectiveness. We examine six sc-pdfs, each of them is a uniform distribution in the range of $[0, 1]$, thus its average value is $(1 + \varepsilon)/2$. The effectiveness of all cleaning algorithms increases with the average sc-probability, since the expected quality improvement will benefit by a larger sc-probability.

**Efficiency.** We present the time required by different algorithms to compute the solution to the cleaning problem in Figure 6(d). We see that $DP$ provides an optimal solution in polynomial time, but its runtime is much higher than other heuristics. $Greedy$ runs more slowly than $RandU$ and $RandP$ since it needs to maintain a heap which stores values of $\gamma_{l,j} = b(l, D, j)/c_l$. The overhead of $RandP$ compared to $RandU$ is caused by the selection criterion of $RandP$ which also considers the top-$k$ probability of each x-tuple.

We then study the effect of $k$ in Figure 6(e). We observe that runtime of $DP$ and $Greedy$ slightly increases with $k$. This is because, when $k$ increases, the value of $|Z|$ slightly increases, which will affect the efficiency of these two algorithms. For example, when $k = 15$, $|Z| = 79$. When $k$ is increased to 30, $|Z|$ becomes 98. For $RandU$ and $RandP$, the change of $k$ does not affect their efficiency.

**Results on MOV.** The results on MOV (shown in Figure 6(f) and 6(g)) are similar to those for the synthetic data. Observed that $Greedy$ always has a close-to-optimal performance, and it is the best among all heuristics.

**VII. Conclusions**

In this paper, we address the challenging problem of computing the PWS-quality score of a probabilistic top-$k$ query. We develop efficient algorithms to evaluate the quality of $U$-$k$Ranks, PT-$k$, and Global-top$k$ queries. We also investigate the problem of cleaning a probabilistic database to achieve an optimal gain of quality under a limited budget. In the
future, we will study how to efficiently compute the PWS-quality of other complex probabilistic queries. We will also examine other uncertain data cleaning problem, e.g., how to use minimal cost to attain a given quality score.

ACKNOWLEDGMENTS

Reynold Cheng, Luyi Mo, Xiang Li, and Xuan Yang were supported by the Research Grants Council of Hong Kong (GRF Project 711309E). We would like to thank the anonymous reviewers for their insightful comments.

REFERENCES


