HKU CS Tech Report TR-2012-11

Cleaning Uncertain Data for Top-k Queries

Luyi Mo, Reynold Cheng, Xiang Li, David Cheung, Xuan Yang

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

***Abstract*— The information managed in emerging applications, such as sensor networks, location-based services, and data integration, are 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 a 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°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 $\mathcal{T}$ [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 $\mathcal{T} = 0.4$, then the answer of the PT-k query is $\{t_1, t_2, t_5\}$, since the probabilities of these tuples that rank second or higher exceed 0.4.

TABLE I
Database *udb1*.

Sensor ID	Tuple ID	Temp. (°C)	Prob.
S_1	t_0	21	0.6
	t_1	32	0.4
S_2	t_2	30	0.7
	t_3	22	0.3
S_3	t_4	25	0.4
	t_5	27	0.6
S_4	t_6	26	1

TABLE II
Database *udb2*.

Sensor ID	Tuple ID	Temp. (°C)	Prob.
S_1	t_0	21	0.6
	t_1	32	0.4
S_2	t_2	30	0.7
	t_3	22	0.3
S_3	t_5	27	1
S_4	t_6	26	1

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 $\mathcal{T}$ 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 *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 pw-result (t_6, t_4). In Step 3, the pw-results are aggregated according to the query semantics to form q's answer. The PWS-quality measures the *entropy* of the occurrence of pw-results (Step A). Figure 2 shows all the 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 pw-results of *udb2*, the number of which is less than that of *udb1*. Intuitively, *udb2* is

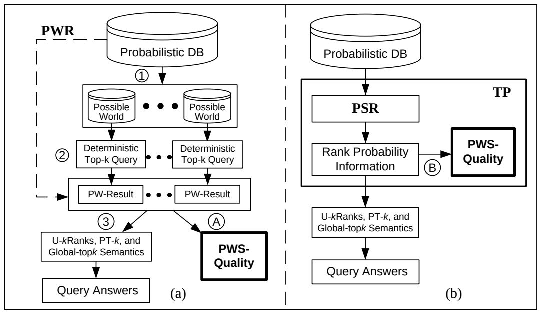

PWR
Probabilistic DB
①
Possible World
Possible World
Deterministic Top-k Query
Deterministic Top-k Query
②
PW-Result
PW-Result
③
Ⓐ
U-kRanks, PT-k, and Global-topk Semantics
PWS-Quality
Query Answers
(a)
Probabilistic DB
TP
PSR
Rank Probability Information
Ⓑ
PWS-Quality
U-kRanks, PT-k, and Global-topk Semantics
Query Answers
(b)

Fig. 1. PWS-Quality, PWR, and TP.

less ambiguous than *udb1*, yielding a higher quailty score. [1]

Our first objective is to study the challenging problem of computing the quality score for a top-k query. A naive solution, called **PW**, computes the score by using the definition of the PWS-quality directly (Figure 1 Step A). Unfortunately, **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 U-kRanks [10], PT-k [11], and Global-top$k$ [12]. The first algorithm, called **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, **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 **TP**, does not generate pw-results (Figure 1(b)). Instead, it uses tuple ranking information, computed by the **PSR** algorithm [15], to compute the PWS-quality efficiently (Step B). [2]

Another advantage of **TP** is that it allows the computation effort of a top-k query to be *shared* by quality evaluation. As shown in Figure 1(b), the rank probability information used to obtain query answers can also be 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 *improve its quality*? Intuitively, this can be done by removing, or *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 *probing*) the sensor to report the latest reading. Let us consider database *udb1*, and suppose that after a probing operation, the reading of S_3 should be 27°C as indicated by tuple t_5. The new database, *udb2* (Table II), shows that t_4 was removed as a result. We say that S_3 is *cleaned*. More importantly, this *udb2* can yield a *higher* quality than *udb1*.

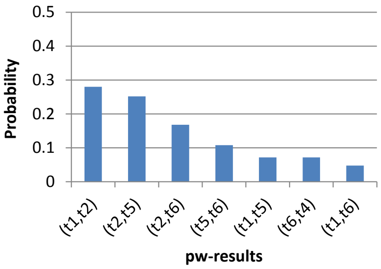


Fig. 2. *udb1* (quality = −2.55).

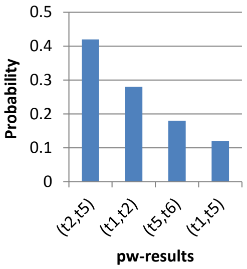


Fig. 3. *udb2* (quality = −1.85).

This is because the number of pw-results generated from *udb2* is only four (Figure 3), which is smaller than the seven pw-results obtained from *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. 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. 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 probing sensors, is often limited in practice. Hence, the number of cleaning operations that can be performed is constrained.

3. 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 may also have 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 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 the three probabilistic top-k queries that we studied. Among our cleaning algorithms, the greedy algorithm is an efficient solution and gives a close-

[1] In the sequel, we use "quality" and "PWS-quality" interchangeably.

[2] The **PSR** algorithm was originally developed to evaluate probabilistic top-k queries. We use it to compute PWS-quality here.

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-kRanks [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-kRanks, 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 (1) 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 τ_l. Each x-tuple is a subset of tuples $t_i (i = 1, 2, \ldots, n)$, and is represented by (ID_i, x_i, v_i, e_i). ID_i is the key of tuple t_i. $x_i = \{l | l = 1, \ldots, m\}$ indicates which x-tuple t_i belongs to. v_i is a set of attribute values of t_i, and e_i is the existential probability of t_i (i.e., the probability that t_i 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_i in the x-tuple τ_l, denoted by s_l, is no larger than 1. If s_l is smaller than 1, we conceptually insert a "null" tuple to τ_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 $\sum_{W \in \mathcal{W}(D)} \Pr(W) = 1$.

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" 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

t_2. Also $t_1 >_f t_2$ indicates that t_1's rank is higher than t_2's. For notational convenience, we omit f from $>_f, \geq_f$.

We now formulate the query process of the probabilistic top-k query we study in this paper. The probabilistic database is first expanded into a set of possible worlds. Then a deterministic top-k query is evaluated on every possible world to derive the pw-results. The pw-results are then aggregated by tuple and rank and produce the query answer based on the query semantics.

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-top$k$ [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**, denoted by $\mathcal{R}(D, Q)$, is given by:

$$\mathcal{R}(D, Q) = \{r \mid \exists W \in \mathcal{W}(D) \text{ s.t. } r = Q^{k,f}(W)\}, \quad (1)$$

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 $\Pr(r) = \sum_{W \in \mathcal{W}(D) \wedge r = Q^{k,f}(W)} \Pr(W)$.

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 *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 $W_2 = \{t_1, t_2, t_5, t_6\}$. 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

$$\Pr(r) = \prod_{t_i \in r} e_i \prod_{\tau_l \cap r = \emptyset} (1 - \sum_{t_i \in \tau_l \wedge t_i > r.t} e_i). \quad (2)$$

Proof: The probability of r is the product of: (1) the existential probabilities of tuples in r (i.e., $\prod_{t_i \in r} e_i$), and (2) the probability that other tuples ranked higher than $r.t$ does not exist, otherwise r is not a top-k result in this possible world. The probability of (2) is $\prod_{\tau_l \cap r = \emptyset} (1 - \sum_{t_i \in \tau_l, t_i > r.t} e_i)$. ■

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.,

$$p_i = \sum_{h=1}^{k} \rho_i(h) = \sum_{t_i \in r \wedge r \in \mathcal{R}(D,Q)} \Pr(r). \quad (3)$$

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]. 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, the **PWS-quality**, denoted by $S(D, Q)$, is given by:

$$S(D, Q) = \sum_{r \in \mathcal{R}(D,Q)} \Pr(r) \log \Pr(r), \quad (4)$$

where the base of the log function is 2.

A higher entropy indicates that the pw-results are more uncertain. Hence, the PWS-quality is lower. PWS-quality achieves its maximum value zero when $\mathcal{R}(D, Q)$ only contains a single result. If the size of $\mathcal{R}(D, Q)$ is fixed, the minimum value of PWS-quality is $\log 1/|\mathcal{R}(D,Q)|$ 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.

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 compute the quality score according to the definition (Step A). 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 the tuples in top-k ranks. If the tuples are pre-sorted, the query result is the first k tuples. Therefore, there is no need to distinguish the 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 we find that there are k tuples exist, we get a pw-result, and

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 $\tau_{x_i} \cap r \neq \emptyset$ then
 9:         DFS$(i+1, r)$
10:     else if $\forall t_{i'} \in \tau_{x_i}, t_{i'} > t_i$ then
11:         DFS$(i+1, r \cup \{t_i\})$
12:     else
13:         DFS$(i+1, r \cup \{t_i\})$, DFS$(i+1, r)$
```

we do not need to scan the rest of tuples. This process can be realized by a depth first search algorithm.

The details are shown in Algorithm 1. $\mathcal{R}$ is used to store all pw-results, and r is used to store all existing tuples currently. The parameter i in DFS represents the index of tuple we are considering. In DFS, we first check whether r is a pw-result, i.e., whether there are k tuples in r. If r is a pw-result, there is no need to consider the rest of tuples, and r should be inserted into $\mathcal{R}$ (Step 6); otherwise, we should enumerate whether t_i exists. However, we should consider two special cases first: (1) if there is another tuple in τ_{x_i} that exists in r, t_i will not exist, since tuples in the same x-tuple are mutually exclusive (Step 8); (2) if any other tuple in τ_{x_i} is ranked higher than t_i and does not exist in r, t_i will exist, since $|W \cap \tau_{x_i}| = 1$ (Step 10). If these two cases are not satisfied, we enumerate two possibilities of t_i: t_i exists or t_i does not exist (Step 12).

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 time 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 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 function of tuple's 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:

$$S(D, Q) = \textstyle\sum_{t_i \in D} \omega_i \; p_i, \tag{5}$$

where ω_i equals to

$$\log e_i + \frac{1}{e_i}\Big(Y\big(1 - \sum_{x_{i'} = x_i \wedge t_{i'} \geq t_i} e_{i'}\big) - Y\big(1 - \sum_{x_{i'} = x_i \wedge t_{i'} > t_i} e_{i'}\big)\Big). \tag{6}$$

In fact, $\sum_{x_{i'} = x_i \wedge t_{i'} \geq t_i} e_{i'}$ is the probability that tuples in τ_{x_i} ranked not less than t_i. And $\sum_{x_{i'} = x_i \wedge t_{i'} > t_i} e_{i'}$ is the probability that tuples in τ_{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 ω_i first. We next show how p_i and ω_i can be computed efficiently to complete the discussion of **TP**.

Evaluation of p_i**.** To compute p_i, we adopts 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 $p_i = \sum_{h=1}^{k} \rho_i(h)$. In fact, **PSR** also enables evaluation of the three top-k queries we consider in this paper. We will further discuss this in Section IV-C. The time complexity of **PSR** is $O(kn)$.

Evaluation of ω_i**.** To compute ω_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 ω_is. Observed that tuples are pre-sorted, and ω_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 ω_is.

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

$$E_{i,l} = \textstyle\sum_{t_{i'} \in \tau_l \wedge t_{i'} \geq t_i} e_{i'} = \sum_{x_{i'} = l \wedge i' \leq i} e_{i'}. \tag{7}$$

Thus, ω_i can be written as

$$\log e_i + \tfrac{1}{e_i}(Y(1 - E_{i,x_i}) - Y(1 - E_{i,x_i} + e_i))). \tag{8}$$

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

$$E_{i,l} = \begin{cases} e_i + E_{i-1,l} & \text{if } l = x_i, \\ E_{i-1,l} & \text{otherwise.} \end{cases} \tag{9}$$

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

The efficiency of the evaluation of ω_i can be further improved by using the following fact. If $p_i = 0$, then $\omega_i p_i = 0$. Hence, if we find that no other tuple has nonzero top-k probability, we can stop the evaluation of ω_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 $t_{i'}$, which indicates that $t_{i'}$ has no chance to be in a pw-result. ■

With Lemma 2, we can stop the evaluation of ω_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 ω_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 $O(kn)$.

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 query and quality 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 (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 of Theorem 1

We outline the proof of Theorem 1 in this section. The detailed derivation can be found in Appendix A.

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

$$\begin{aligned} S(D,Q) = & \sum_{r \in \mathcal{R}(D,Q)} \Pr(r) \sum_{t_i \in r} \log e_i \\ & + \sum_{r \in \mathcal{R}(D,Q)} \Pr(r) \sum_{\tau_l \cap r = \emptyset} \log(1 - \sum_{t_i \in \tau_l \wedge t_i > r.t} e_i). \end{aligned} \tag{10}$$

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_is' in the first part of Equation 10.

For the second part of Equation 10, $\sum_{t_i \in \tau_l \wedge t_i > r.t} e_i$ is essentially the probability that τ_l has at least a tuple ranked higher than $r.t$. It can be obtained by summing up the existential probabilities of tuples in τ_l whose ranks are higher than $r.t$. Thus,

$$\sum_{t_i \in \tau_l \wedge t_i > r.t} e_i = \sum_{t_i \in \tau_l \wedge t_i \geq t(l, r.t)} e_i, \tag{11}$$

where $t(l, r.t)$ is the smallest tuple in τ_l whose rank is higher than $r.t$. Then the second part of Equation 10 is equivalent to:

$$\sum_{\tau_l \in D} \sum_{t_i \in \tau_l} \log(1 - \sum_{t_{i'} \in \tau_l \wedge t_{i'} \geq t_i} e_{i'}) \sum_{\substack{\tau_l \cap r = \emptyset \wedge \\ t(l, r.t) = t_i}} \Pr(r). \tag{12}$$

The summation of $\Pr(r)$s' in Equation 12 is the probability that a pw-result r contains no tuple from τ_l and the smallest tuple in τ_l whose rank is higher than $r.t$ is tuple t_i. This probability can be further replaced by the tuples' top-k probabilities and existential probabilities in τ_l 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 τ_l 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 τ_l whose rank is not smaller than t_i does not appear. $\sum_{\tau_l \cap r = \emptyset \wedge t_i > r.t} \Pr(r)$ equals to the probability that A_i is true ($\Pr(A_i)$). Also,

$$p_i = e_i \, {}^{\Pr(A_i)}/_{(1 - \sum_{t_{i'} \in \tau_l \wedge t_{i'} \geq t_i} e_{i'})}. \tag{13}$$

In fact, ${}^{\Pr(A_i)}/_{(1 - \sum_{t_{i'} \in \tau_l \wedge t_{i'} \geq t_i} e_{i'})}$ is the probability that for tuples not in τ_l, there are 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 we study in this paper in Section V-A. We then discuss how to efficiently solve the cleaning problem in Section V-B, Section V-C and Section V-D.

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 [31]. Let us first define the cleaning operation $pclean(\tau_l)$ with sc-probability.

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

According to this definition, if $pclean(\tau_l)$ is performed, it has probability P_l to be successful. If $pclean(\tau_l)$ is successful, τ_l becomes "certain". For example, in $udb1$ (Table I), after $pclean(S_3)$ is successfully performed, S_3 in the new database, $udb2$ (Table II), only contains a single tuple $\{t_5, S_3, 27°\text{C}, 1\}$, derived from t_5, with the existential probability equals to 1.

We use c_l, a natural number, to model the *cost* of performing the cleaning operation $pclean(\tau_l)$ once. We assume that for the current query Q, it 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 $X = \{\tau_1, \ldots, \tau_{|X|}\}$. Let $M = \{M_1, \ldots, M_{|X|}\}$, where M_l represents the number of times $pclean(\tau_l)$ to be performed.

We use z_l to denote the set of all possible cleaned results of τ_l. Thus, z_l contains $(|\tau_l| + 1)$ x-tuples. Among them, one is τ_l, and others are new x-tuples which contain a single tuple derived from some tuple t_i that belongs to τ_l. Also, let $\vec{x}$ be an "x-tuple vector" of $|X|$ dimensions, which is the possible result after $pclean(\tau_l)$ is performed for M_l times for all $\tau_l \in X$. We have $\vec{x} \in z_1 \times \ldots \times z_{|X|}$.

In Table I (*udb1*), for example, if $X = \{\tau_1, \tau_2\}$, we have $z_1 = \{\tau_1, \{t_0\}, \{t_1\}\}$ and $z_2 = \{\tau_2, \{t_2\}, \{t_3\}\}$. Since $\vec{x} \in z_1 \times z_2$, there are 9 possible values of $\vec{x}$, including $(\tau_1, \{t_2\})$, $(\{t_0\}, \{t_3\})$, and (τ_1, τ_2).

The existential probability of a tuple in an x-tuple represents the probability that this tuple exists in the real world. Therefore, if $t_i \in \tau_l$ and $pclean(\tau_l)$ is successful, the probability that $\vec{x}(l) = \{t_i\}$ is e_i.[3] Consider the sc-probability P_l, and the number of times $pclean(\tau_l)$ to be performed M_l, we have

$$\Pr(\vec{x}(l) = \tau_l) \quad = \quad (1 - P_l)^{M_l} \tag{14}$$

$$\Pr(\vec{x}(l) = \{t_i\}, t_i \in \tau_l) \quad = \quad e_i(1 - (1 - P_l)^{M_l}) \tag{15}$$

Equation 14 is the probability that after $pclean(\tau_l)$ is performed for M_l times, uncertainty of τ_l still cannot be removed. Equation 15 is the probability that $pclean(\tau_l)$ is successful and the real tuple in τ_l is t_i.

We can further derive the probability of $\vec{x} = \vec{x}_0$, where $\vec{x}_0$ is one of the possible values of $\vec{x}$ (i.e., $\vec{x}_0 \in z_1 \times \ldots \times z_{|X|}$):

$$\Pr(\vec{x} = \vec{x}_0) = \textstyle\prod_{l=1}^{|X|} \Pr(\vec{x}(l) = \vec{x}_0(l)) \tag{16}$$

The new database after $pclean(\tau_l)$ is performed for M_l times for all $\tau_l \in X$ is denoted by D' (i.e., in D', τ_l is replaced by the corresponding x-tuple in $\vec{x}$). The expected quality of the cleaned database D' is equal to:

$$E(S(D', Q)) = \sum_{\vec{x}_0 \in z_1 \times \ldots \times z_{|X|}} \Pr(\vec{x} = \vec{x}_0) \cdot S(D', Q) \tag{17}$$

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

Definition 6: Given $X = \{\tau_1, \ldots, \tau_{|X|}\}$, and $M = \{M_1, \ldots, M_{|X|}\}$, the **expected quality improvement** after the cleaning is:

$$I(X, M, D, Q) = E(S(D', Q)) - S(D, Q) \tag{18}$$

Definition 7: Given a budget of C units, the **Problem of cleaning uncertain data** is to return a set of x-tuples X from D, and determine the number of cleaning operations to be performed on each selected x-tuple M, such that $I(X, M, D, Q)$ 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_l for M_l times for all $\tau_l \in X$ is equal to $\sum_{\tau_l \in X} c_l M_l$. A naïve 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.[4]

We first use e'_i, ω'_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 $g(l, D) = \sum_{t_i \in \tau_l} \omega_i p_i$, which is the weighted sum of top-k probabilities for tuples in τ_l and D. Thus, the quality of Q in D, i.e. $S(D, Q)$, equals to $\sum_{\tau_l \in D} g(l, D)$. Accordingly, we let $g(l, D') = \sum_{t_i \in \tau_l} \omega'_i p'_i$ and $S(D', Q) = \sum_{\tau_l \in D} g(l, D')$.

[3] $\vec{x}(l)$ is the l-th element of $\vec{x}$, which is corresponding to τ_l.

[4] The detailed proofs of lemmas and theorem in this section can be found in Appendix B to 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: Given $X = \{\tau_1, \ldots, \tau_{|X|}\}$, and $M = \{M_1, \ldots, M_{|X|}\}$. If $t_i \in \tau_l$ $(l > |X|)$, the expected top-k probability of t_i over all possible cleaned databases, denoted by $E(p'_i)$, equals to p_i.

Proof: **(Sketch)** To evaluate $E(p'_i)$ directly, we should examine all possible cleaned databases. Furthermore, to compute p'_i in D', it requires querying on every possible world. Thus, we can first aggregate the same possible world from different cleaned databases, and evaluate $E(p'_i)$ 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 $E(p'_i)$ is the same with p_i. ■

In fact, Lemma 3 is used to support the following theorem about expected quality improvement.

Theorem 2: Given $X = \{\tau_1, \ldots, \tau_{|X|}\}$, and $M = \{M_1, \ldots, M_{|X|}\}$, the expected quality improvement is:

$$I(X, M, D, Q) = -\sum_{l=1}^{|X|} (1 - (1 - P_l)^{M_l}) g(l, D). \quad (19)$$

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

$$\sum_{l=1}^{|X|} E(g(l, D')) + \sum_{l=|X|+1}^{m} E(g(l, D')) \quad (20)$$

We first claim that if $1 \leq l \leq |X|$, $E(g(l, D')) = (1 - P_l)^{M_l} g(l, D)$. Consider two cases. (1) if τ_l is cleaned, for all $t_i \in \tau_l$, $\omega'_i = 0$, and so $g(l, D') = 0$; (2) if τ_l is not cleaned, for all $t_i \in \tau_l$, $\omega'_i = \omega_i$. And the the expected top-k probability of t_i over all possible cleaned databases where τ_l is not cleaned, is equivalent to the expected top-k probability of t_i on $\tilde{X}$ and $\tilde{M}$ where $\tilde{X} = X - \{\tau_l\}$, $\tilde{M} = M - \{M_l\}$. This expected top-k probability equals to p_i according to Lemma 3. Since the probability that τ_l is not cleaned is $(1 - P_l)^{M_l}$, we have $E(g(l, D')) = (1 - P_l)^{M_l} \sum_{t_i \in \tau_l} \omega_i p_i = (1 - P_l)^{M_l} g(l, D)$.

We next claim that if $|X| + 1 \leq l \leq m$, $E(g(l, D')) = g(l, D)$. Since τ_l is not selected to be cleaned, the existential probability of tuples in τ_l keeps unchanged. Thus, for all $t_i \in \tau_l$, 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 ω_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 $(1 - (1 - P_l)^{M_l}) g(l, D)$ in Equation 19 equals to the value of $I(X, M, D, Q)$ when $X = \{\tau_l\}$, this term can be regarded as the expected quality improvement caused by performing $pclean(\tau_l)$ for M_l times. For convenience, we use $G(l, D, j) (j \geq 0)$ to denote this term where $j = M_l$.

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

$$\begin{aligned} b(l, D, j) &= G(l, D, j) - G(l, D, j-1) \\ &= -(1 - P_l)^{j-1} P_l \cdot g(l, D). \end{aligned} \quad (21)$$

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:

$$I(X, M, D, Q) = \sum_{\tau_l \in X} \sum_{j=1}^{M_l} b(l, D, j) \quad (22)$$

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 claims that $b(l, D, j)$ has a monotonic property.

Lemma 4: $b(l, D, j)$ monotonically decreases with j, which is the number of times $pclean(\tau_l)$ to be performed.

Proof: **(Sketch)** We first prove that $g(l, D)$ is non-positive by recalling the proof of Theorem 1. Hence, $b(l, D, j)$ is non-negative, and $b(l, D, j+1) = (1 - P_l) b(l, D, j) \leq b(l, D, j)$. ■

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 τ_l, if $\forall t_i \in \tau_l$, $p_i = 0$, there is no need to clean τ_l.

Proof: Since $g(l, D) = \sum_{t_i \in \tau_l} \omega_i p_i = 0$, 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 let 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 $pclean(\tau_l)$ once, and C be the budget assign to the cleaning. Under the limited budget, we can derive that the number of times $pclean(\tau_l)$ to be performed cannot exceed $J_l = \lfloor \frac{C}{c_l} \rfloor$. We also assume that the value of $g(l, D)$ have 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 $P(C, Z)$:

$$\begin{aligned} \text{maximize} \quad & \sum_{l=1}^{|Z|} \sum_{j=1}^{J_l} y_{l,j} \cdot b_{l,j} & (23) \\ \text{subject to} \quad & \sum_{l=1}^{|Z|} \sum_{j=1}^{J_l} y_{l,j} \cdot c_l \leq C & (24) \\ & y_{l,j} = 0 \text{ or } 1 & (25) \end{aligned}$$

We claim that $P(C, Z)$ is equivalent to our cleaning problem (Definition 7).

Theorem 3: The optimal solution to the cleaning problem is given by:

$$X^* = \{\tau_l | \sum_{j=1}^{J_l} y^*_{l,j} > 0\} \quad (26)$$

$$M^* = \{M_l | \tau_l \in X^* \wedge M_l = \sum_{j=1}^{J_l} y^*_{l,j}\} \quad (27)$$

where $y^*_{l,j}$ is the optimal solution to $P(C, Z)$.

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

Suppose by contradiction that for an x-tuple τ_{l_0}, $\exists j_1 > j_0$ and $y^*_{l_0,j_1} = 1$. Consider $y'_{l,j} = y^*_{l,j}$, except $y'_{l_0,j_0} = 1$, $y'_{l_0,j_1} = 0$. We have $\sum_l \sum_j y'_{l,j} \cdot c_l = \sum_l \sum_j y^*_{l,j} \cdot c_l \leq C$. Thus $y'_{l,j}$ is a feasible solution to $P(C, Z)$. As discussed in Lemma 4, $b_{l_0,j_0} \geq b_{l_0,j_1}$, thus, $\sum_l \sum_j y'_{l,j} \cdot b_{l,j} \geq \sum_l \sum_j y^*_{l,j} \cdot b_{l,j}$, which violates the assumption that $y^*_{l,j}$ is the optimal solution to $P(C, Z)$.

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

$$\begin{array}{ll} \text{maximize} & \sum_{l=1}^{|Z|} \sum_{j=1}^{j_l} b_{l,j} \\ \text{subject to} & \sum_{l=1}^{|Z|} j_l \cdot c_l \leq C, j_l = 0, 1, \ldots, J_l \end{array}$$

Furthermore, this optimization problem is essentially the one corresponding to Definition 7, with $M_l = j_l$. Thus, we obtain the equivalence of $P(C, Z)$ and our cleaning problem. We can then derive the optimal solution to the cleaning problem based on $y^*_{l,j}$, 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 Appendix E to G.

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 [32]. There are $N = \sum_{l=1}^{|Z|} J_l$ items in total, while each item has the value of $b_{l,j}$ 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 τ_l is selected to be cleaned equals to $\sum_{t_i \in \tau_l} p_i / k$. The selection process is the same as *RandU*'s, where x-tuples are randomly selected base 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 $\gamma_{l,j} = b_{l,j} / c_l$ be the score of an item with value of $b_{l,j}$ and cost of c_l. Conceptually, $Greedy$ selects the item with the highest values of $\gamma_{l,j}$ 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, $\gamma_{l,j}$ is the expected quality improvement of $pclean(\tau_l)$ 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 $\gamma_{l,j+1} \leq \gamma_{l,j}$, 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 data set 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 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 $\aleph(\mu, \sigma^2)$. The default parameter is $\sigma = 100$ and the mean value μ 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 μ. 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 always 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 the other one.

Real Datasets (MOV). We also perform experiments on a real-world probabilistic dataset [4], which stores the uncertainty about movie-viewer 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 large date, that are the newest.

Top-k Queries. The tuples in all datasets are pre-sorted in the descending order of rank. For top-k queries, by default, $k = 15$, and the threshold for PT-k is 0.1. To evaluate the query answers, we implement **PSR** and adopt the evaluation framework proposed in [15].

Quality Score. To compute the quality score, we employ **TP** by default. We have verified the correctness of **PWR** and **TP** by comparing with **PW** in several experiments under different settings. We found that the absolute difference between the quality scores calculated by different methods is always smaller than 10^{-8}. The slight difference is caused by the precision loss during the computing process.

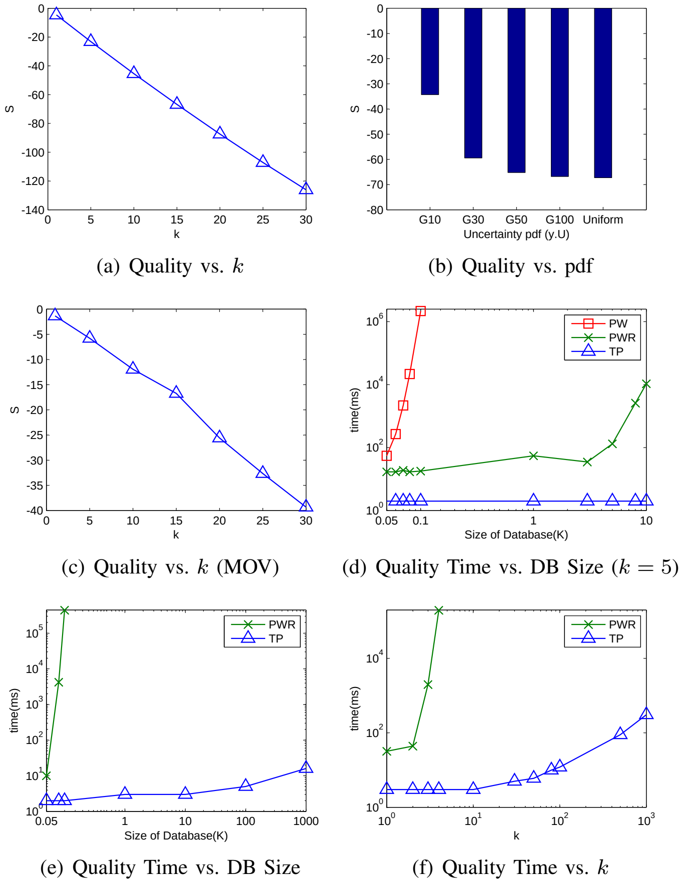
Cleaning Problem. To model the cleaning problem, for both datasets, we first generate a cleaning cost for each x-tuple, which is an integer and uniformly distributed in the range of $[1, 10]$. We also attach a successful cleaning probability to each x-tuple. The sc-probability is generated from a *sc-probability distribution function*, or *sc-pdf* for short. By default, the sc-pdf is a uniform distribution in the range of $[0, 1]$. The default cleaning budget C is 100 units.

A. Quality of Top-k Queries

Effect of k. We next investigate how the value of k affects the quality score of the query results. Figure 4(a) shows that when k increases the quality score decreases. This is because the number of pw-results increases with k, and the query results are more uncertain. Thus, the quality score reflects the ambiguity of the query results.

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 GX 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 yeilds 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.



(a) Quality vs. k

(b) Quality vs. pdf

(c) Quality vs. k (MOV)

(d) Quality Time vs. DB Size ($k = 5$)

(e) Quality Time vs. DB Size

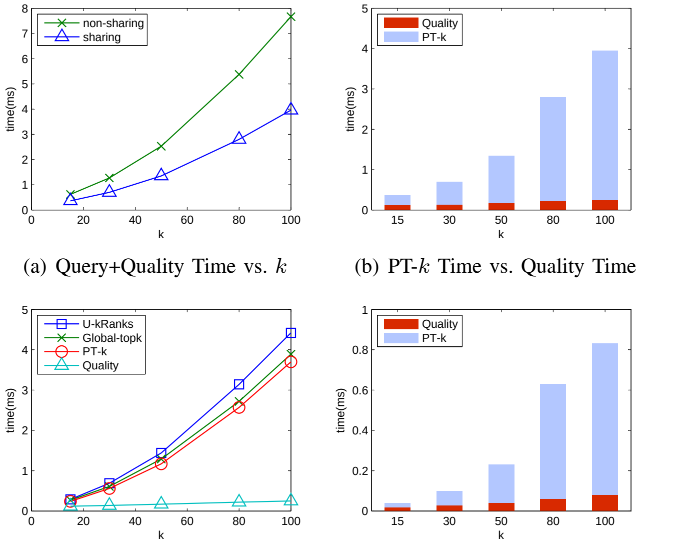
(f) Quality Time vs. k

Fig. 4. Effectiveness and Efficiency of Query Quality Computation.

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., 1.1×10^5 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



(a) Query+Quality Time vs. k

(b) PT-k Time vs. Quality Time

(c) Query Time vs. Quality Time

(d) PT-k Time vs. Quality Time (MOV)

Fig. 5. Effect of Sharing.

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% at $k = 15$ to 6.3% at $k = 100$. We also test the evaluation time of U-kRanks and Global-top$k$, and shows the result in Figure 5(c). Generally, the evaluation time of these two queries is slightly more than PT-k, and so the extra time to compute quality only occupies a small percentage of the total evaluation time. The results for varying the database size are similar, and so they are skipped here.

Results on MOV. As shown in Figure 5(d), the overhead for quality computation in addition to query evaluation is also not too much in MOV. However, we observe that the total time for query and quality evaluation in MOV is less than the one in synthetic dataset. For example, when $k = 15$, the total evaluation time in MOV is 0.04ms, while the total evaluation time in synthetic dataset is 0.36ms. This is because when $k =$ 15, the number of tuples with nonzero top-k probability in MOV is 75, while the number in synthetic dataset is 579.

C. Data Cleaning

We next examine the results for data cleaning. We assume that before running the cleaning algorithms, the query quality of has been obtained, and the values of $g(l, D) = \sum_{t_i \in \tau_l} \omega_i p_i$ have been stored in a lookup table such that they can be retrieved in $O(1)$ time.

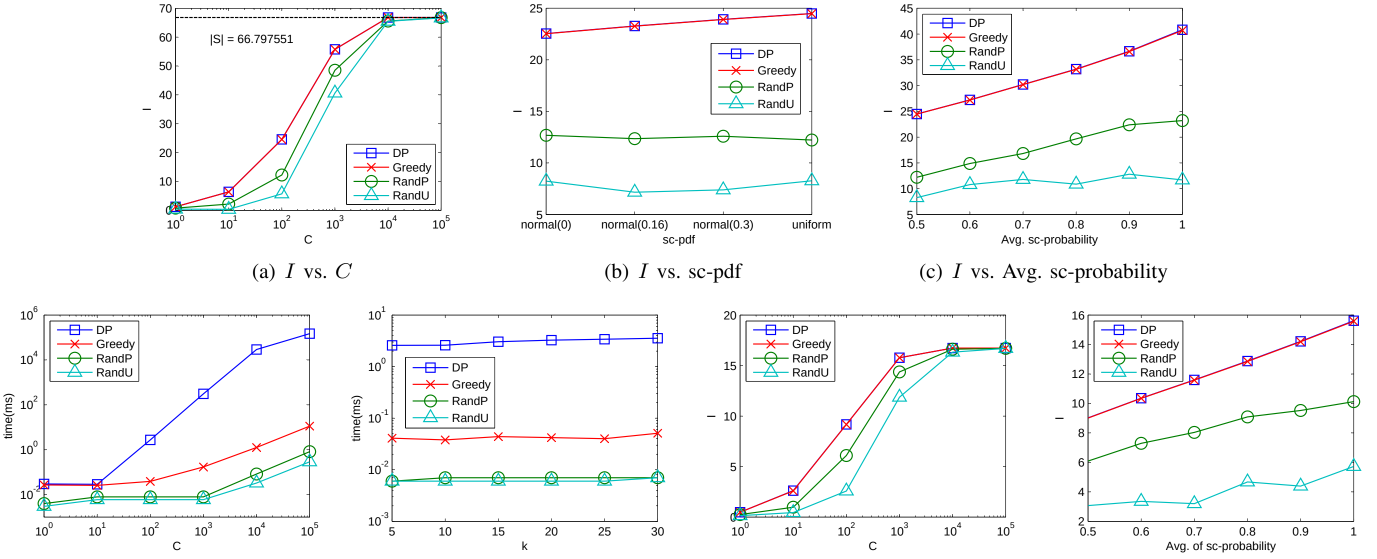
Effectiveness. We first compare the effectiveness of different cleaning algorithms. Figure 6(a) shows the quality improvement (I) under a wide range of budget (C). Since the quality score (S) of the default dataset for a top-15 query is -66.797551, the maximum expected quality improvement cannot exceed $|S|$. DP is the optimal solution, thus it performs the best. When C increases, it has more budget to perform a cleaning operation for more times, and so the quality improvement is close to the maximum value when C is large enough. It is worth to notice that $Greedy$ comes close to DP. This is because the cleaning problem is essentially a variant of *0-1 knapsack problem* (Section V-D.1), for which it has been proved that a greedy algorithm has a close-to-optimal performance [33]. 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 σ, 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 $[x, 1]$, thus its average value is $^{(1+x)}/_2$. 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



(a) I vs. C

(b) I vs. sc-pdf

(c) I vs. Avg. sc-probability

(d) Time vs. C

(e) Time vs. k

(f) I vs. C (MOV)

(g) I vs. Avg. sc-probability (MOV)

Fig. 6. Effectiveness and Efficiency of Cleaning Algorithms.

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-kRanks, 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.

References

[1] P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao, "Querying the uncertain position of moving objects," in *In Temporal Databases: Research and Practice*, 1998.

[2] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong, "Model-driven data acquisition in sensor networks," in *VLDB*, 2004.

[3] N. Dalvi and D. Suciu, "Efficient query evaluation on probabilistic databases," *The VLDB Journal*, 2007.

[4] A. moving rating database, "http://infolab.stanford.edu/trio/code/index.html #examples."

[5] R. Gupta and S. Sarawagi, "Creating probabilistic databases from information extraction models," in *VLDB*, 2006.

[6] P. Agrawal, O. Benjelloun, A. D. Sarma, C. Hayworth, S. Nabar, T. Sugihara, and J. Widom, "Trio: a system for data, uncertainty, and lineage," in *VLDB*, 2006.

[7] S. Singh, C. Mayfield, S. Mittal, S. Prabhakar, S. Hambrusch, and R. Shah, "Orion 2.0: native support for uncertain data," in *SIGMOD*, 2008.

[8] J. Boulos, N. Dalvi, B. Mandhani, S. Mathur, C. Re, and D. Suciu, "Mystiq: a system for finding more answers by using probabilities," in *SIGMOD*, 2005.

[9] J. Huang, L. Antova, C. Koch, and D. Olteanu, "Maybms: a probabilistic database management system," in *SIGMOD*, 2009.

[10] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang, "Top-k query processing in uncertain databases," in *ICDE*, 2007.

[11] M. Hua, J. Pei, W. Zhang, and X. Lin, "Ranking queries on uncertain data: a probabilistic threshold approach," in *SIGMOD*, 2008.

[12] K. Yi, F. Li, G. Kollios, and D. Srivastava, "Efficient processing of top-k queries in uncertain databases with x-relations," *TKDE*, 2008.

[13] X. Zhang and J. Chomicki, "On the semantics and evaluation of top-k queries in probabilistic databases," in *ICDE Workshop*, 2008.

[14] G. Cormode, F. Li, and K. Yi, "Semantics of ranking queries for probabilistic data and expected ranks," in *ICDE*, 2009.

[15] T. Bernecker, H. Kriegel, N. Mamoulis, M. Renz, and A. Zuefle, "Scalable probabilistic similarity ranking in uncertain databases," *TKDE*, 2010.

[16] R. Cheng, J. Chen, and X. Xie, "Cleaning uncertain data with quality guarantees," 2008.

[17] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler, "Lessons from a sensor network expedition," *Wireless Sensor Networks*, 2004.

[18] J. Li, B. Saha, and A. Deshpande, "A unified approach to ranking in probabilistic databases," 2009.

[19] X. Lian and L. Chen, "Probabilistic ranked queries in uncertain databases," in *EDBT08*.

[20] M. van Keulen and A. de Keijzer, "Qualitative effects of knowledge rules and user feedback in probabilistic data integration," *The VLDB Journal*, 2009.

[21] B. Kanagal, J. Li, and A. Deshpande, "Sensitivity analysis and explanations for robust query evaluation in probabilistic databases," in *SIGMOD*, 2011.

[22] R. Cheng, E. Lo, X. S. Yang, M.-H. Luk, X. Li, and X. Xie, "Explore or exploit? effective strategies for disambiguating large databases," 2010.

[23] J. Chen and R. Cheng, "Quality-aware probing of uncertain data with resource constraints," in *SSDBM*, 2008.

[24] Z. Liu, K. C. Sia, and J. Cho, "Cost-efficient processing of min/max queries over distributed sensors with uncertainty," in *SAD*, 2005.

[25] C. Olston, J. Jiang, and J. Widom, "Adaptive filters for continuous queries over distributed data streams," in *SIGMOD*, 2003.

[26] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate record detection: A survey," *IEEE Trans. on Knowl. and Data Eng.*, 2007.

[27] P. Andritsos, A. Fuxman, and R. J. Miller, "Clean answers over dirty databases: A probabilistic approach," in *ICDE*, 2006.

[28] G. Beskales, M. A. Soliman, I. F. Ilyas, and S. Ben-David, "Modeling and querying possible repairs in duplicate detection," 2009.

[29] N. Khoussainova, M. Balazinska, and D. Suciu, "Towards correcting input data errors probabilistically using integrity constraints," in *MobiDE*, 2006.

[30] C. Shannon, *The Mathematical Theory of Communication*. University of Illinois Press, 1949.

[31] Y. Li, Z. Wang, and Y. Song, "Wireless sensor network design for wildfire monitoring," in *Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on*. IEEE.

[32] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, 2001.

[33] G. Diubin and A. Korbut, "The average behaviour of greedy algorithms for the knapsack problem: general distributions," *Mathematical Methods of Operations Research*, 2003.

APPENDIX

TABLE III
SYMBOLS USED IN THIS PAPER.

Notation	Description
Data Model	
D	A probabilistic database
τ_l	An x-tuple of D with $l = 1, \ldots, m$
t_i	A tuple of D with $i = 1, \ldots, n$
ID_i	The key for t_i
x_i	The ID of the x-tuple (l) that t_i belongs to
v_i	Value attribute of tuple t_i
e_i	Existential probability of the tuple t_i
$\mathcal{W}(D)$	A set of all possible worlds of D
Probabilistic Top-k Query and Quality Metric	
$\mathcal{R}(D,Q)$	A set of all top-k PW-results of D
$S(D,Q)$	PWS-quality of probabilistic top-k query on D
p_i	The top-k probability of t_i
Data Cleaning	
C	Cleaning budget
c_l	Cost of performing $clean(\tau_l)$ once
P_l	Successful cleaning probability of τ_l
X	Set of x-tuples to be cleaned
M_l	Number of times that $clean(\tau_l)$ is performed

A. Proof of Theorem 1

For convenience, let d, r_j and q_j be the size of $\mathcal{R}(D,Q)$, the j-th pw-result in $\mathcal{R}(D,Q)$ and the probability of r_j, respectively.

By substituting Equation 2 into $\log q_j$, Equation 4 becomes

$$S(D,Q) = \sum_{j=1}^{d} q_j \log\Big(\prod_{t_i \in r_j} e_i \prod_{\tau_l \cap r_j = \emptyset} (1 - \sum_{\substack{t_i \in \tau_l \wedge \\ t_i > r_j.t}} e_i)\Big). \quad (28)$$

Using the property that $\log(ab) = \log a + \log b$, we have

$$S(D,Q) = \sum_{j=1}^{d} q_j \sum_{t_i \in r_j} \log e_i + \sum_{j=1}^{d} q_j \sum_{\tau_l \cap r_j = \emptyset} \log(1 - \sum_{\substack{t_i \in \tau_l \wedge \\ t_i > r_j.t}} e_i)). \quad (29)$$

By swapping the summation orders and notice that $p_i = \sum_{t_i \in r_j} q_j$, all q_js' in the first of Equation 29 can be replaced by p_is'. Hence,

$$\sum_{j=1}^{d} q_j \sum_{t_i \in r_j} \log e_i = \sum_{i=1}^{n} p_i \log e_i. \quad (30)$$

We next show how the q_js' in the second part of Equation 29 can be replaced by tuples' top-k probability. Observed that $\sum_{t_i \in \tau_l \wedge t_i > r_j.t} e_i$ is essentially the probability that τ_l has at least a tuple ranked higher than $r_j.t$. It can be obtained by summing up the existential probabilities of tuples in τ_l whose ranks are higher than $r_j.t$. Thus,

$$\sum_{t_i \in \tau_l \wedge t_i > r_j.t} e_i = \sum_{t_i \in \tau_l \wedge t_i \geq t(l, r_j.t)} e_i, \quad (31)$$

where $t(l, r_j.t)$ is the smallest tuple in τ_l whose rank is higher than $r_j.t$. Then the second part of Equation 29 equals to:

$$\sum_{\tau_l \in D} \sum_{t_i \in \tau_l} \log(1 - \sum_{t_{i'} \in \tau_l \wedge t_{i'} \geq t_i} e_{i'}) \sum_{\tau_l \cap r_j = \emptyset \wedge t(l, r_j.t) = t_i} q_j. \quad (32)$$

The summation of q_js' in Equation 12 is the probability that a pw-result r_j contains no tuple from τ_l and the smallest tuple in τ_l whose rank is higher than $r_j.t$ is tuple t_i. This probability can be further replaced by the tuples' top-k probabilities and existential probabilities in τ_l using the following facts.

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

$$\sum_{\tau_l \cap r_j = \emptyset \wedge t(l, r_j.t) = t_i} q_j = \sum_{\tau_l \cap r_j = \emptyset \wedge t_i > r_j.t \geq t_{\xi(t_i)}} q_j \quad (33)$$

Notice that, if t_i is the smallest tuple in τ_l, there is no pw-result r_j can satisfy the condition $\tau_l \cap r_j = \emptyset \wedge t_i > r_j.t$ since exactly one tuple from τ_l will exist in any possible world.

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 τ_l whose rank is not smaller than t_i does not appear. $\sum_{\tau_l \cap r = \emptyset \wedge t_i > r_j.t} \Pr(r)$ equals to the probability that A_i is true ($\Pr(A_i)$). Also,

$$p_i = e_i \, {}^{\Pr(A_i)}\!/\!_{(1 - \sum_{t_{i'} \in \tau_l \wedge t_{i'} \geq t_i} e_{i'})}. \quad (34)$$

In fact, ${}^{\Pr(A_i)}\!/\!_{(1 - \sum_{t_{i'} \in \tau_l \wedge t_{i'} \geq t_i} e_{i'})}$ is the probability that for tuples not in τ_l, there are 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. Thus,

$$\Pr(A_i) = (1 - \sum_{t_{i'} \in \tau_l \wedge t_{i'} \geq t_i} e_{i'}) \frac{p_i}{e_i}. \quad (35)$$

Similarly, let B_i be the event that (1) the rank of the smallest tuple in a pw-result is not higher than $t_{\xi(t_i)}$, and (2) tuples in τ_l whose rank is not smaller than t_i does not appear. We have

$$\Pr(B_i) = (1 - \sum_{t_{i'} \in \tau_l \wedge t_{i'} \geq t_i} e_{i'}) \frac{p_{\xi(t_i)}}{e_{\xi(t_i)}}. \quad (36)$$

According to the definition of $\xi(t_i)$, there is no tuple in τ_l whose rank is between t_i and $\xi(t_i)$. Thus, we have

$$\sum_{\tau_l \cap r_j = \emptyset \wedge t_i > r_j.t \geq t_{\xi(t_i)}} q_j = \Pr(A_i) - \Pr(B_i). \quad (37)$$

By substituting Equation 37 into q_j, Equation 32 becomes

$$\sum_{i=1}^{n} Y(1 - \sum_{t_{i'} \in \tau_l \wedge t_{i'} \geq t_i} e_{i'})(\frac{p_i}{e_i} - \frac{p_{\xi(t_i)}}{e_{\xi(t_i)}}). \quad (38)$$

Since $\xi(t_i)$ is the largest tuple in τ_l whose rank is lower than t_i, Equation 38 can be further rewritten as

$$\sum_{i=1}^{n} \frac{p_i}{e_i} (Y(1 - \sum_{x_{i'} = x_i \wedge t_{i'} \geq t_i} e_{i'}) - Y(1 - \sum_{x_{i'} = x_i \wedge t_{i'} > t_i} e_{i'})). \quad (39)$$

By substituting Equation 30 and Equation 39 into Equation 29, all q_js' can be replaced by tuples' top-k probability and existential probability, and we prove that Equation 5and 6 is correct.

B. Proof of Lemma 3

To evaluate $E(p'_i)$ directly, we should examine all possible cleaned databases. Furthermore, to compute p'_i in D', it requires querying on every possible world. Thus, we can first aggregate the same possible world from different cleaned databases, and evaluate $E(p'_i)$ on the aggregated possible worlds.

We next prove that that the probability of a possible world over all possible cleaned databases is the same with the probability of this possible world in D. Recall that a possible world contains exactly one tuple from each x-tuple, and its probability equals to the product of existential probability of tuples in this possible world, thus we only need to prove that the existential probability of a tuple t_i over all possible cleaned databases, denoted by $\tilde{e}_i$, is equal to e_i. Suppose $t_i \in \tau_l$, we consider two cases.

First, $\tau_l \notin X$. Since τ_l is not selected to clean, it is trivial that $\tilde{e}_i = e_i$.

Second, $\tau_l \in X$. In this case, $\tilde{e}_i$ is the summation of the following two probabilities:

1) τ_l is cleaned, and the remaining tuple is t_i. The probability equals to $e_i \times (1-(1-P_l)^{M_l})$;
2) τ_l is not cleaned, and t_i exists. The probability equals to $e_i \times (1-P_l)^{M_l}$.

Therefore, $\tilde{e}_i = e_i \times (1-(1-P_l)^{M_l}) + e_i \times (1-P_l)^{M_l} = e_i$.

Hence, $\tilde{e}_i = e_i$ for all $t_i \in D$, and we obtain the conclusion that the function of $E(p'_i)$ is the same with p_i.

C. Proof of Theorem 2

Observed that $E(S(D', Q))$ equals to

$$\sum_{l=1}^{|X|} E(g(l, D')) + \sum_{l=|X|+1}^{m} E(g(l, D')) \tag{40}$$

We first claim that if $1 \le l \le |X|$,

$$E(g(l, D')) = (1-P_l)^{M_l} g(l, D). \tag{41}$$

Consider two cases.

(1) If τ_l is cleaned, for all $t_i \in \tau_l$, $\omega'_i = 0$, and so $g(l, D') = \sum_{t_i \in \tau_l} \omega'_i p'_i = 0$;

(2) If τ_l is not cleaned, for all $t_i \in \tau_l$, $\omega'_i = \omega_i$. For the expected top-k probability of t_i over all possible cleaned databases where τ_l is not cleaned, it is equivalent to

$$\sum_{\vec{x}_0 \in z_1 \times \ldots \times z_{l-1} \times \tau_l \times z_{l+1} \times \ldots \times z_{|X|}} \Pr(\vec{x} = \vec{x}_0 | \vec{x}(l) = \tau_l) p'_i. \tag{42}$$

In fact, this is the same function with the expected top-k probability of t_i over all possible cleaned databases with $\tilde{X} = X - \{\tau_l\}$ and $\tilde{M} = M - \{M_l\}$. According to Lemma 3, this probability equals to p_i.

The probability that τ_l is cleaned is $(1-(1-P_l)^{M_l})$ and the probability that τ_l is cleaned is $(1-P_l)^{M_l}$. Hence, we have $E(g(l, D')) = (1-P_l)^{M_l} \sum_{t_i \in \tau_l} \omega_i p_i = (1-P_l)^{M_l} g(l, D)$.

We next claim that if $|X| + 1 \le l \le m$,

$$E(g(l, D')) = g(l, D). \tag{43}$$

Since τ_l is not selected to be cleaned, the existential probability of tuples in τ_l keeps unchanged. Thus, for all $t_i \in \tau_l$, 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 Equation 41 and 43 into Equation 40, we obtain the final result (Equation 19).

D. Proof of Lemma 4

We first prove that $g(l, D)$ is non-positive. Recall the proof of Theorem 1, $g(l, D)$ is essentially equivalent to

$$\sum_{t_i \in \tau_l} p_i \log e_i + \sum_{t_i \in \tau_l} \log\left(1 - \sum_{\substack{t_{i'} \in \tau_l \wedge \\ t_{i'} \ge t_i}} e_{i'}\right) \sum_{\substack{\tau_l \cap r_j = \emptyset \wedge \\ t_i > r_j.t > succ(t_i)}} q_j.$$

Since $0 < e_i \le 1, p(t_i) \ge 0, q_j \ge 0$, and $\sum_{t_i \in \tau_l} e_i = 1$, it is obvious that $g(l, D)$ is non-positive.

By using Equation 21, we have $b(l, D, j)$ is non-negative. Hence, $b(l, D, j+1) = (1-P_l) b(l, D, j) \le b(l, D, j)$.

E. A Dynamic Programming Solution

Algorithm 2 shows the details of DP algorithm. It first generates N items and decides its value and cost as discussed in Section V-D.1 (Step 1 to 8). $xid[i]$ indicates which x-tuple the i-th item is generated from.

Step 9 to 17 is the dynamic programming solution to solve the *0-1 knapsack problem*. $dp[c, i]$ $(c = 1, \ldots, C, i = 1, \ldots, N)$ is the optimal solution for the subproblem which contains the first i items and with budget equals to c. Thus, the optimal solution to the knapsack problem is $dp[C, N]$. Notice that, if the i-th item is selected, it consumes $cost[i]$ units of budget, and obtains $value[i]$ units of gain. Hence, we can obtain the recurrence as follows:

$$dp[c,i] = \begin{cases} dp[c, i-1] & \text{if } c < cost[i], \\ \max(dp[c, i-1], & \\ \quad dp[c - cost[i], i-1] + value[i]) & \text{otherwise.} \end{cases} \tag{44}$$

Furthermore, we use $case[c, i]$ to store whether the i-th item is selected (e.g., 0 means it is not selected) in order to obtain the optimal value of $dp[c, i]$.

In Step 18 to 28, we derive the optimal solution to the knapsack problem based on the information of $case[c, i]$, and then obtain the optimal solution to the cleaning problem according to Theorem 3.

The time complexity of DP is $O(C \times N)$. Since $dp[c, i]$ and $case[c, i]$ are both $C \times N$ arrays, the space complexity of DP is $O(C \times N)$. Observed that $N = \sum_{\tau_l \in Z} J_l = O(C|Z|)$, the time and space complexities of DP are both $O(C^2|Z|)$.

Algorithm 2 DP

```
Require: $Z$, $g(l,D)$, $P_l$, $c_l$, Budget $C$
$i \leftarrow 0$
for $l \leftarrow 1$ to $|Z|$ do
    $J_l \leftarrow \lfloor \frac{C}{c_l} \rfloor$
    for $j \leftarrow 1$ to $J_l$ do
        $b_{l,j} \leftarrow -(1-P_l)^{j-1} P_l \cdot g(l,D)$
        $i \leftarrow i+1$
        $xid[i] \leftarrow l$, $value[i] \leftarrow b_{l,j}$, $cost[i] \leftarrow c_l$
$N \leftarrow i$
$dp[0,i] \leftarrow 0, i = 0, \ldots, N$
for $i \leftarrow 1$ to $N$ do
    for $c \leftarrow 1$ to $C$ do
        if $c < cost[i]$ or $dp[c,i-1] > dp[c-cost[i],i-1] + value[i]$ then
            $dp[c,i] \leftarrow dp[c,i-1]$
            $case[c,i] \leftarrow 0$
        else
            $dp[c,i] \leftarrow dp[c-cost[i],i-1] + value[i]$
            $case[c,i] \leftarrow 1$
$M_l \leftarrow 0, l = 1, \ldots, |Z|$
$c \leftarrow C, i \leftarrow N$
while $c > 0$ and $i > 0$ do
    if $case[c,i] = 1$ then
        $M_{xid[i]} \leftarrow M_{xid[i]} + 1$
        $c \leftarrow c - cost[i]$
    $i \leftarrow i-1$
$X \leftarrow \emptyset$
for $l \leftarrow 1$ to $|Z|$ do
    if $M_l > 0$ then
        $X \leftarrow X \cup \{\tau_l\}$
  return $X$, $M$
```

Algorithm 3 RandU

```
Require: $Z$, $c_l$, Budget $C$
 1: $M_l \leftarrow 0, l = 1, \ldots, |Z|$
 2: while $C \geq \min c_l$ do
 3:     Randomly select $\tau_l$ from $Z$ (each x-tuple has probability of $^1/_m$ to be selected)
 4:     if $C > c_l$ then
 5:         $M_l \leftarrow M_l + 1$
 6:         $C \leftarrow C - c_l$
 7: $X \leftarrow \emptyset$
 8: for $l \leftarrow 1$ to $|Z|$ do
 9:     if $M_l > 0$ then
10:         $X \leftarrow X \cup \{\tau_l\}$
    return $X$, $M$
```

Algorithm 4 RandP

```
Require: $Z$, $p_i$, $c_l$, Budget $C$
 1: $M_l \leftarrow 0, l = 1, \ldots, |Z|$
 2: while $C \geq \min c_l$ do
 3:     Randomly select $\tau_l$ from $Z$ (each x-tuple has probability of $\sum_{t_i \in \tau_l} p_i/k$ to be selected)
 4:     if $C > c_l$ then
 5:         $M_l \leftarrow M_l + 1$
 6:         $C \leftarrow C - c_l$
 7: $X \leftarrow \emptyset$
 8: for $l \leftarrow 1$ to $|Z|$ do
 9:     if $M_l > 0$ then
10:         $X \leftarrow X \cup \{\tau_l\}$
    return $X$, $M$
```

Algorithm 5 Greedy

```
Require: $Z$, $g(l,D)$, $P_l$, $c_l$, Budget $C$
$M_l \leftarrow 0, l = 1, \ldots, |Z|$
for $l \leftarrow 1$ to $|Z|$ do
    $J_l \leftarrow \lfloor \frac{C}{c_l} \rfloor$
    for $j \leftarrow 1$ to $J_l$ do
        $b_{l,j} \leftarrow -(1-P_l)^{j-1} P_l \cdot g(l,D)$
        $\gamma_{l,j} \leftarrow \frac{b_{l,j}}{c_l}$
    Q.insert($\gamma_{l,1}$)            // Q is a heap.
while $C \geq \min c_l$ do
    $\gamma_{l,j} \leftarrow$ Q.pop()            // The largest $\gamma_{l,j}$ in Q is popped.
    if $C > c_l$ then
        $M_l \leftarrow M_l + 1$, $C \leftarrow C - c_l$
        if $j + 1 < J_l$ then
            Q.insert($\gamma_{l,j+1}$)
$X \leftarrow \emptyset$
for $l \leftarrow 1$ to $|Z|$ do
    if $M_l > 0$ then
        $X \leftarrow X \cup \{\tau_l\}$
  return $X$, $M$
```

F. *RandU and RandP*

The details of $RandU$ and $RandP$ are shown in Algorithm 3 and 4, respectively. The process of these two heuristics is very similar except the selection criterion. The probability of an x-tuple to be selected in $RandU$ is the same (Step 3 Algorithm 3), while the probability in $RandP$ is proportioned to the x-tuple's top-k probability (Step 3 Algorithm 4). The time and space complexities of these two heuristics are both $O(C)$ and $O(Z)$, respectively.

G. *Greedy*

We use a heap to maintain the items' scores (i.e., $\gamma_{l,j}$) and retrieve the items one by one in descending order of their scores. Observed that $b_{l,j+1} \leq b_{l,j}$, we have $\gamma_{l,j+1} \leq \gamma_{l,j}$. Using this observation, we can just maintain $\gamma_{l,1} (1 \leq l \leq |Z|)$ in the heap initially, and insert $\gamma_{l,j+1}$ into the heap after $\gamma_{l,j}$ is popped (during each iteration, only the one with largest value of $\gamma_{l,j}$ is popped). With this technique, the time for retrieving all items is reduced to $O(N \log |Z|)$, since the size of the heap is always not larger than $|Z|$. The details of $Greedy$ is shown in Algorithm 5.

The time complexity of $Greedy$ is $O(N \log |Z|) = O(C|Z| \log |Z|)$, while the space complexity is $O(N) = O(C|Z|)$, since $\gamma_{l,j}$ should be computed and stored in advance.