Cleaning Uncertain Data with Quality Guarantees

Reynold Cheng*
Department of Computer Science
The University of Hong Kong
Pokfulam Road, Hong Kong
ckcheng@cs.hku.hk

Jinchuan Chen Xike Xie
Department of Computing
The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong
{csjcchen,csxxie}@comp.polyu.edu.hk

ABSTRACT

Uncertain or imprecise data are pervasive in applications like location-based services, sensor monitoring, and data collection and integration. For these applications, *probabilistic databases* can be used to store uncertain data, and querying facilities are provided to yield answers with statistical confidence. Given that a limited amount of resources is available to "clean" the database (e.g., by probing some sensor data values to get their latest values), we address the problem of choosing the set of uncertain objects to be cleaned, in order to achieve the best improvement in the quality of query answers. For this purpose, we present the *PWS-quality* metric, which is a universal measure that quantifies the ambiguity of query answers under the *possible world semantics*. We study how PWS-quality can be efficiently evaluated for two major query classes: (1) queries that examine the satisfiability of tuples independent of other tuples (e.g., range queries); and (2) queries that require the knowledge of the relative ranking of the tuples (e.g., `MAX` queries). We then propose a polynomial-time solution to achieve an optimal improvement in PWS-quality. Other fast heuristics are presented as well. Experiments, performed on both real and synthetic datasets, show that the PWS-quality metric can be evaluated quickly, and that our cleaning algorithm provides an optimal solution with high efficiency. To our best knowledge, this is the first work that develops a quality metric for a probabilistic database, and investigates how such a metric can be used for data cleaning purposes.

1. INTRODUCTION

Traditionally, a database assumes that the values of the data stored are exact or precise. In many emerging applications, however, the database is inherently uncertain. Consider a habitat monitoring system where data like temperature, humidity, and wind speed are acquired from sensors. Due to the imperfect nature of the sensing instruments, the data obtained are often contaminated with noises [14]. As another example, in the Global-Positioning System (GPS), the location values collected have some measurement error [16, 32]. In biometric databases, the attribute values of the feature vectors stored are not exact [5]. Integration and record linkage tools also associate confidence values to the output tuples according to the quality of matching [12]. To deal with the increasing need of handling uncertainty, researchers have recently proposed to consider uncertainty as a "first-class citizen", by managing data in an "uncertain database" [1, 3, 9, 12].

In these databases, queries can be evaluated to produce imprecise answers with probabilistic guarantees. The ambiguity of a query answer constitutes the notion of *query quality*, which describes "how good" a query answer is [9]. In this paper, we address the issue of how to improve query quality, through the means of reducing the ambiguity of the database. Data impreciseness can be alleviated in different ways. For example, in a sensor monitoring application, a database system is used to store the current values of thousands of sensors deployed in a geographical region. Due to limited resources, the system may not be able to capture the sensor information at every point of time; instead, it uses the stored values to estimate the current sensor readings [9, 14]. To reduce the error of estimation, the system can "probe" a sensor, which responds to the system with its newest value. As another example, consider a database that captures the movie ratings, based on the fusion of the IMDB movie information and the user ratings obtained from the Netflix challenge [23]. The database contains the customers' ratings of each movie, represented as a probability distribution. The uncertainty about these ratings can be "sanitized" by contacting the respective customers for clarification. The resulting database, which is less uncertain than before, could then provide a higher-quality service.

Ideally, the whole database should be cleaned. In reality, this may not be feasible, since cleaning data can be costly. A sensor monitoring system, for example, may only probe a small portion of sensors, partly due to the limited bandwidth in the wireless network, and partly due to the scarce battery power of the sensing devices. As for the movie rating database, it may be difficult to validate the ratings of all the customers who are involved in the movie evaluation. Generally, a cleaning operation is constrained, for instance, by a fixed "budget", which describes the maximal amount of effort that can be invested for cleaning the data. The cleaning budget for a sensor monitoring system can be the maximum amount of bandwidth that can be used for sensor

*This work was done in Hong Kong Polytechnic University.

VLDB '08, August 24-30, 2008, Auckland, New Zealand

probing. For the movie-rating database, such a budget can be the maximum number of man-hours allowed for verifying the movie-ratings.

In this paper, we address the problem of cleaning uncertain data for achieving better query or service quality, under a limited budget. Despite the importance of uncertain database cleaning, relatively little work has been done in the area (e.g., [2, 6, 14, 18, 25]). Our main idea is to make use of the query information to decide the set of data items to be cleaned. By operating on these data, the quality of the answers returned to the user can attain the highest improvement. We develop our solution based on the *probabilistic database* [3, 12], a widely-studied uncertain data model. The main challenges that we address include: (1) define a sound and general quality metric over query results; (2) develop efficient methods to compute this metric; and (3) devise efficient and optimal cleaning algorithms.

To illustrate, Figure 1 shows a relation in a probabilistic database, which stores the quotations of four products (with IDs a, b, c and d), collected from webpages by using some automatic schema matching methods. An attribute called *existential probability* (Prob. in short) is used to indicate the confidence of the existence of each tuple. A tuple is also associated with an "x-tuple" [1], which represents a distribution of alternatives. For example, product a has a 0.7 chance for offering a price of \$120, and a 0.3 chance for having a quotation of \$80. Now consider a `MAX` query: "Return the tuple with the highest `price`". Due to data imprecision, this query can produce imprecise answers. Table 2 shows the query result, which contains the IDs of the tuples, and their non-zero probabilities (or *qualification probabilities*) for being the correct answers. These queries, which produce answers with statistical guarantees, are generally known as *probabilistic queries* [1, 9, 12].

Product ID	Tuple ID	Price ($)	Prob.
a	a_1	120	0.7
a	a_2	80	0.3
b	b_1	110	0.6
b	b_2	90	0.4
c	c_1	140	0.5
c	c_2	110	0.3
c	c_3	100	0.2
d	d_1	10	1

Table 1: Uncertain database example.

Tuple	Qualification Probability
a_1	0.35
b_1	0.09
c_1	0.5
c_2	0.09
c_3	0.024

Table 2: Results of the `MAX` query on Table 1.

Based on the answer probabilities, a real-valued "quality score" can be defined to capture the degree of ambiguity of a query answer. For example, the score of the `MAX` query result (Table 2) is -1.73 (according to our quality metric). Suppose Table 1 is partially cleaned (e.g., by consulting the companies about the actual prices of the products). Table 3 shows one possible scenario, where the uncertainties associated with x-tuples a and c are removed. In this table, only one tuple exists for each of a and c, and the existential probability of this tuple is equal to one. The new result of the `MAX` query is shown in Table 4, with a lower ambiguity, or an improved quality score of -0.97. In the extreme, if all the x-tuples are cleaned, the quality score becomes the highest (a value of zero with our metric).

Product ID	Tuple ID	Price ($)	Prob.
a	a_2	80	1
b	b_1	110	0.6
b	b_2	90	0.4
c	c_3	100	1
d	d_1	10	1

Table 3: A partially-cleaned instance of Table 1.

Tuple	Qualification Probability
b_1	0.6
c_3	0.4

Table 4: Results of the `MAX` query on Table 3.

How should such a query quality metric be defined? Although some quality measures have been proposed before, they are either catered for specific types of queries (e.g., [9, 10, 14, 29]), or not designed for use in a probabilistic database (e.g., [13, 17]). To solve these problems, we propose the *PWS-quality*. This metric provides a *universal measure* of query quality (i.e., can be used by any queries) for the probabilistic database. It is essentially an entropy function [28], which returns a real-valued score for conveniently indicating the amount of impreciseness in query answers. The PWS-quality also enables efficient data cleaning solutions, as we will show in this paper.

Another salient feature of PWS-quality is that it assumes the *Possible-World Semantics* (or PWS in short). The PWS provides a formal interpretation of the probabilistic data model [14], where a database is viewed as a set of deterministic database instances (called *possible worlds*), each of which contains a set of tuples extracted from each x-tuple. An example possible world for Table 1 contains the tuples $\{a_1, b_2, c_3, d_1\}$. Query evaluation algorithms for a probabilistic database should follow the notion of PWS, i.e., the results produced should be the same as if the query is evaluated on all the possible worlds [14]. Analogously, the PWS-quality score is calculated based on the query results obtained from all the possible worlds.

One apparent problem about the PWS-quality is that it is inefficient to calculate. This is because evaluating this measure requires examining all possible worlds, the number of which can be exponentially large [3, 12]. Interestingly, we observe that it is not often necessary to examine all the database instances; the PWS-quality can, in fact, be computed by using the query answers returned to the user. This is true for a broad class of queries known as the *entity-based* query [9]. This kind of query has the property that the final answer returned to the user contains the IDs of the tuples that satisfy it, as well as their qualification probabilities (e.g., Table 2). We study two representative examples of entity-based queries, namely, the range query and the `MAX` query. Both queries are used in many applications. For example, in a sensor-monitoring application, a range query can be: "Return the IDs of the sensors whose temperature values are within $[10^oC, 20^oC]$". In the movie database, a `MAX`

query can be: "Return the ID of the movie-viewer whose rating is the highest". We show that the PWS-quality of these two queries can be quickly computed by using query answer information. Our methods are effective because a query answer can be efficiently generated by existing query evaluation and indexing algorithms, and the complexity of our technique is linear to the size of the query answer.

The PWS-quality also serves as a useful tool for solving the data cleaning problem. Given the set of x-tuples to be cleaned, we prove that there is always a monotonic increase in the expected value of PWS-quality. This helps us to formulate the data cleaning problem as: choose the subset X of x-tuples such that (1) the increase in the expected quality of cleaning the x-tuples in X is the highest; and (2) the total cost of cleaning X does not exceed a given budget. This problem is challenging because calculating the expected quality improvement of X requires the processing of all combinations of the tuples in X. Moreover, a naïve approach of finding the optimal set X requires the testing of different combinations of x-tuples in the database, rendering an exponential time complexity. To solve these problems, we convert the PWS-quality expression into an "x-form" – a linear function of the probability information of the x-tuples. The x-form allows us to compute the expected quality improvement for cleaning a set of x-tuples easily. Moreover, it has the same format for both the range and the `MAX` queries (with different parameters), so that only a single solution is needed to support both queries. To find the optimal solution without testing all combinations of x-tuples from the whole database, we show that it is only necessary to select the x-tuples whose tuples appear in a query answer. We then model the cleaning task as an optimization problem, and develop a dynamic-programming-based algorithm, in order to deduce the optimal set of x-tuples in polynomial time. We also propose other approximate heuristics (such as the greedy algorithm). Our algorithms serve both the range and the `MAX` queries. They also support databases that contain tuples with the same attribute value.

We have performed detailed experimental evaluation to examine our approaches. For both real and synthetic datasets, the results show that PWS-quality can be efficiently computed. Moreover, x-tuples can be quickly selected to achieve an optimal improvement in expected quality. Among the heuristics, the greedy algorithm provides a close-to-optimal performance with the highest efficiency.

Figure 1 illustrates a system design that incorporates our solution. The query engine, upon receiving a user's request, produces a probabilistic query answer. This information is passed to the *quality manager*. Inside this module, the *quality evaluator* computes the PWS-quality score. It then sends the necessary information to the *data cleaning algorithm*, which derives the optimal set of x-tuples to be cleaned (or "cleaning set"), with the available budget considered. The *cleaning manager* is responsible for performing the sanitization activity (e.g., requesting the selected sources to report their updated values). The query, when executed again, will then have an improvement in the expected quality. [1] Notice that the quality manager is decoupled from the query engine, since it only requires the probability information of the answer tuples. Another issue is that the PWS-quality score is also sent to the user. This real-valued rating provides an intuitive way for the user to understand the degree of ambiguity in his results, without interpreting the numerous probability values that may appear in his query answers. In this paper, we focus on the design of the quality evaluator and the data cleaning algorithm.

[1]When the query is re-run on the refreshed database, a full evaluation is not needed, as explained in Appendix C.

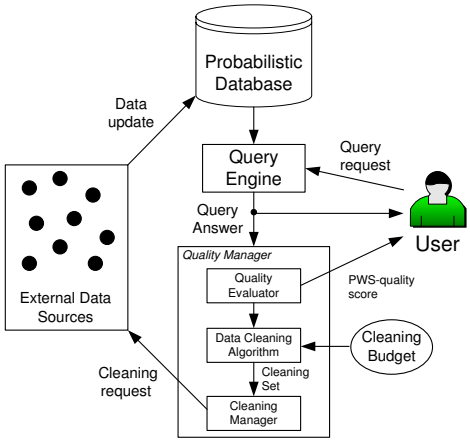


Figure 1: The framework of our solution.

The rest of this paper is organized as follows. In Section 2 we present the related works. Section 3 discusses the data and query models. In Section 4 we present the formal notion of the PWS-quality, and efficient methods for evaluating it. Section 5 describes the quality-based cleaning method and other heuristics. We present our experimental results in Section 6. The paper is concluded in Section 7. In Appendix A we detail the proof of finding the x-form for PMaxQ. Appendix B presents a dynamic programming algorithm for the cleaning problem. We also discuss how a query can be efficiently evaluated on a cleaned database in Appendix C.

2. RELATED WORK

Querying uncertain databases. Due to its simplicity and clarity in semantics, the probabilistic database model [3, 12] has received plenty of attention. Particularly, the notion of x-tuples [1] has been commonly adopted as a formal model for representing the uncertainty of tuples. Dalvi et al. [12] demonstrated that evaluating queries using the notion of PWS can be inefficient, since an exponentially large number of possible worlds needs to be examined. Thus, researchers have proposed to modify the query semantics. For example, efficient solutions for different variants of top-k queries are studied in [27, 33, 36]. Another well-studied data model is the "attribute uncertainty", where attribute values are characterized by a range and a probability distribution function (pdf) [9, 32]. For this data model, efficient evaluation and indexing algorithms have been proposed, including range queries [34], nearest-neighbor [7, 19], `MIN/MAX` queries [9, 14], skylines [26] and reverse skylines [20]. In [30], efficient query algorithms for uncertainty of categorical data are studied. Recently, a formal model for attribute uncertainty based on the PWS has been proposed [31]. In [4], the ULDB model is presented, which combines the properties of probabilistic and lineage databases. Although our work is based on probabilistic databases, the idea can potentially be extended to support other data models.

Quality metrics for uncertain data. A number of quality measures have been studied. In [10, 14], if the qualification probability of a result is higher than a user-defined threshold, then the query result is considered to be satisfactory. In [29], the quality of a top-k query is given by the fraction of the true top-k values contained in the query results. In [9], different metrics are defined for range queries, nearest-neighbor queries, `AVG` and `SUM` queries. In these works, quality metrics are designed for specific query types. The PWS-quality, on the other hand, provides a general notion of quality that can be applied to any kind of queries. Thus, PWS-quality can provide a fair comparison of quality among the answers from different queries. Another quality metric, called the query reliability, is defined in [13, 17]. However, this metric was not studied in the context of probabilistic databases. Moreover, it is not clear how they can be applied to the problem of data cleaning.

Cleaning uncertain data. In [6, 14, 21, 25], efficient methods for probing fresh data from stream sources and sensor networks have been considered. In [18], integrity constraints are used to clean dirty data. In [2], the authors examine the detection and merging of duplicate tuples in inconsistent databases. Complementary to these works, we study how the PWS-quality metric can be used to facilitate the cleaning of a probabilistic database.

3. DATA AND QUERY MODELS

We now describe the probabilistic data model (Section 3.1), and the types of queries studied in this paper (Section 3.2).

3.1 The Probabilistic Database Model

A probabilistic database D contains m entities known as the "x-tuples" [1, 3, 14]. We denote the k-th x-tuple by τ_k, where $k = 1, \ldots, m$. We also assume that the x-tuples are independent of each other. Each x-tuple is a set of tuples t_i, which represent a distribution of values within the x-tuple. There are a total of n tuples in D. Each tuple has four attributes: $(\mathrm{ID}_i, v_i, e_i, x_i)$. Here ID_i is an unique identifier of t_i, and v_i is a real-valued attribute used in queries (called the *querying attribute*). For simplicity, we assume v_i is one-dimensional, but our solutions can generally be extended to support multi-dimensional attributes. The attribute e_i is the existential probability of t_i – the probability that t_i exists in the real world. Each tuple belongs to one of the x-tuples, and $x_i = \{k | k = 1, \ldots, m\}$ denotes that t_i belongs to the k-th x-tuple.

Within the same x-tuple, the existence of tuples is mutually exclusive. We also assume that the sum s_k of all e_i's of the tuples in the same x-tuple τ_k is equal to 1. If s_k is less than 1, we conceptually augment a "null" tuple to τ_k, whose querying attribute has a value equal to $-\infty$, and existential probability equal to $1 - s_k$. This null tuple is only used for completeness in proofs; they do not exist physically. In Table 1, for example, there are four x-tuples (a, b, c, d). The "Price" and "Prob." columns represent the querying attribute and existential probability respectively.

3.2 Queries

We study two types of entity-based queries: *non-rank-based* and *rank-based* [9]. In a non-rank-based query, a tuple's qualification probability is independent of the existence of other tuples. For example, queries whose selection clauses involve only the attributes of a single tuple belong to this query class. Another good example is the range query:

Notation	Description
Data model	
D	A probabilistic database
τ_k	An x-tuple of D, with $k = 1, \ldots, m$
t_i	A tuple of D with $i = 1, \ldots, n$
ID_i	A unique identifier of t_i
v_i	Querying attribute of t_i
e_i	Existential probability of t_i
x_i	The ID of the x-tuple (k) that contains t_i
Query model	
Q	A probabilistic query
p_i	Qualification prob. of t_i for Q
P_k	Qualification prob. of τ_k for Q
Quality Metrics	
r_j	A distinct PW-result ($j = 1, \ldots, d$)
q_j	Prob. of occurrence of r_j
$S(D, Q)$	PWS-quality of Q on database D
Data Cleaning	
C	Cleaning budget for Q
c_k	Cost of cleaning τ_k
X	Set of x-tuples to be cleaned
$I(X, D, Q)$	Quality improvement of cleaning X

Table 5: Symbols used in this paper.

DEFINITION 1. **Probabilistic Range Query (PRQ).** *Given a closed interval $[a, b]$, where $a, b \in \Re$ and $a \leq b$, a PRQ returns a set of tuples (t_i, p_i), where p_i, the qualification probability of t_i, is the non-zero probability that $v_i \in [a, b]$.*

For a rank-based query, a tuple's qualification probability is dependent on the existence of other tuples. Examples of this class include the `MAX/MIN` query and the nearest-neighbor query. We study the `MAX` query in this paper.

DEFINITION 2. **Probabilistic Maximum Query (PMaxQ).** *A PMaxQ returns a set of tuples (t_i, p_i), where p_i, the qualification probability of t_i, is the non-zero probability that $v_i \geq v_j$, where $j \neq i \wedge j = 1, \ldots, n$.*

Although the answers returned by both queries have the same form, their PWS-quality scores are computed in a different way, as illustrated in the next section. We will also discuss how PWS-quality can be computed for other entity-based queries (e.g., nearest-neighbor queries). Table 5 shows the symbols used in this paper.

Let us now briefly explain how PRQ and PMaxQ can be evaluated efficiently (without consulting the possible worlds). A PRQ can be computed by examining each tuple t_i, and testing whether its querying attribute, v_i, is within $[a, b]$. If this is not true, then t_i's qualification probability, p_i, must be zero. Otherwise, $p_i = e_i$, its existential probability. The probability of t_i for satisfying the PMaxQ is the product of (1) its existential probability and (2) the probability that x-tuples other than the one that t_i belongs to do not have a tuple with value larger than v_i. Indexing solutions (e.g., B-tree and R-tree) can be built on the querying attributes in order to improve the query performance.

Also, as mentioned in Section 1, a query may need to be re-evaluated after cleaning is done. However, this round of query evaluation can be done more efficiently. In particular, only the x-tuples whose tuples appear in the query answer of

the first evaluation round need to be considered. We explain these details in Appendix C.

4. THE PWS-QUALITY

To understand this metric, let us review how the possible world semantics (PWS) are used to evaluate a query. As shown in Figure 2, a probabilistic database is expanded to a set of *possible worlds* (Step 1). The query is then issued on each possible world, producing answers that we call *PW-results* (Step 2). In Step 3, the PW-results are combined to produce the final query answer. For example, a possible world in Table 1 is the set W of tuples whose IDs are a_1, b_2, c_3 and d_1. If a `MAX` query is issued on `price`, the PW-result of W is a_1 (it has the largest `price`), with a probability of $0.7 \times 0.4 \times 0.2 \times 1 = 0.056$. In the final query answer, the qualification probability of a_1 is equal to the sum of the probabilities that a_1 is the answer in all possible worlds, i.e, 0.35.

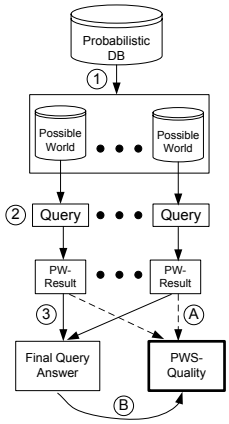


Figure 2: PWS and PWS-Quality.

The PWS-quality is essentially a function of the PW-results (computed in Step A in Figure 2). Since the form of the queries for computing the PW-results is not specified, the PWS-quality can be applied to *any* type of queries. The major problem of this approach is that there can be an exponentially large number of possible worlds [3, 12], so as the PW-results. Computing PWS-quality can thus be very costly. To address this problem, we show how PWS-quality can be evaluated by using the tuple information in the final query answers, instead of the PW-results (Step B), for PRQ and PMaxQ. Since the number of tuples in the query answer set is much smaller than that of the possible worlds, the computation efficiency of PWS-quality is also better.

We now examine the definition of PWS-quality (i.e., Step A) in Section 4.1. Then we propose a better method (i.e., Step B) in Section 4.2. Sections 4.3 and 4.4 outline the proofs of the new method for PRQ and PMaxQ.

4.1 Evaluating the PWS-Quality

The PWS-quality is essentially the entropy [28] of the PW-results produced in Step 2 of Figure 2. Let $\{r_1, \ldots, r_d\}$ be the set of d distinct PW-results. Also, let q_j be the probability that r_j is the actual answer (we call q_j the *PW-result probability* of r_j).

DEFINITION 3. *The* **PWS-quality** *of a query Q evaluated on a database D, denoted by $S(D, Q)$, is:*

$$S(D, Q) = \sum_{j=1}^{d} q_j \log q_j \tag{1}$$

Notice that the base of the log() function is 2. Moreover, the sum of the PW-result probabilities must be equal to one (i.e., $\sum_{j=1}^{d} q_j = 1$). By using this fact and comparing with the entropy function [28], it can be shown that the PWS-quality (Equation 1) is the negated value of the entropy of the PW-results. The entropy, a popular function for measuring uncertainty in the information theory literature, is adopted here to quantify the impreciseness of query answers. The value of the PWS-quality score ranges from $-\log d$ (i.e., the most ambiguous result) to zero (i.e., a single PW-result).

The problem of computing PWS-quality in this way is that we need to know all the PW-result probabilities. This may represent a performance bottleneck, since the number of PW-result possibilities, derived from possible worlds, can be exponentially large. This is true for a PRQ, where each PW-result contains a unique set of tuples whose querying attributes are inside the query range. For a PMaxQ, the number of PW-results can also be combinatorial, if more than one tuple contain the same querying attribute value. For instance, in Table 1, tuples b_1 and c_2 have the same `price` (i.e., \$110). We then have $2^2 - 1 = 3$ PW-results that contain one or more of these tuples as the answer (i.e., $\{b_1\}, \{c_2\}, \{b_1, c_2\}$), derived from the possible worlds where all tuples with `price` above \$110 are excluded. Let us investigate how PWS-quality can be evaluated more efficiently for these queries.

4.2 The x-Form of the PWS-Quality

The PWS-quality can in fact be computed by using the probability information of the tuples in the query answer (Step B of Figure 2). In particular, the PWS-quality (for both PRQ and PMaxQ) can be converted to an expression known as the *x-form*. An x-form is essentially a sum of some function g evaluated on each x-tuple τ_k, and g can be computed efficiently based on the probability information of the tuples in τ_k. For notational convenience, we use $Y(x)$ to denote the function $x \log x$. We also let P_k be the qualification probability of τ_k (i.e., its probability for satisfying the query). Since tuples belonging to the x-tuple are mutually exclusive, we have

$$P_k = \sum_{t_i \in \tau_k} p_i \tag{2}$$

The following lemma presents an alternative formula of PWS-quality.

LEMMA 1. *The x-form of the PWS-quality is given by:*

$$S(D, Q) = \sum_{k=1}^{m} g(k, D, Q) \tag{3}$$

For **PRQ**,

$$g(k, D, Q) = \sum_{t_i \in \tau_k} p_i \log e_i + Y(1 - P_k) \tag{4}$$

For **PMaxQ**, *let the i-th tuple of τ_k be $t_{k,i}$, sorted in descending order of $v_{k,i}$. If $t_{k,i}$ has existential probability $e_{k,i}$*

and qualification probability $p_{k,i}$, then,

$$g(k, D, Q) = \sum_{i=1}^{|\tau_k|} (p_{k,i} \log e_{k,i} + \omega_{k,i} \log(1 - \sum_{j=1}^{i} e_{k,j})) \quad (5)$$

where

$$\omega_{k,i} = \begin{cases} (1 - \sum_{j=1}^{i} e_{k,j})(\frac{p_{k,i}}{e_{k,i}} - \frac{p_{k,i+1}}{e_{k,i+1}}) & i < |\tau_k| \\ 0 & i = |\tau_k| \end{cases} \quad (6)$$

Lemma 1 states that PWS-quality is the sum of some function g for $k = 1, \ldots, m$. Each g is a function of existential and qualification probabilities of tuples of x-tuple τ_k. Interestingly, even though PRQ and PMaxQ have different semantics, their PWS-quality has the same form (i.e., Equation 3). Evaluating the x-form of PMaxQ needs some preprocessing, by sorting the tuples belonging to the same x-tuple with the querying attributes. An example of tuples sorted in this way is shown in Table 1.

Given that the values of $g(k, D, Q)$ are available, the x-form of both queries can be computed by iterating on the whole table of x-tuples, in $O(m)$ times. For PMaxQ, an additional average cost of $O(m \log \frac{n}{m})$ may be needed to sort the tuples. This is still faster than using an exponential number of PW-result probability values to evaluate the PWS-quality. The x-form is also useful to solve the data cleaning problem, to be presented in Section 5. We next show a useful fact.

LEMMA 2. *$g(k, D, Q) < 0$ if and only if there exists $t_i \in \tau_k$ such that $p_i \in (0, 1)$. Otherwise, $g(k, D, Q) = 0$.*

Given an x-tuple τ_k, the above states that $g(k, d, Q)$ is less than zero if there exists a tuple $t_i \in \tau_k$, such that its qualification probability, p_i, is neither zero nor one. [2]. More importantly, an x-tuple whose tuples' qualification probabilities are either zero or one *does not* need to be included in computing the PWS-quality (Equation 3). Thus, we do not need to examine the whole database. Instead, we can just pick the x-tuples that satisfy the conditions stated in Lemma 2. This is exactly the set of x-tuples whose tuples in the final query answer have qualification probabilities not equal to one. If we are given the query answer (which are produced by the query engine), then the set of x-tuples required to compute the PWS-quality can be derived easily.

We remark that the x-forms for PRQ and PMaxQ can also be used by other entity-based queries. Particularly, the x-form of PRQ can be used by other non-rank-based queries, whose selection conditions only involve the attributes of a single tuple. The x-form of the PMaxQ can also be used by `MIN` and nearest-neighbor queries, by using a different sorting criterion on the query attribute. Next, we explain briefly how the x-form is obtained for PRQ and PMaxQ.

4.3 Deriving the x-Form for PRQ

We now outline the proof of the x-form expression for PRQ. Here, a distinct PW-result r_j is essentially a set of tuples t_i's that satisfy the PRQ (i.e., $v_i \in [a, b]$) in one or more possible worlds. Note that r_j cannot possess more than one tuple from the same x-tuple, since each possible world only contains one of the tuples selected from each x-tuple. The probability q_j of getting r_j is then equal to:

$$q_j = \prod_{t_i \in r_j} e_i \prod_{\tau_k \cap r_j = \emptyset} (1 - P_k) \quad (7)$$

Equation 7 is the product of: (1) the probability all tuples t_i's that satisfy the PRQ are in r_j (i.e., $\prod_{t_i \in r_j} e_i$), and (2) the probability that other x-tuples satisfy the PRQ but do not appear in r_j(i.e., $\prod_{\tau_k \cap r_j = \emptyset}(1 - P_k)$). We then substitute this into Equation 1. We further observe that the qualification probability p_i (of tuple t_i) can be obtained by summing up the distinct PW-result probabilities (q_j's), where result r_j contains t_i. That is,

$$p_i = \sum_{j=1 \wedge t_i \in r_j}^{d} q_j \quad (8)$$

This is an important equation because it allows us to replace all q_j's with p_i's in the expression. Since there are at most m p_i's in the final query answer, the PWS-quality can be computed faster than using q_j's, the number of which is exponential. Finally, by using the property that $\log(ab) = \log a + \log b$ (where the log() function is used in the PWS-quality metric), we obtain a "sum-of-product" expression, as shown by Equation 4.

Notice that our proof still holds for other queries that involve different selection constraints, by replacing the condition that a tuple is included in a possible result (i.e., "$v_i \in [a, b]$") with the required conditions (e.g., "$v_i < a \vee v_i > b$"). That is, the x-form of PRQ can be generalized to a non-rank-based query, which tests whether a tuple satisfies it based on the tuple's own attributes. For details of the proof, the reader is referred to our technical report [8].

4.4 Deriving the x-Form for PMaxQ

The derivation of the x-form for PMaxQ is similar to that of PRQ, with the following major differences: (1) all tuples in an x-tuple are assumed to be sorted in descending order, and (2) the PW-result probability q_j has a different formula. Our solution also handles the scenario where more than one tuple with the same querying attribute value exist.

For convenience, let the i-th tuple of τ_k be $t_{k,i}$, sorted in descending order of $v_{k,i}$. A distinct PW-result r_j for PMaxQ is then a set of tuples t_i's that have the same maximum value, in one or more possible worlds. Let $r_j.v$ be the attribute value shared by the tuples in r_j. The probability q_j of getting r_j is then equal to:

$$q_j = \prod_{t_i \in r_j} e_i \prod_{\tau_k \cap r_j = \emptyset} Pr(\tau_k < r_j.v) \quad (9)$$

where $Pr(\tau_k < r_j.v)$ is the probability that τ_k has a tuple with querying attribute value smaller than $r_j.v$. Equation 9 is the product of: (1) the probability that all tuples t_i's in r_j exist (i.e., $\prod_{t_i \in r_j} e_i$), and (2) the probability that all other x-tuples have at least a tuple with a value smaller than $r_j.v$ (i.e., $\prod_{\tau_k \cap r_j = \emptyset} Pr(\tau_k < r_j.v)$). Moreover, since all tuples of an x-tuple are sorted in descending order, we can rewrite $Pr(\tau_k < r_j.v)$ as:

$$Pr(\tau_k < r_j.v) = 1 - \sum_{l=1}^{s(j,k)} e_{k,l} \quad (10)$$

[2]The proof of this fact is simple and can be found in our technical report [8].

where $s(j,k)$ is some integer inside $[1, |\tau_k|]$, such that $v_{k,s(j,k)}$ is the smallest value not smaller than $r_j.v$.

We then substitute Equations 9 and 10 into the PWS-quality definition (Equation 1), and derive the final result (Equation 5). These details are explained in Appendix A.

We can easily adapt the x-form of PMaxQ to other rank-based queries. For example, for `MIN` queries, we can sort the tuples within an x-tuple in ascending order, and change the comparison signs accordingly. As another example, the x-form for the nearest-neighbor query can be derived by ordering the tuples according to the Euclidean distance of their querying attributes from the query point.

5. CLEANING UNCERTAIN DATA

Let us now discuss how the PWS-quality can be used to facilitate the cleaning of uncertain data. Section 5.1 presents the formal definition of this problem. In Sections 5.2 and 5.3, we describe an efficient solution that can be applied to the queries under study. Several heuristics that provide efficient solutions are presented in Section 5.4.

5.1 Problem Definition

Recall that our goal is to select the most appropriate set of x-tuples to be cleaned, under a stringent budget, in order to achieve the highest expected quality improvement. Formally, let us define an operation called *clean*(τ_k):

DEFINITION 4. *Given an x-tuple τ_k, clean(τ_k) replaces τ_k with an x-tuple that contains a single tuple: $\{ID_i, v_i, 1, k\}$, such that ID_i and v_i are the corresponding identifier and querying attribute value of some tuple t_i that belongs to τ_k.*

Essentially, τ_k becomes "certain" after *clean*(τ_k) is performed. Only one of the tuples in the original x-tuple is retained, with existential probability changed to one. The value of the new tuple depends on the cleaning operation. In Table 1, for example, after *clean*(a) is performed, a contains a single tuple $\{a_2, 80, 1, a\}$, derived from a_2, with a `price` of \$80 and existential probability of 1.

Cleaning an x-tuple may involve a cost. For example, if an x-tuple represents a sensor reading in a sensor monitoring application, then the cost of cleaning this x-tuple (by probing the sensor to get the latest value) can be the amount of battery power required for that sensor's value to be shipped to the base station. We use c_k, a natural number, to capture the cost of performing *clean*(τ_k). We also assume that a query Q is associated with a *budget* of C units, where C is a natural number. This value limits the maximum amount of cleaning effort that can be used to improve the quality of Q. In sensor monitoring, C can be the total amount of energy allowed for probing the sensors. The value of C may be based on the amount of system resource available, or the priority of the query user.

Our goal is to obtain the set of x-tuples that, under a given budget, yields the most significant expected improvement in PWS-quality. This set of x-tuples is then selected to be cleaned. Specifically, let X be any set of x-tuples chosen from database D. Without loss of generality, let $X = \{\tau_1, \ldots, \tau_{|X|}\}$. Also, let $\vec{t}$ be a "tuple vector" of $|X|$ dimensions, where the k-th dimension of $\vec{t}$ is a tuple that belongs to the k-th x-tuple of X. For example, if $X = \{\tau_1, \tau_2\}$, where $\tau_1 = \{t_0, t_3\}$ and $\tau_2 = \{t_2, t_5\}$, then two possible values of $\vec{t}$ are $\{t_0, t_5\}$ and $\{t_3, t_2\}$.

Now, let $D'(\vec{t})$ be the new database obtained, after *clean*(τ_k) is performed on each x-tuple τ_k in X, which produces tuples described in $\vec{t}$. The expected quality of cleaning a set X of x-tuples is then equal to:

$$E(S(D'(\vec{t}), Q)) = \sum_{\vec{t} \in \tau_1 \times \ldots \times \tau_{|X|}} \Pi_{t_i \in \vec{t}} e_i \cdot S(D'(\vec{t}), Q) \quad (11)$$

For every tuple vector in $\tau_1 \times \ldots \times \tau_{|X|}$, Equation 11 calculates the probability that the new database $D'(\vec{t})$ is obtained (i.e., $\Pi_{t_i \in \vec{t}} e_i$) and the PWS-quality score of query Q evaluated on $D'(\vec{t})$ (i.e., $S(D'(\vec{t}), Q)$).

DEFINITION 5. *The quality improvement of cleaning a set X of x-tuples is*

$$I(X, D, Q) = E(S(D'(\vec{t}), Q)) - S(D, Q) \quad (12)$$

Our problem can now be formulated as follows:

DEFINITION 6. **The Data Cleaning Problem.** *Given a budget of C units, choose a set X of x-tuples from D such that $I(X, D, Q)$ attains the highest value.*

A straightforward way of solving this problem is to obtain the powerset of all x-tuples in D. For each element (a set X of x-tuples) of the powerset, we test whether the total cost of cleaning the x-tuples in X exceeds the budget C. Among those that do not, we select the set of x-tuples whose quality improvement is the highest.

This solution is inefficient for two reasons. First, given a set X of x-tuples, computing Equation 12 requires the consideration of all tuple vectors of X, which are the combinations of tuples selected from the x-tuples in X. Second, the number of sets of x-tuples to be examined is exponential. We tackle the first problem in Section 5.2. The second problem is addressed in Section 5.3.

5.2 Evaluating Quality Improvement

Equation 12 can be computed more easily by using the x-form of PWS-quality, as shown by the following lemma.

LEMMA 3. *The quality improvement of cleaning a set X of x-tuples is:*

$$I(X, D, Q) = -\sum_{k=1}^{|X|} g(k, D, Q) \quad (13)$$

where $g(k, D, Q)$ is given by Equations 4 and 5, for PRQ and PMaxQ respectively.

PROOF. By using the x-form (Equation 3), we can rewrite $E(S(D'(\vec{t}), Q))$ as

$$\sum_{k=1}^{|X|} E(g(k, D'(\vec{t}), Q)) + \sum_{k=|X|+1}^{m} E(g(k, D'(\vec{t}), Q)) \quad (14)$$

For both PRQ and PMaxQ, we claim that:

$$g(k, D'(\vec{t}), Q)) = 0, \textit{for } k = 1, \ldots, |X| \quad (15)$$

$$E(g(k, D'(\vec{t}), Q)) = g(k, D, Q), \textit{for } k = |X|+1, \ldots, m \quad (16)$$

By using Equations 15 and 16, Equation 14 can be written as $\sum_{k=|X|+1}^{m} g(k, D, Q)$. Together with Equations 12 and 3, we

can see that Equation 13 is correct. The following sketches the proof of Equations 15 and 16 for PRQ and PMaxQ.

PRQ: First, notice that the new database $D'(t)$ contains a single tuple for every $\tau_k \in X$, whose existential probability is 1, and qualification probability is either 0 or 1. Using this fact and Equation 4, we can see Equation 15 is true for every $k \in [1, |X|]$. For Equation 16, observe that $g(k, D'(\vec{t}), Q)$ is just some function (Equation 4) of p_i's and e_i's for $t_i \in \tau_k$, where $\tau_k \notin X$. As discussed in Section 3.2, the value of p_i for PRQ is either e_i or zero. Since these values of p_i's and e_i's are not changed by any cleaning operations on the x-tuples in X, Equation 16 holds for $k = |X| + 1, \ldots, m$.

PMaxQ: Let the existential and qualification probabilities of each tuple $t_{i,k}$ for the new database $D'(\vec{t})$ be $e'_{k,i}(\vec{t})$ and $p'_{k,i}(\vec{t})$ respectively. Then, After *clean*(τ_k), only one tuple $(t_{k,1})$ in τ_k can exist in $D'(\vec{t})$. Since $\omega_{k,1} = 0$ (Equation 5), we have $g(k, D'(\vec{t}), Q) = p'_{k,1}(\vec{t}) \log e'_{k,1}(\vec{t})$. Moreover, $e'_{k,1}(\vec{t}) = 1$. Thus, $g(k, D'(\vec{t}), Q) = 0$ and the proof for Equation 15 is complete.

To prove Equation 16, note that by using Equation 5, $E(g(k, D'(\vec{t}), Q))$ can be written as:

$$\sum_{i=1}^{|\tau_k|} (E(p'_{k,i}(\vec{t})) \log e_{k,i} + (\frac{E(p'_{k,i}(\vec{t}))}{e_{k,i}} - \frac{E(p'_{k,i+1}(\vec{t}))}{e_{k,i+1}}) Y (1 - \sum_{j=1}^{i} e_{k,j})) \quad (17)$$

Next, we claim that

$$E(p'_{k,i}(\vec{t})) = p_{k,i}, \forall k > |X| \quad (18)$$

In order to compute $E(p'_{k,i}(\vec{t}))$ directly, $p'_{k,i}(\vec{t})$ needs to be evaluated for every vector $\vec{t} \in \tau_1 \times \ldots \times \tau_{|X|}$. Furthermore, computing each $p'_{k,i}(\vec{t})$ involves querying on every possible world in $D'(\vec{t})$. Thus, $E(p'_{k,i}(\vec{t}))$ is just some function of all the PW-result probabilities queried on D, and this is the same function for $p_{k,i}$. Thus, Equation 18 is correct. Finally, by substituting Equation 18 into Equation 17, we obtain Equation 16. □

Equation 13 reveals three important facts. First, the quality improvement, $I(X, D, Q)$, is non-negative (since $g(k, D, Q)$ is non-positive). This implies that the expected quality monotonically increases with the performance of the *clean*(τ_k) operation. Second, the task of computing $I(X, D, Q)$ is made easier (compared with Equation 12), since $g(k, D, Q)$ can be computed in polynomial time. If these g values have been stored (e.g., in a lookup table) during the process of computing the x-form of the PWS-quality (Equation 3), then $I(X, D, Q)$ can be evaluated by a table lookup. Third, Equation 13 can be applied to both PRQ and PMaxQ, since $g(k, D, Q)$ have been derived for both queries in Section 4.2. Let us see how these results can be used to develop an efficient data cleaning algorithm.

5.3 An Optimal and Efficient Data Cleaning Algorithm

We now address the second question: to find out the set B of x-tuples that leads to the optimal expected quality improvement in PWS-quality, is it possible to avoid enumerating all the combinations of x-tuples in the whole database? To answer this, we first state the following lemma:

LEMMA 4. *For any x-tuple $\tau_k \in B$, τ_k must satisfy the condition: there exists $t_i \in \tau_k$ such that (t_i, p_i) appears in the final answer of Q, with $p_i \in (0, 1)$.*

PROOF. Consider an x-tuple τ_j, whose tuples' qualification probabilities are either zero or one. We can show that τ_j does not need to be included in B. Suppose by contradiction that $\tau_j \in B$. According to Lemma 2, $g(j, d, Q) = 0$. By using Lemma 3, we can see that including τ_j in B has no effect on the quality improvement i.e., $I(B, D, Q)$. Thus, it is unnecessary to include τ_j in B.

In fact, by excluding τ_j, the remaining x-tuples that we need to consider for cleaning are those that contain at least a tuple t_i with the following conditions: (1) t_i appears in the final query answer, and (2) $p_i \in (0, 1)$. □

For example, for the `MAX` query evaluated on Table 1, the optimal set B can be derived from the result of the `MAX` query (Table 2), which contains the tuples from x-tuples a,b, and c, but not d. Correspondingly, B is the subset of the x-tuples $\{a, b, c\}$. Thus, Lemma 4 reduces the search space to the x-tuples whose tuples appear in the query answer. It also means that the input of our data cleaning algorithm can be the tuples contained in the query answer (c.f. Figure 1).

We now focus on the x-tuples that satisfy the conditions of Lemma 4. Let Z be the number of these x-tuples. We use τ_k (where $k = 1, ..., Z$) to denote these x-tuples.

An Optimization Problem. We now present an efficient algorithm that provides an optimal solution to the data cleaning problem. This algorithm can be applied to entity-based queries, including PRQ and PMaxQ. We assume the values of $g(k, D, Q)$ have been obtained for all values of $k = 1, \ldots, Z$. For notational convenience, we also use g_k to represent $g(k, D, Q)$ (since D and Q are constant parameters). Then, Definition 6 can be reformulated as an optimization problem $P(C, M)$, where $M = \{\tau_1, \ldots, \tau_Z\}$ is the set of candidates to be considered, and C is the budget assigned to the query:

Maximize

$$\sum_{k=1}^{Z} b_k \cdot g_k \quad (19)$$

Subject to

$$\sum_{k=1}^{Z} b_k \cdot c_k \leq C \quad (20)$$

Here $\{b_k | k = 1, \ldots, Z, b_k = 0|1)$ is a bit vector of length Z, encoding the IDs of x-tuple(s) chosen from M to be cleaned. Particularly, $b_k = 1$ if x-tuple τ_k is selected, and $b_k = 0$ otherwise. Equation 19 is the total quality improvement for cleaning a set of x-tuples (where τ_k is chosen if $b_k = 1$), which is the same as Equation 13. The optimization constraint is described in Equation 20, which requires that the total cost of cleaning the set of x-tuples cannot be more than C. Note that since Equation 19 (or Equation 13) is true for PRQ and PMaxQ, the solution to this problem can be applied to both queries.

We further note that $P(C, M)$ is essentially a variant of the *0/1 knapsack* problem [11], which can be solved by using dynamic programming techniques. The details of this algorithm are presented in Appendix B. The time and space complexities of this solution are respectively $O(CZ)$ and $O(CZ^2)$.

5.4 Heuristics for Data Cleaning

To further improve the efficiency of data cleaning, we have developed three other heuristics:

1. Random: This is the simplest heuristic, where x-tuples are selected randomly until the query budget is exhausted.
2. MaxQP: Compute the qualification probability P_k for each x-tuple τ_k, using Equation 2. Then, choose the x-tuples in descending order of P_k (where $P_k \neq 1$) until the total cost exceeds C. The rationale is that selecting x-tuples with higher qualification probabilities may have a better effect on the PWS-quality than those with small values.
3. Greedy: Let $f_k = \frac{g_k}{c_k}$. Select x-tuples with the highest values of f_k such that the maximum total cost is less than C. Intuitively, f_k is the quality improvement of $clean(\tau_k)$ per unit cost. The choice of x-tuples is decided by the amount of quality improved and the cost required.

The MaxQP and Greedy heuristics can be extended to support large query answer sets. Specifically, if the number of x-tuples to be considered by the data cleaning algorithm is too large to be stored in the main memory, disk-based algorithms (e.g., [24]) can be used to sort the x-tuples. Then, the x-tuples that rank the highest can be retrieved. In our experiments, since the main memory is large enough to hold the tuples returned to a user, our data cleaning algorithms are executed on the main memory.

As a sidenote, consider a query evaluated on the database before cleaning is performed. After the database is cleaned, the re-running of this query does not require the examination of the whole database. This is because only the tuples that appear in the query result of the pre-cleaned database need to be handled. We explain this "incremental processing technique" in Appendix C.

6. RESULTS

We now discuss the experimental results. In Section 6.1 we describe the experiment settings. We present our findings in Section 6.2.

6.1 Experimental Setup

We have used a synthetic dataset in our experiments. This dataset contains $10K$ objects (e.g., products), each of which has a 1D attribute y (as a price collected automatically from web pages), in the domain $[0, 10,000]$. The value of y follows *attribute uncertainty* described in [9, 35], where y has two components: "uncertainty interval" $y.L$ and "uncertainty pdf" $y.U$. The center of $y.L$ is uniformly distributed in the domain, and the range of $y.L$ is uniformly distributed in the range of $[60, 100]$. The uncertainty pdf $y.U$ is a Gaussian distribution defined on $y.L$, with the mean equal to the center of $y.L$, and the variance of 100 units. To store these objects in a probabilistic database, we discretize $y.U$ by obtaining its histogram representation, where the probabilities of 10 equal "histogram bars" within $y.L$ are computed. Each object is then treated as an x-tuple, under which 10 tuples are created, whose querying attributes are the mean values of the histogram bars, and the existential probabilities are the probabilities computed for the histogram bars. Our synthetic database thus has $10K$ x-tuples, or $100K$ tuples.

We also perform experiments on a real dataset [23], which contains some uncertainty in the viewers' ratings for specific movies. The table has 4,999 x-tuples, or 10,037 tuples. It has five attributes: `<movie-id,customer-id,date,rate, confidence>`, where `<movie-id,customer-id>` is the key of the x-tuple, and `confidence` records the existential probability of a tuple.

To model the data cleaning problem, for both datasets we attach a "cleaning cost" attribute to each x-tuple. This cost is an integer, uniformly distributed in the range of $[1, 10]$. The query budget has a default value of 30 units.

We have implemented both PRQ and PMaxQ for the synthetic dataset. For PRQ, the query range has a width of 20 units, and its position is evenly distributed in the domain. For the real dataset, the PRQs use `<date,rate>` as a $2D$ querying attribute. We also implement a probabilistic nearest-neighbor query (PNNQ) with a random 4D query point q on the dimensions `<movie-id,customer-id,date,rate>`. Notice that a PNNQ is essentially a PMaxQ by ranking on the Euclidean distance of each data point from q. Thus our algorithms can also be used by a PNNQ.

We have used an R-tree based on the codes in [22] to index the querying attributes, in order to improve the efficiency of computing the query answers. The two primary metrics used for evaluation are: (1) PWS-quality score; and (2) quality evaluation time. Notice that metric (2) does not include the time required for evaluating the query answer.

Each data point is the average of 100 queries. Unless stated otherwise, the results are based on the synthetic dataset. Our codes, implemented in J2SE 1.5.0_09, are run on a PC with an Intel T2400 CPU of 1.83GHz and 1GB memory.

6.2 Results

Section 6.2.1 investigates the expressiveness of PWS-quality. Section 6.2.2 examines the x-forms of PWS-quality, and Section 6.2.3 presents the results on data cleaning. We report the results on a real dataset in Section 6.2.4.

6.2.1 Expressiveness of PWS-Quality

We first investigate how well PWS-quality can quantify the ambiguity of query results. We use z to denote the number of distinct tuples in the query answer, whose qualification probabilities are non-zero. Figure 3 shows the quality score of PRQ (S) under a wide range of z. We see that the quality score decreases (i.e., a degradation in quality) when z increases. Intuitively, the larger the value of z, the more tuples are in the query answers, implying a more uncertain answer. Thus, the PWS-quality naturally reflects the vagueness in a query answer.

In the same graph, we present the PWS-quality for two different uncertainty pdfs ($y.U$) of the attribute y in our dataset. As we can see, the uniform pdf generally demonstrates a lower quality score than its Gaussian counterpart. This is not surprising, since a uniform pdf has a larger entropy (i.e., more uncertain) than a Gaussian pdf. Consequently, the query answer becomes more ambiguous, as illustrated by the lower quality scores.

Next, we compare the quality scores of PRQ and PMaxQ in five different databases. For fairness, we compare the scores only for the queries that produce the same number of tuples (with a 1% difference) in their answers in the same database. As shown in Figure 4, PRQ scores lower than PMaxQ across all the database samples. The reason is that the PWS-quality is an entropy function of PW-result probabilities (Equation 1). On average, the PRQ (which finds tuple(s) with querying attribute(s) in a specified range) yields more PW-results than PMaxQ (which finds tuple(s) that gives the maximum value). Hence, the answer of PRQ is also more uncertain than PMaxQ, as shown by our results.

6.2.2 Evaluation of PWS-Quality

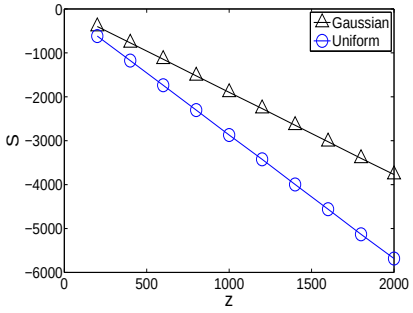


Figure 3: Quality vs. z.

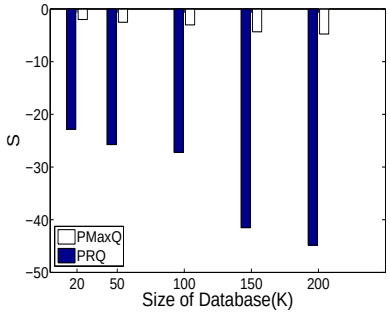


Figure 4: Quality vs. Database Size.

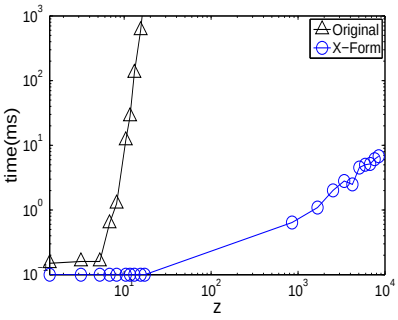


Figure 5: The x-Form (PRQ).

Sanity Check. We first verify the correctness of the x-form by running several experiments. We found that the relative difference between the x-form and the original definition of PWS-quality (Equation 1) is in the order of 10^{-4} or less. For PRQ, at $z = 3.18$, the relative difference is $2.08e{-}6$. For PMaxQ, that difference is $3.99e - 6$ at $z = 77.46$. The slight difference is due to the precision loss at computing small probability values.

Evaluation Time. Figure 5 compares the time required for calculating the x-form and the original definition of PWS-quality for PRQ. The amount of time for both methods increases with z, since more result probabilities have to be considered. However, the x-form needs much shorter time to evaluate than the original definition. This follows from the fact that the x-form (Equation 4) runs in polynomial time, whereas the original definition (Equation 1) requires an exponential time complexity.

Figure 6 shows the time required to compute the x-form for PRQ, and the query evaluation time. We notice that the former needs no more than 10% of the time required by the latter. The quality evaluation time for PMaxQ, not shown here, requires an average of 0.16 ms, or 1.6% of the query evaluation time. The difference in the evaluation time of a query and its quality may actually be larger, since in these experiments we have constructed indexes to speed up the query evaluation. Thus computing PWS-quality adds little overhead to the query evaluation process.

Effect of duplicate tuples. We also study the effect of "duplicate tuples" on computing the PWS-quality of PMaxQ. These are the tuples whose querying attribute values are the same. We modify the synthetic database by treating attribute values within a range of ± 1 as a single value. As a result, each querying attribute value is associated with an average of 6.33 tuples. We found that computing PWS-quality with the original definition takes 259.58 ms to complete, while x-form can finish the job in 3.48 ms. The huge difference (98.6% improvement) is due to the fact that the original definition has to consider a large number of PW-results due to the duplicate tuples, but the x-form only needs to iterate over the x-tuples in the answer once. We also test with other databases; since the results are similar, they are not reported here.

6.2.3 Data Cleaning

Next, we examine the results for data cleaning. We assume that the quality of the queries being tested has been obtained. Moreover, prior to cleaning, all the values of $g(D, k, Q)$, computed during the evaluation of the x-forms, have been stored in a lookup table. We first compare the performance of the "enhanced" method in calculating quality improvement (Equation 13), with its "original" definition (i.e., Definition 5). Figure 7 shows the results for both methods on a given set X of x-tuples, with $|X| = 1, \ldots, 5$. The time required by the original definition increases sharply with $|X|$. The enhanced method just needs to sum up the $g(D, k, Q)$ values for the x-tuples $\tau_k \in X$, and these values can be retrieved from the lookup table. Thus, its execution time is much less. Thus, the enhanced method will be used in our subsequent experiments.

We then study the time required to compute the quality improvement, using the methods presented in Sections 5.3 and 5.4. Here, the *Basic* method means the quality improvement of each member of the powerset of all x-tuples is examined, and then the set of x-tuples that yields the highest improvement is chosen. Figure 8 examines the amount of time (in log scale) for different methods under a wide range of query budgets used by the PMaxQ. We see that *Basic* performs the worst. The *DP* method provides an optimal solution in polynomial time, and so it is faster than *Basic*. However, its time is higher than other heuristics (i.e., *Random*, *MaxQP* and *Greedy*). The results for PRQ are similar and so they are skipped here.

Figures 9 and 10 examine the quality improvement (I) for PRQ and PMaxQ respectively. Since both *Basic* and *DP* give the optimal solution, for clarity we only show the result for *DP*. Although *DP* performs the best, it is worth notice that both *Greedy* and *MaxQP* come close to it. This is because *Greedy* selects the x-tuple according to the cost and the benefit of cleaning it, while *MaxQP* gives priority to x-tuples with higher qualification probabilities. These factors are important to deciding the optimal solution. Moreover, the data cleaning problem is a variant of the knapsack problem [11], and it has been shown in [15] that the average performance of a greedy solution is close to the optimal one. *Random* does not consider any of these factors at all, and thus it performs the worst. Observe that *MaxQP* is better in PMaxQ than in PRQ. In PMaxQ, by considering a x-tuple with the highest qualification probability, the tuple that remains in that x-tuple may have a chance to give the highest value than all other tuples, yielding a high-quality result; and so the expected improvement in quality is also higher than the case of PRQ.

We also compare the quality improvement (I) of PRQ and PMaxQ, relative to their original quality (S). We assume the DP algorithm is used to obtain the x-tuples. The results, shown in Figure 11, reveal that under a fixed query budget and answer size, the relative quality improvement ($I/|S|$) for PMaxQ is consistently higher than that of PRQ.

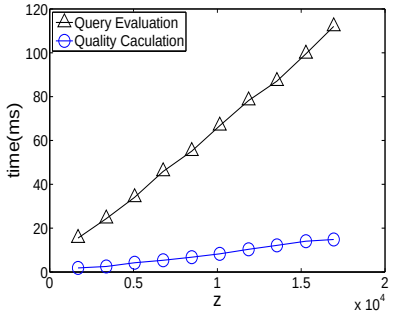


Figure 6: Query vs. Quality Evaluation Time.

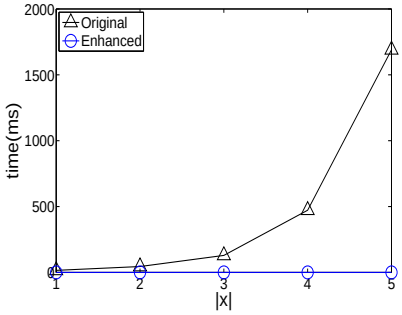


Figure 7: Evaluation Time of Quality Improvement.

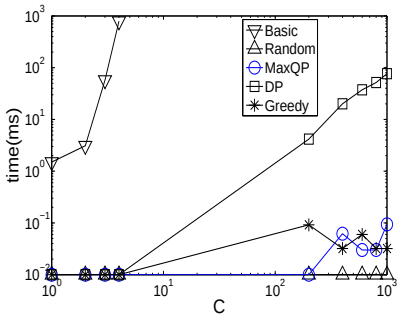


Figure 8: Time for selecting x-tuples (PMaxQ).

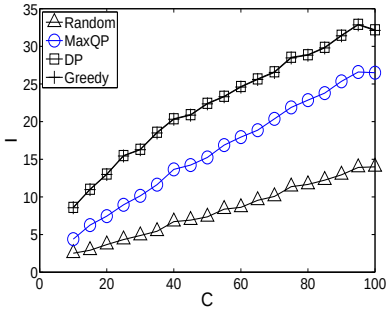


Figure 9: I vs. C (PRQ).

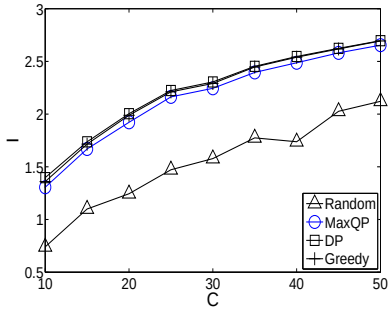


Figure 10: I vs. C (PMaxQ).

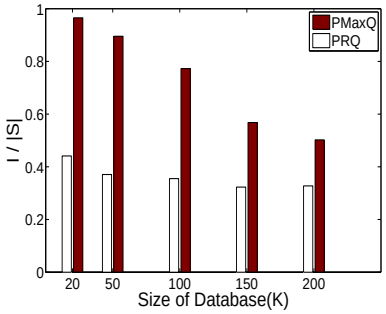


Figure 11: PRQ vs. PMaxQ ($I/|S|$).

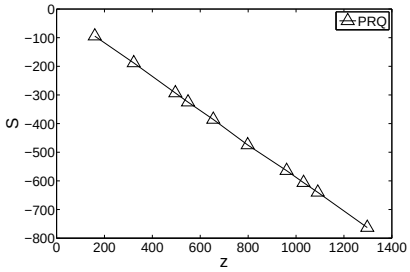
Note that PMaxQ has fewer distinct PW-results than PRQ (Section 6.2.1). Thus there are less tuples in the answer of a PMaxQ than that of a PRQ. Cleaning an x-tuple for PMaxQ then has more impact. In environments where multiple queries are concurrently executed, a system can thus choose to place more effort on PMaxQ than on PRQ.

6.2.4 *Results on the Real Dataset*

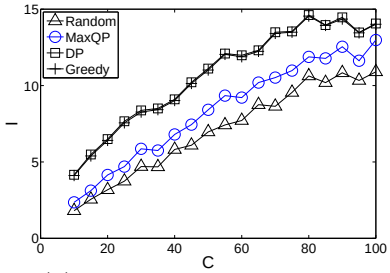
We now present selected results for the real dataset. Figure 12(a) shows the quality of PRQ under different values of z. Similar to Figure 3, the quality of PRQ worsens as z increases. On the other hand, PNNQ has an average score of -0.86. The reason for the high quality obtained by PNNQ is that the dataset also has a high quality: on average, each x-tuple has 2 tuples, and 33% of the x-tuples have no uncertainty (i.e., they only have one single tuple). Thus, it is easy to obtain an unambiguous answer for PNNQ.

We also observe that the PWS-quality scores of the queries in the real dataset are generally higher than those obtained for the synthetic dataset. To understand why, for each x-tuple, we have measured the entropy of the existential probabilities of all tuples in a x-tuple. We found that the average of these entropy values over the real dataset is 0.78, which is lower than that of the synthetic dataset (1.85). Thus, the real dataset is generally less uncertain than the synthetic one, and the PWS-quality scores for the real dataset are also better.

Finally, Figure 12(b) shows the quality improvement of PRQ under different query budgets. The results are similar to those for the synthetic data (Figure 9). We have also measured the quality improvement for the PNNQ. Since its original quality score is high, the data cleaning algorithms does not have much effect on the quality. Thus, we do not show their results here.



(a) Quality vs. z.



(b) Quality Improvement vs. C.

Figure 12: Results on Real Data Set

7. CONCLUSIONS

The management of uncertain and probabilistic databases has become an important topic in emerging applications. In this paper, we investigated a cleaning problem for these databases, with the goal of optimizing the expected quality improvement under a limited budget. To accomplish this task, we designed the PWS-quality metric to quantify query answer ambiguities. We showed how PWS-quality can be

efficiently computed for common entity-based queries (PRQ and PMaxQ). We also illustrated that it is possible to develop optimal and efficient solutions around this metric.

We plan to extend our solutions to support other kinds of queries, e.g., top-k query. We will also examine other cleaning models, e.g., a cleaning request that may not be immediately accomplished. We can also investigate how to perform optimal cleaning where an x-tuple, after cleaning, becomes a set of tuples with arbitrary distributions. It is also interesting to study how cleaning can be done on databases where the uncertainty of attributes is given by a continuous distribution (e.g., [16, 32]).

Acknowledgments

This work was supported by the Research Grants Council of Hong Kong (Projects PolyU 5138/06E and PolyU 5133/07E), and the Germany/HK Joint Research Scheme (Project No. G_HK013/06). We also thank Prof. Charles Ling (University of Western Ontario, Canada) and the reviewers for their insightful comments.

8. REFERENCES

[1] 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.

[2] P. Andritsos, A. Fuxman, and R. Miller. Clean answers over dirty databases: A probabilistic approach. In *ICDE*, 2006.

[3] D. Barbara, H. Garcia-Molina, and D. Porter. The management of probabilistic data. 4(5), 1992.

[4] O. Benjelloun, A. Sarma, A. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In *VLDB*, 2006.

[5] C. Böhm, A. Pryakhin, and M. Schubert. The gauss-tree: Efficient object identification in databases of probabilistic feature vectors. In *ICDE*, 2006.

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

[7] R. Cheng, J. Chen, M. Mokbel, and C. Chow. Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data. In *ICDE*, 2008.

[8] R. Cheng, J. Chen, and X. Xie. Cleaning uncertain data with quality guarantees (technical report). In `http://www2.comp.polyu.edu.hk:8080/~csjcchen/quality.pdf`.

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

[10] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter. Efficient indexing methods for probabilistic threshold queries over uncertain data. In *VLDB*, 2004.

[11] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2001.

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

[13] M. de Rougemont. The reliability of queries. In *PODS*, 1995.

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

[15] G. Diubin. The average behaviour of greedy algorithms for the knapsack problem: general distributions. *Mathematical Methods of Operations Research*, 57(3), 2003.

[16] D.Pfoser and C. Jensen. Capturing the uncertainty of moving-objects representations. In *SSDBM*, 1999.

[17] E. Gradel, Y. Gurevich, and C. Hirsch. The complexity of query reliability. In *PODS*, 1998.

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

[19] H. Kriegel, P. Kunath, and M. Renz. Probabilistic nearest-neighbor query on uncertain objects. In *DASFAA*, 2007.

[20] X. Lian and L. Chen. Monochromatic and bichromatic reverse skyline search over uncertain databases. In *SIGMOD*, 2008.

[21] Z. Liu, K. Sia, and J. Cho. Cost-efficient processing of min/max queries over distributed sensors with uncertainty. In *ACM SAC*, 2005.

[22] M.Hadjieleftheriou. Spatial index library 0.44.2b. `http://u-foria.org/marioh/spatialindex/index.html`.

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

[24] M. Nodine and J. Vitter. Greed sort: An optimal sorting algorithm for multiple disks. *JACM*, 42(4), 1995.

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

[26] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *VLDB*, 2007.

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

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

[29] A. Silberstein, R. Braynard, C. Ellis, K. Munagala, and J. Yang. A sampling-based approach to optimizing top-k queries in sensor networks. In *ICDE*, 2006.

[30] S. Singh, C. Mayfield, S. Prabhakar, R. Shah, and S. Hambrusch. Indexing uncertain categorical data. In *ICDE*, 2007.

[31] Singh et al. Database support for pdf attributes. In *ICDE*, 2008.

[32] P. A. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Querying the uncertain position of moving objects. In *Temporal Databases: Research and Practice*. Springer Verlag, 1998.

[33] M. Soliman, I. Ilyas, and K. Chang. Top-k query processing in uncertain databases. In *ICDE*, 2007.

[34] Y. Tao, R. Cheng, X. Xiao, W. K. Ngai, B. Kao, and S. Prabhakar. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In *VLDB*, 2005.

[35] O. Wolfson, A. Sistla, S. Chamberlain, and Y. Yesha. Updating and querying databases that track mobile units. *Distributed and Parallel Databases*, 1999.

[36] K. Yi, F. Li, D. Srivastava, and G. Kollios. Efficient processing of top-k queries in uncertain databases. In *ICDE*, 2008.

APPENDIX

A. THE PWS-QUALITY OF PMAXQ

We now show the proof that the x-form of PWS-quality for PMaxQ is given by Equation 5.

By substituting Equation 9 into $\log q_i$, Equation 1 becomes:

$$S(D,Q) = \sum_{j=1}^{d} q_j \Big(\sum_{i=1 \wedge t_i \in r_j}^{n} \log e_i + \sum_{k=1 \wedge \tau_k \cap r_j = \emptyset}^{n} \log(Pr(\tau_k < r_j.v)) \Big) \tag{21}$$

Similar to the proof in Section 4.3, we have

$$\sum_{j=1}^{d} \sum_{i=1 \wedge t_i \in r_j}^{n} q_j \log e_i = \sum_{i=1}^{n} p_i \log e_i \tag{22}$$

Next we prove that

$$\sum_{j=1}^{d} \sum_{k=1 \wedge \tau_k \cap r_j = \emptyset}^{n} q_j \log(Pr(\tau_k < r_j.v)) = \sum_{k=1}^{n} \sum_{i=1}^{|\tau_k|} \omega_{k,i} \log(1 - \sum_{j=1}^{i} e_{k,j}) \tag{23}$$

By substituting Equation 10 into $\log Pr(\tau_k < r_j.v)$, the left part of Equation 23 becomes:

$$\sum_{j=1}^{d} \sum_{k=1 \wedge \tau_k \cap r_j = \emptyset}^{n} q_j \log(1 - \sum_{l=1}^{s(j,k)} e_{k,l}) \tag{24}$$

By swapping the summation orders and noticing that $s(j,k)$ is just some number in $[1, |\tau_k|]$, Equation 24 can be simplified to the form of Equations 3 and 5.

The rest is to show that $\omega_{k,i}$ is equivalent to Equation 6. First, notice that $\omega_{k,i}$ is just a sum of q_j's (for PW-results r_j). In particular,

$$\omega_{k,i} = \sum_{v_{k,i+1} < r_j.v \le v_{k,i}} q_j \tag{25}$$

This is because the value i in the term $\log(1 - \sum_{j=1}^{i} e_{k,j})$ of Equation 5 represents the i-th position of the tuple in the x-tuple τ_k (i.e., $t_{k,i}$) which has the smallest querying attribute larger than $r_j.v$, which must be between $(v_{k,i+1}, v_{k,i}]$, or else i cannot be the position where $r_j.v$ is just larger than $v_{k,i}$. Notice that if $i = |\tau_k|$, no tuples $t_{h,l}$ can satisfy this condition, and so $\omega_{k,|\tau_k|} = 0$.

Next, we claim that

$$\sum_{r_j.v \le v_{k,i}} q_j = (1 - \sum_{l=1}^{i} e_{k,l}) \cdot \frac{p_{k,i}}{e_{k,i}} \tag{26}$$

To prove Equation 26, note that its left side is the probability that the answer to PMaxQ has a querying attribute value smaller than $v_{k,i}$. This is equal to the product of the occurrence probabilities of two events: E_1, the event that all tuples that do not belong to τ_k and whose values are larger than $v_{k,i}$ do not exist; E_2, the event that none of the tuples $\{t_{k,1}, \ldots, t_{k,i}\}$ exist. The probability that E_1 is true ($Pr(E_1)$) can be derived by using the fact that $p_{k,i} = Pr(E_1) \cdot e_{k,i}$. Thus, $Pr(E_1) = \frac{p_{k,i}}{e_{k,i}}$. Since $Pr(E_2) = 1 - \sum_{j=1}^{i} e_{k,j}$, Equation 26 can be obtained.

Finally, Equation 25 can be written as

$$\sum_{v_{h,l} \le v_{k,i}} p_{h,l} - p_{k,i+1} - \sum_{v_{h,l} \le v_{k,i+1}} p_{h,l} \tag{27}$$

By substituting Equation 27 with the result of Equation 26, we prove that Equation 6 is correct.

B. A DYNAMIC PROGRAMMING SOLUTION

The problem $P(C, M)$ obeys the *optimal substructure* property, which enables dynamic programming [11]. Let W, a set of x-tuples, be the optimal solution to $P(C, M)$. Consider the problem $P(C - c_a, M - \{\tau_a\})$, where τ_a is some x-tuple in M. Let $W' = W - \{\tau_a\}$. Then, W' must also be an optimal solution to $P(C - c_a, M - \{\tau_a\})$. To see this, suppose W'' (where $W'' \neq W'$) is the optimal solution to $P(C - c_a, M - \{\tau_a\})$. Then $W^{\#} = W'' \cup \{\tau_a\}$ is also a solution to $P(C, M)$. Since W'' is better than W', the quality improvement for $W^{\#}$ (obtained by adding g_a to the quality improvement of W'' using Equation 19) must also be higher than W. This violates the assumption that W is the optimal solution to $P(C, M)$. Therefore, W' must be the optimal solution to $P(C - c_a, M - \{\tau_a\})$.

We consider the subproblem $P(c, M_k)$, where $0 \le c \le C$ is the current budget available to the query. Also, $M_k = \{\tau_1, \ldots, \tau_k | 1 \le k \le Z\}$ is the set of x-tuples that can be selected ($M_k = \emptyset$ for $k = 0$). For each subproblem $P(c, M_k)$, we use a Z-bit vector array, $s[c,k]$, to store the optimal set of x-tuples, where $s[c,k][j]$ has a value of 1 if τ_j is chosen in the solution. Another $C \times Z$ array h is used to store the quality improvement for cleaning with the set encoded by $s[c,k]$. For the cases of $c = 0$ or $k = 0$, we initialize the values of s and h to zero. Figure 13 shows the *DP* algorithm. As we can see, all the problem instances are scanned once (Steps 1 and 2). For each problem $P(c, M_k)$, we test whether the current x-tuple being examined (τ_k) yields a higher quality improvement than the current set (Step 3). If that is true, Steps 7-9 adds τ_k to the solution and updates s and h. Otherwise, both $s[c,k]$ and $h[c,k]$ use the solution without τ_k (Steps 4-5). Finally, $s[C, Z]$ is returned as an optimal solution to $P(C, M)$. The time and space complexities of this algorithm are respectively $O(CZ)$ and $O(CZ^2)$.

C. INCREMENTAL QUERY PROCESSING

Consider a query Q which has just been evaluated on the database D. Let D' be the new database after cleaning is done. We now show that rerunning Q on D' does not require the examination of all tuples in D'. In particular, let $R(D)$ be the set of tuples appearing in the answer of Q on D. We claim that the tuples that are included in the answer of Q being evaluated on D', i.e., $R(D')$, must be a subset of $R(D)$. We term this *incremental query processing*, since the evaluation of Q on D' can be based upon the answer set $R(D)$, instead of the whole database D'.

To see this, let u be a tuple which does not appear in $R(D)$. Then, u must not satisfy Q in any possible worlds generated from D. Now, consider a possible world W that appears in D. After cleaning (Definition 4) is completed,

Input
Cost $(c_1, \ldots, c_Z)$
Quality improvement $(g_1, \ldots, g_Z)$
Budget (C)

1. **for** $k \leftarrow 1$ **to** Z **do**
2. **for** $c \leftarrow 1$ **to** C **do**
3. **if** $c_k > c$ **or** $h[c, k-1] > h[c - c_k, k-1] + g_k$
4. $s[c, k] \leftarrow s[c, k-1]$;
5. $h[c, k] \leftarrow h[c, k-1]$;
6. **else**
7. $s[c, k] \leftarrow s[c - c_k, k-1]$;
8. $s[c, k][k] \leftarrow 1$;
9. $h[c, k] \leftarrow h[c - c_k, k-1] + g_k$;
10. **return** $s[C, Z]$;

Figure 13: The DP algorithm.

two cases can occur: (1) W consists of the same set of tuples; or (2) W is eliminated, where one or more tuples that belong to W are removed due to cleaning. Hence, the set of possible worlds of D' must be a subset of those in D. This implies u still cannot satisfy Q in any possible world of D'. Consequently, u must not be in $R(D')$. Hence, in the second round of query evaluation, we only need to examine the tuples that appear in $R(D)$, instead of D'. This can reduce the effort of evaluating a query on D' significantly.

Next, let us investigate how PRQ and PMaxQ on D' make use of the above observation.

PRQ: We only need to re-evaluate the probabilities of the tuples belonging to the x-tuples in the cleaning set, since these x-tuples are chosen from the answer set $R(D)$ (according to Lemma 2). Qualification probabilities of tuples that do not appear in the cleaning set remain unchanged.

PMaxQ: Let t_m be the tuple that consists of the maximum attribute value among all the cleaned tuples. We claim that only tuples whose values larger than or equal to v_m need to be considered. Note that after cleaning, the existential probability of t_m becomes one. Thus all tuples with attribute values smaller than v_m, which has no chance to be the maximum object, can be removed from $R(D)$. The remaining tuples are inserted to $R(D)$, with qualification probabilities recomputed.