Evaluating Probabilistic Queries over Imprecise Data *

Reynold Cheng   Dmitri V. Kalashnikov   Sunil Prabhakar
Department of Computer Science, Purdue University
Email: {ckcheng,dvk,sunil}@cs.purdue.edu
http://www.cs.purdue.edu/place/

ABSTRACT

Many applications employ sensors for monitoring entities such as temperature and wind speed. A centralized database tracks these entities to enable query processing. Due to continuous changes in these values and limited resources (e.g., network bandwidth and battery power), it is often infeasible to store the exact values at all times. A similar situation exists for moving object environments that track the constantly changing locations of objects. In this environment, it is possible for database queries to produce incorrect or invalid results based upon old data. However, if the degree of error (or uncertainty) between the actual value and the database value is controlled, one can place more confidence in the answers to queries. More generally, query answers can be augmented with probabilistic estimates of the validity of the answers. In this paper we study probabilistic query evaluation based upon uncertain data. A classification of queries is made based upon the nature of the result set. For each class, we develop algorithms for computing probabilistic answers. We address the important issue of measuring the quality of the answers to these queries, and provide algorithms for efficiently pulling data from relevant sensors or moving objects in order to improve the quality of the executing queries. Extensive experiments are performed to examine the effectiveness of several data update policies.

1. INTRODUCTION

In many applications, sensors are used to continuously track or monitor the status of an environment. Readings from the sensors are sent back to the application, and decisions are made based on these readings. For example, temperature sensors installed in a building are used by a central air-conditioning system to decide whether the temperature of any room needs to be adjusted or to detect other problems. Sensors distributed in the environment can be used to detect if hazardous materials are present and how they are spreading. In a moving object database, objects are constantly monitored and a central database may collect their updated locations.

The framework for many of these applications includes a database or server to which the readings obtained by the sensors or the locations of the moving objects are sent. Users query this database in order to find information of interest. Due to several factors such as limited network bandwidth to the server and limited battery power of the mobile device, it is often infeasible for the database to contain the exact status of an entity being monitored at every moment in time. An inherent property of these applications is that readings from sensors are sent to the central server periodically. In particular, at any given point in time, the recorded sensor reading is likely to be different from the actual value. The correct value of a sensor's reading is known only when an update is received. Under these conditions, the data in the database is only an estimate of the actual state of the environment at most points in time.

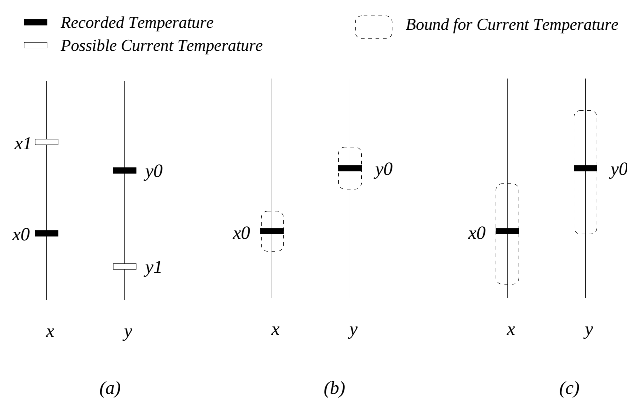

Figure 1: Example of sensor data and uncertainty.

This inherent uncertainty of data affects the accuracy of answers to queries. Figure 1(a) illustrates a query that determines the sensor (either x or y) that reports the lower temperature reading. Based upon the data available in the database (x_0 and y_0), the query returns "x" as the result. In reality, the temperature readings could have changed to values x_1 and y_1, in which case "y" is the correct answer. This example demonstrates that the database does not always truly capture the state of the external world, and the value of the sensor readings can change without being rec-

* Portions of this work were supported by NSF CAREER grant IIS-9985019, NSF grant 0010044-CCR, NSF grant 9988339-CCR and Intel PhD Fellowship

SIGMOD 2003, June 9-12, San Diego, CA.

ognized by the database. Sistla et. al [12] identify this type of data as a *dynamic attribute*, whose value changes over time even if it is not explicitly updated in the database. In this example, because the exact values of the data items are not known to the database between successive updates, the database incorrectly assumes that the recorded value is the actual value and produces incorrect results.

Given the uncertainty of the data, providing meaningful answers seems to be a futile exercise. However, one can argue that in many applications, the values of objects cannot change drastically in a short period of time; instead, the degree and/or rate of change of an object may be constrained. For example, the temperature recorded by a sensor may not change by more than a degree in 5 minutes. Such information can help solve the problem. Consider the above example again. Suppose we can provide a guarantee that at the time the query is evaluated, the actual values monitored by x and y could be no more than some deviations, d_x and d_y, from x_0 and y_0, respectively, as shown in Figure 1(b). With this information, we can state with confidence that x yields the minimum value.

In general, the uncertainty of the objects may not allow us to identify a single object that has the minimum value. For example, in Figure 1(c), both x and y have the possibility of recording the minimum value since the reading of x may not be lower than that of y. A similar situation exists for other types of queries such as those that request a numerical value (e.g., "What is the lowest temperature reading?"). For these queries too, providing a single value may be infeasible due to the uncertainty in each object's value. Instead of providing a definite answer, the database can associate different levels of confidence with each answer (e.g., as a probability) based upon the uncertainty of the queried objects.

The notion of probabilistic answers to queries over uncertain data has not been well studied. Wolfson et. al briefly touched upon this idea [14] for the case of range queries in the context of a moving object database. The objects are assumed to move in straight lines with a known average speed. The answers to the queries consist of objects' identities and the probability that each object is located in the specified range. Cheng et. al [3] considered the problem of answering probabilistic nearest neighbor queries under a moving-object database model. In this paper we extend the notion of probabilistic queries to cover a much broader class of queries. The class of queries considered includes aggregate queries that compute answers over a number of objects. We also discuss the importance of the nature of answer requested by a query (identity of object versus the value). For example, we show that there is a significant difference between the following two queries: "Which object has the minimum temperature?" versus "What is the minimum temperature?". Furthermore, we relax the model of uncertainty so that any reasonable model can be used by the application. Our techniques are applicable to the common models of uncertainty that have been proposed elsewhere.

The probabilities in the answer allow the user to place appropriate confidence in the answer as opposed to having an incorrect answer or no answer at all. Depending upon the application, one may choose to report only the object with the highest probability as having the minimum value, or only those objects whose probability exceeds a minimum probability threshold. Our proposed work will be able to work with any of these models.

Answering aggregate queries (such as *minimum* or *average*) is much more challenging than range queries, especially in the presence of uncertainty. The answer to a probabilistic range query consists of a set of objects along with a non-zero probability that the object lies in the query range. Each object's probability is determined by the uncertainty of the object's value and the query range. However, for aggregate queries, the interplay between multiple objects is critical. The resulting probabilities are greatly influenced by the uncertainty of attribute values of other objects. For example, in Figure 1(c) the probability that x has the minimum value is affected by the relative value and bounds for y.

A probabilistic answer also reflects a certain level of uncertainty that results from the uncertainty of the queries' object values. If the uncertainty of all (or some) of the objects was reduced (or eliminated completely), the uncertainty of the result improves. For example, without any knowledge about the value of an object, one could arguably state that it falls within a query range with 50% probability. On the other hand, if the value is known perfectly, one can state with 100% confidence that the object either falls within or outside the query range. Thus the quality of the result is measured by degree of ambiguity in the answer. We therefore need metrics to evaluate the quality of a probabilistic answer. We propose metrics for evaluating the quality of the probabilistic answers. As we shall see, it turns out that different metrics are suitable for different classes of queries.

It is possible that the quality of a query result may not be acceptable for certain applications – a more definite result may be desirable. Since the poor quality is directly related to the uncertainty in the object values, one possibility for improving the results is to delay the query until the quality improves. However this is an unreasonable solution due to the increased query response time. Instead, the database could request updates from all objects (e.g., sensors) – this solution incurs a heavy load on the resources. In this paper, we propose to request updates only from objects that are being queried, and furthermore those that are likely to improve the quality of the query result. We present several heuristics and an experimental evaluation. These policies attempt to optimize the use of the constrained resource (e.g., network bandwidth to the server) to improve average query quality.

It should be noted that the imprecision in the query answers is inherent in this problem (due to uncertainty in the actual values of the dynamic attribute), in contrast to the problem of providing approximate answers for improved performance wherein accuracy is traded for efficiency.

To sum up, the contributions of this paper are:

- A broad classification of probabilistic queries over uncertain data, based upon a flexible model of uncertainty;
- Techniques for evaluating probabilistic queries, including optimizations;
- Metrics for quantifying the quality of answers to probabilistic queries;
- Policies for improving the quality of answers to probabilistic queries under resource constraints.

The rest of this paper is organized as follows. In Section 2 we describe a general model of uncertainty, and the concept of probabilistic queries. Sections 3 and 4 discuss the algorithms for evaluating different kinds of probabilistic queries. Section 5 discusses quality metrics that are appropriate to these queries. Section 6 proposes update policies that improve the query answer quality. We present an experimental evaluation of the effectiveness of these update policies in Section 7. Section 8 discusses related work and Section 9 concludes the paper.

2. PROBABILISTIC QUERIES

In this section, we describe the model of uncertainty considered in this paper. This is a generic model, as it can accommodate a large number of application paradigms. Based on this model, we introduce a number of probabilistic queries.

2.1 Uncertainty Model

One popular model for uncertainty for a dynamic attribute is that at any point in time, the actual value is within a certain bound, d of its last reported value. If the actual value changes further than d, then the sensor reports its new value to the database and possibly changes d. For example, [14] describes a moving-object model where the location of an object is a dynamic attribute, and an object reports its location to the server if its deviation from the last reported location exceeds a threshold. Another model assumes that the attribute value changes with known speed, but the speed may change each time the value is reported. Other models include those that have no uncertainty. For example, in [4], the exact speed and direction of movement of each object are known. This model requires updates at the server whenever an object's speed or direction changes.

For the purpose of our discussion, the exact model of uncertainty is unimportant. All that is required is that at the time of query execution the range of possible values of the attribute of interest are known. We are interested in queries over some dynamic attribute, a, of a set of database objects, T. Also, we assume that a is a real-valued attribute, although our models and algorithms can be extended to other domains e.g., integer and coordinates easily. We denote the ith object of T by T_i and the value of attribute a of T_i by $T_i.a$ (where $i = 1, \ldots, |T|$). Throughout this paper, we treat $T_i.a$ as a continuous random variable. The uncertainty of $T_i.a$ can be characterized by the following two definitions (we use *pdf* to abbreviate the term "probability density function"):

Definition 1: An **uncertainty interval** of $T_i.a$ at time instant t, denoted by $U_i(t)$, is an interval $[l_i(t), u_i(t)]$ such that $l_i(t)$ and $u_i(t)$ are real-valued functions of t, and that the conditions $u_i(t) \geq l_i(t)$ and $T_i.a \in [l_i(t), u_i(t)]$ hold.

Definition 2: An **uncertainty pdf** of $T_i.a$ at time t, denoted by $f_i(x,t)$, is a pdf of random variable $T_i.a$, such that $f_i(x,t) = 0$ if $x \notin U_i(t)$.

Notice that since $f_i(x,t)$ is a pdf, it has the property that $\int_{l_i(t)}^{u_i(t)} f_i(x,t)dx = 1$. The above definition specifies the uncertainty of $T_i.a$ at time instant t in terms of a closed interval and the probability distribution of $T_i.a$. Notice that this definition does not specify how the uncertainty interval evolves over time, and what the nature of the pdf $f_i(x,t)$ is inside the uncertainty interval. The only requirement for $f_i(x,t)$ is that its value is 0 outside the uncertainty interval. Usually, the scope of uncertainty is determined by the last recorded value, the time elapsed since its last update, and some application-specific assumptions. For example, one may decide that $U_A(t)$ contains all the values within a distance of $(t - t_{update}) \times v$ from its last reported value, where t_{update} is the time that the last update was obtained, and v is the maximum rate of change of the value. One may also specify that $T_i.a$ is uniformly distributed inside the interval, i.e., $f_i(x,t) = 1/[u_i(t) - l_i(t)]$ for $x \in U_i(t)$, assuming that $u_i(t) > l_i(t)$. It should be noted that a uniform distribution represents the worst-case uncertainty (highest entropy) over a given interval.

2.2 Classification of Probabilistic Queries

We now present a classification of probabilistic queries and examples of common representative queries for each class. We identify two important dimensions for classifying database queries. First, queries can be classified according to the nature of the answers. An *entity-based* query returns a set of objects that satisfy the condition of the query. A *value-based* query returns a single value, examples of which include querying the value of a particular sensor, and computing the average value of a subset of sensor readings. The second property for classifying queries is whether aggregation is involved. We use the term aggregation loosely to refer to queries where interplay between objects determines the result. In the following definitions, we use the following naming convention: the first letter is either E (for entity-based queries) or V (for value-based queries).

1. Value-based Non-Aggregate Class

This is the simplest type of query in our discussions. It returns an attribute value of an object as the only answer, and involves no aggregate operators. One example of a probabilistic query for this class is the *VSingleQ*:

Definition 3: **Probabilistic Single Value Query (VSingleQ)** Given an object T_k, a VSingleQ returns l, u, $\{p(x) \mid x \in [l,u]\}$ where $l, u \in \Re$ with $l \leq u$, and $p(x)$ is the pdf of $T_k.a$ such that $p(x) = 0$ when $x < l$ or $x > u$.

An example of VSingleQ is "What is the wind speed recorded by sensor s_{22}?" Observe how this definition expresses the answer in terms of a bounded probabilistic value, instead of a single value. Also notice that $\int_l^u p(x)dx = 1$.

2. Entity-based Non-Aggregate Class

This type of query returns a set of objects, each of which satisfies the condition(s) of the query, independent of other objects. A typical example of this class is the ERQ:

Definition 4: **Probabilistic Range Query (ERQ)** Given a closed interval $[l,u]$, where $l, u \in \Re$ and $l \leq u$, an ERQ returns a set of tuples (T_i, p_i), where p_i is the non-zero probability that $T_i.a \in [l,u]$.

An *ERQ* returns a set of objects, augmented with probabilities, that satisfy the query interval.

3. Entity-based Aggregate Class

The third class of query returns a set of objects which satisfy an aggregate condition. We present the definitions of three typical queries for this class. The first two return objects with the minimum or maximum value of $T_i.a$:

Definition 5: **Probabilistic Minimum (Maximum) Query (EMinQ (EMaxQ))** An EMinQ (EMaxQ) returns a set R of tuples (T_i, p_i), where p_i is the non-zero probability that $T_i.a$ is the minimum (maximum) value of a among all objects in T.

A one-dimensional nearest neighbor query, which returns object(s) having a minimum absolute difference of $T_i.a$ and

Query Class	Entity-based	Value-based
Aggregate	ENNQ, EMinQ, EMaxQ	VAvgQ, VSumQ, VMinQ, VMaxQ
Non-Aggregate	ERQ	VSingleQ
Answer (Probabilistic)	$\{(T_i, p_i) \mid 1 \le i \le \lvert T\rvert \wedge p_i > 0\}$	$l, u, \{p(x) \mid x \in [l, u]\}$
Answer (Non-probabilistic)	$\{T_i \mid 1 \le i \le \lvert T\rvert\}$	$x \mid x \in \Re$

Table 1: Classification of Probabilistic Queries.

a given value q, is also defined:
Definition 6: **Probabilistic Nearest Neighbor Query (ENNQ)** Given a value $q \in \Re$, an ENNQ returns a set R of tuples (T_i, p_i), where p_i is the non-zero probability that $|T_i.a - q|$ is the minimum among all objects in T.

Note that for all the queries we defined in this class the condition $\sum_{T_i \in R} p_i = 1$ holds.

4. Value-based Aggregate Class

The final class involves aggregate operators that return a single value. Example queries for this class include:
Definition 7: **Probabilistic Average (Sum) Query (VAvgQ (VSumQ))** A VAvgQ (VSumQ) returns l, u, $\{p(x) \mid x \in [l, u]\}$, where $l, u \in \Re$ with $l \le u$, X is a random variable for the average (sum) of the values of a for all objects in T, and $p(x)$ is a pdf of X satisfying $p(x) = 0$ if $x < l$ or $x > u$.
Definition 8: **Probabilistic Minimum (Maximum) Value Query (VMinQ (VMaxQ))** A VMinQ (VMaxQ) returns l, u, $\{p(x) \mid x \in [l, u]\}$ where $l, u \in \Re$ with $l \le u$, X is a random variable for minimum (maximum) value of a among all objects in T, and $p(x)$ is a pdf of X satisfying $p(x) = 0$ if $x < u$ or $x > l$.

All these aggregate queries return answers in the form of a probabilistic distribution $p(x)$ in a closed interval $[l, u]$, such that $\int_l^u p(x)dx = 1$.

Table 1 summarizes the basic properties of the probabilistic queries discussed above. For illustrating the difference between probabilistic and non-probabilistic queries, the last row of the table lists the forms of answers expected if probability information is not augmented to the result of the queries e.g., the non-probabilistic version of EMaxQ is a query that returns object(s) with maximum values based only on the recorded values of $T_i.a$. It can be seen that the probabilistic queries provide more information on the answers than their non-probabilistic counterparts.

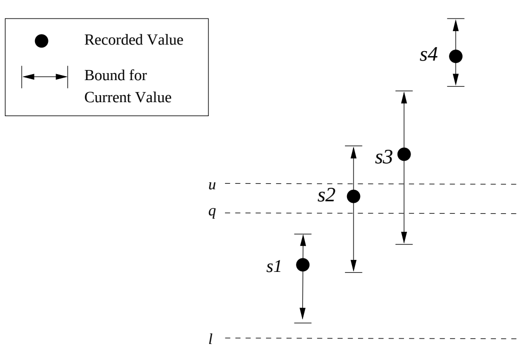

Figure 2: Illustrating the probabilistic queries.

Example. We illustrate the properties of the probabilistic queries with a simple example. In Figure 2, readings of four sensors s_1, s_2, s_3 and s_4, each with a different uncertainty interval, are being queried at time t_0. Suppose we have a set T of four database objects to record the information of these sensors, and each object has an attribute a to store the sensor reading. Further, assume that for each reading, its actual value has an even chance of being located at every point in its uncertainty interval, and the actual value has no chance of lying outside the interval i.e., $f_{s_1}(x, t_0), f_{s_2}(x, t_0), f_{s_3}(x, t_0), f_{s_4}(x, t_0)$ are bounded uniform distribution functions. Also let the uncertainty intervals from the readings of s_1, s_2, s_3 and s_4 at time t_0 be $[l_{s_1}, u_{s_1}]$, $[l_{s_2}, u_{s_2}]$, $[l_{s_3}, u_{s_3}]$ and $[l_{s_4}, u_{s_4}]$ respectively. A VSingleQ applied on s_4 at time t_0 will give us the result: $l_{s_4}, u_{s_4}, 1/(u_{s_4} - l_{s_4})$. When an ERQ (represented by the interval $[l, u]$) is invoked at time t_0 to find out how likely each reading is inside $[l, u]$, we see that the reading of s_1 is always inside the interval. It therefore has a probability of 1 for satisfying the ERQ. The reading of s_4 is always outside the rectangle, thus it has a probability of 0 of being located inside $[l, u]$. Since $U_{s_2}(t_0)$ and $U_{s_3}(t_0)$ partially overlap $[l, u]$, s_2 and s_3 have some chance of satisfying the query. In this example, the result of the ERQ is: $\{(s_1, 1), (s_2, 0.7), (s_3, 0.4)\}$.

In the same figure, an EMinQ is issued at time t_0. We observe that s_1 has a high probability of having the minimum value, because a large portion of the $U_{s_1}(t_0)$ has a smaller value than others. The reading of s_1 has a high chance of being located in this portion because $f_{s_1}(x, t)$ is a uniform distribution. The reading of s_4 does not have any chance of yielding the minimum value, since none of the values inside $U_{s_4}(t_0)$ is smaller than others. The result of the EMinQ for this example is: $\{(s_1, 0.7), (s_2, 0.2), (s_3, 0.1)\}$. On the other hand, an EMaxQ will return $\{(s_4, 1)\}$ as the only result, since every value in $U_{s_4}(t_0)$ is higher than any readings from any other sensors, and we are assured that s_4 yields the maximum value. An ENNQ with a query value q is also shown, where the results are: $\{(s_1, 0.2), (s_2, 0.5), (s_3, 0.3)\}$.

When a value-based aggregate query is applied to the scenario in Figure 2, a bounded pdf $p(x)$ is returned. If a VSumQ is issued, the result is a distribution in $[l_{s_1} + l_{s_2} + l_{s_3} + l_{s_4}, u_{s_1} + u_{s_2} + u_{s_3} + u_{s_4}]$; each x in this interval is the sum of the readings from the four sensors. The result of a VAvgQ is a pdf in $[(l_{s_1} + l_{s_2} + l_{s_3} + l_{s_4})/4, (u_{s_1} + u_{s_2} + u_{s_3} + u_{s_4})/4]$. The results of VMinQ and VMaxQ are probability distributions in $[l_{s_1}, u_{s_1}]$ and $[l_{s_4}, u_{s_4}]$ respectively, since only the values in these ranges have a non-zero probability value of satisfying the queries.

3. EVALUATING ENTITY- QUERIES

In this section we examine how the probabilistic entity-based queries introduced in the last section can be answered. We start with the discussion of an ERQ, followed by a more complex algorithm for answering an ENNQ. We also show how the algorithm for answering an ENNQ can be easily changed for EMinQ and EMaxQ.

3.1 Evaluation of ERQ

Recall that ERQ returns a set of tuples (T_i, p_i) where p_i is the non-zero probability that $T_i.a$ is within a given interval $[l, u]$. Let R be the set of tuples returned by the ERQ. The algorithm for evaluating the ERQ at time instant t is described in Figure 3.

---

1. $R \leftarrow \emptyset$
2. **for** $i \leftarrow 1$ **to** $|T|$ **do**
   (a) $OI \leftarrow U_i(t) \cap [l, u]$
   (b) **if** (width of $OI \neq 0$) **then**
      i. $p_i \leftarrow \int_{OI} f_i(x, t)dx$
      ii. **if** $p_i \neq 0$ **then** $R \leftarrow R \cup (T_i, p_i)$
3. **return** R

Figure 3: ERQ Algorithm.

In this algorithm, each object in T is checked once. To evaluate p_i for T_i, we first compute the amount overlapping interval OI of the two intervals: $U_i(t)$ and $[l, u]$ (Step 2a). If OI has zero width, we are assured that $T_i.a$ does not lie in $[l, u]$, and by the definition of ERQ, T_i will not be included in the result. Otherwise, we calculate the probability that $T_i.a$ is inside $[l, u]$ by integrating $f_i(x, t)$ over OI, and putting the result into R if $p_i \neq 0$ (Step 2b). The set of tuples (T_i, p_i), stored in R, are returned in Step 3.

3.2 Evaluation of ENNQ

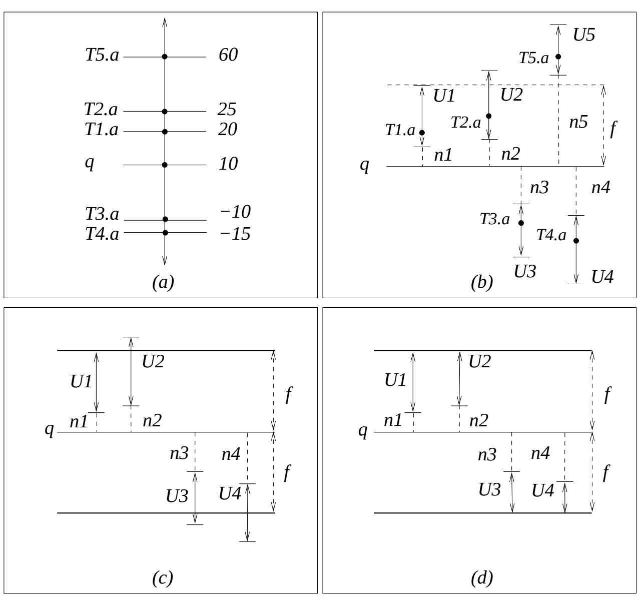

Figure 4: Phases of the ENNQ algorithm.

Processing an ENNQ involves evaluating the probability of the attribute a of each object T_i being the closest to (nearest-neighbor of) a value q. In general, this query can be applied to multiple attributes, such as coordinates. In particular, it could be a nearest-neighbor query for moving objects. Unlike the case of ERQ, we can no longer determine the probability for an object independent of the other objects. Recall that an ENNQ returns a set of tuples (T_i, p_i) where p_i denotes the non-zero probability that T_i has the minimum value of $|T_i.a - q|$. Let S be the set of objects to be considered by the ENNQ, and let R be the set of tuples returned by the query. The algorithm presented here consists of 4 phases: *projection*, *pruning*, *bounding* and *evaluation*. The first three phases filter out objects in T whose values of a have no chance of being the closest to q. The final phase, *evaluation*, is the core of our solution: for every object i that remains after the first three phases, the probability that $T_i.a$ is the nearest to q is computed.

1. **Projection Phase.**

In this phase, the uncertainty interval of each $T_i.a$ is computed based on the application's uncertainty model. Figure 4(a) shows the last recorded values of $T_i.a$ in S at time t_0, and the uncertainty intervals are shown in Figure 4(b).

---

1. **for** $i \leftarrow 1$ **to** $|S|$ **do**
   (a) **if** $(q \in U_i(t))$ **then** $N_i \leftarrow q$
   (b) **else**
      i. **if** $(|q - l_i(t)| < |q - u_i(t)|)$ **then** $N_i \leftarrow l_i(t)$
      ii. **else** $N_i \leftarrow u_i(t)$
   (c) **if** $(|q - l_i(t)| < |q - u_i(t)|)$ **then** $F_i \leftarrow u_i(t)$
   (d) **else** $F_i \leftarrow l_i(t)$
2. Let $f = min_{i=1,\ldots,|S|}|F_i - q|$ and $m = |S|$
3. **for** $i \leftarrow 1$ **to** m **do**
   if $(|N_i - q| > f)$ **then** $S \leftarrow S - T_i$
4. **return** S

Figure 5: Algorithm for the Pruning Phase.

2. **Pruning Phase.**

Consider two uncertainty intervals $U_1(t)$ and $U_2(t)$. If the smallest difference between $U_1(t)$ and q is longer than the largest difference between $U_2(t)$ and q, we can immediately conclude that T_1 is not a part of the answer to the ENNQ: even if the actual value of $T_2.a$ is as far as possible from q, $T_1.a$ still has no chance to be closer to q than $T_2.a$. Based on this observation, we can eliminate objects from T by the algorithm shown in Figure 5. In this algorithm, N_i and F_i record the closest and farthest possible values of $T_i.a$ to q, respectively. Steps 1(a) to 1(d) assign proper values to N_i and F_i. If q is inside interval $U_i(t)$, then N_i is taken as point q itself. Otherwise, N_i is either $l_i(t)$ or $u_i(t)$, depending on which value is closer to q. F_i is assigned in a similar manner. After this phase, S contains the minimal set of objects which must be considered by the query; any of them can have a value of $T_i.a$ closest to q. Figure 4(b) illustrates how this phase removes T_5, which is irrelevant to the ENNQ, from S.

3. **Bounding Phase.**

For each object in S, we only need to look at its portion of uncertainty interval located no farther than f from q. We do this conceptually by drawing a *bounding interval* B of length $2f$, centered at q. Any portion of the uncertainty interval outside B can be ignored. Figure 4(c) shows a bounding interval with length $2f$, and Figure 4(d) illustrates the result of this phase.

The phases we have just described attempt to reduce the number of objects to be evaluated, and derive tight bounds on the range of values to be considered.

4. **Evaluation Phase.**
Based on S and the bounding interval B, our aim is to calculate, for each object in S, the probability that it is the nearest neighbor of q. In the *pruning phase*, we have already found N_i, the point in $U_i(t)$ nearest to q. Let us call $|N_i - q|$ the *near_distance* of T_i, or n_i. Denote the interval with length $2r$, centered at q by $I_q(r)$. Also, let $P_i(r)$ be the probability that T_i is located inside $I_q(r)$, and $pr_i(r)$ be the pdf of R_i, where $R_i = |T_i.a - q|$. Figure 6 presents the algorithm for this phase.

Note that if $T_i.a$ has no uncertainty i.e., $U_i(t)$ is exactly equal to $T_i.a$, the evaluation phase algorithm needs to be modified. Our technical report [2] discusses how this algorithm can be changed to adapt to such situations. In the rest of this section, we will explain how the evaluation phase works, assuming non-zero uncertainty.

---

1. $R \leftarrow \emptyset$
2. Sort the elements in S in ascending order of n_i, and rename the sorted elements in S as $T_1, T_2, \ldots, T_{|S|}$
3. $n_{|S|+1} \leftarrow f$
4. **for** $i \leftarrow 1$ **to** $|S|$ **do**
   (a) $p_i \leftarrow 0$
   (b) **for** $j \leftarrow i$ **to** $|S|$ **do**
      i. $p \leftarrow \int_{n_j}^{n_{j+1}} pr_i(r) \cdot \prod_{k=1 \wedge k \neq i}^{j} (1 - P_k(r))\ dr$
      ii. $p_i \leftarrow p_i + p$
   (c) $R \leftarrow R \cup (T_i, p_i)$
5. **return** R

Figure 6: Algorithm for the Evaluation Phase.

Evaluation of $P_i(r)$ and $pr_i(r)$ To understand how the evaluation phase works, it is crucial to know how to obtain $P_i(r)$. As introduced before, $P_i(r)$ is the probability that $T_i.a$ is located inside $I_q(r)$. We illustrate the evaluation of $P_i(r)$ in Figure 7.

---

1. **if** $r \leq n_i$ **return** 0
2. **if** $r \geq |q - F_i|$, **return** 1
3. $O_i \leftarrow U_i(t) \cap I_q(r)$
4. **return** $\int_{O_i} f_i(x, t) dx$

Figure 7: Computation of $P_i(r)$.

We also need to compute $pr_i(r)$ – a pdf denoting $T_i.a$ equals either the upper or lower bound of $I_q(r)$. If $P_i(r)$ is a differentiable function of r, $pr_i(r)$ is the derivative of $P_i(r)$.
Evaluation of p_i. We can now explain how p_i, the probability that $T_i.a$ is closest to q, is computed. Let $Prob(r)$ be the probability that (1) $|T_i.a - q| = r$ and (2) $|T_i.a - q| = min_{k=1,\ldots,|S|} |T_k.a - q|$. Then Equation 1 outlines the structure of our solution:

$$p_i = \int_{n_i}^{f} Prob(r)\ dr \qquad (1)$$

The correctness of Equation 1 depends on whether it can correctly consider the probability that $T_i.a$ is the nearest neighbor for every possible value in the interval $U_i(t)$, and then sum up all those probability values. Recall that n_i represents the shortest distance from $U_i(t)$ to q, while $[q - f, q + f]$ is the bounding interval B, beyond which we do not need to consider. Equation 1 expands the width of the interval $I_q(r)$ from $2n_i$ to $2f$. Each value in $U_i(t)$ must therefore lie on either the upper bound or the lower bound of some $I_q(r)$, where $r \in [n_i, f]$. In other words, by gradually increasing the width of $I_q(r)$, we visit every value x in $U_i(t)$ and evaluates the probability that if $T_i.a = x$, then $T_i.a$ is the nearest-neighbor of q. We can rewrite the above formula by using $pr_i(r)$ and $P_k(r)$ (where $k \neq i$), as follows:

$$p_i = \int_{n_i}^{f} Prob(|T_i.a - q| = r) \cdot Prob(|T_k.a - q| > r) dr \qquad (2)$$

$$= \int_{n_i}^{f} p_i(r) \cdot \prod_{k=1 \wedge k \neq i}^{|S|} (1 - P_k(r))\ dr \qquad (3)$$

Observe that each $1 - P_k(r)$ term registers the probability that $T_k.a$ is farther from q than $T_i.a$.
Efficient Computation of p_i The computation time for p_i can be improved. Note that $P_k(r)$ has a value of 0 if $r \leq n_k$. This means when $r \leq n_k$, $1 - P_k(r)$ is always 1, and T_k has no effect on the computation of p_i. Instead of always considering $|S| - 1$ objects in the $\prod$ term of Equation 3 throughout $[n_i, f]$, we may actually consider fewer objects in some ranges, resulting in a better computation speed.

This can be achieved by first sorting the objects according to their *near_distance* from q. Next, the integration interval $[n_i, f]$ is broken down into a number of intervals, with end points defined by the *near_distance* of the objects. The probability of an object having a value of a closest to q is then evaluated for each interval in a way similar to Equation 3, except that we only consider $T_k.a$ with non-zero $P_k(r)$. Then p_i is equal to the sum of the probability values for all these intervals. The final formula for p_i becomes:

$$p_i = \sum_{j=i}^{|S|} \int_{n_j}^{n_{j+1}} pr_i(r) \cdot \prod_{k=1 \wedge k \neq i}^{j} (1 - P_k(r))\ dr \qquad (4)$$

Here we let $n_{|S|+1}$ be f for notational convenience. Instead of considering $|S| - 1$ objects in the $\prod$ term, Equation 4 only handles $j - 1$ objects in interval $[n_j, n_j + 1]$. This optimization is shown in Figure 6.
Example Let us use our previous example to illustrate how the evaluation phase works. After 4 objects $T_1, \ldots, T_4$ were captured (Figure 4(d)), Figure 8 shows the result after these objects have been sorted in ascending order of their *near_distance*, with the r-axis being the absolute difference of $T_i.a$ from q, and n_5 equals f. The probability p_i of each $T_i.a$ being the nearest neighbor of q is equal to the integral of the probability that $T_i.a$ is the nearest neighbor over the interval $[n_i, n_5]$.

Let us see how we evaluate uncertainty intervals when computing p_2. Equation 4 tells us that p_2 is evaluated by integrating over $[n_2, n_5]$. Since objects are sorted according

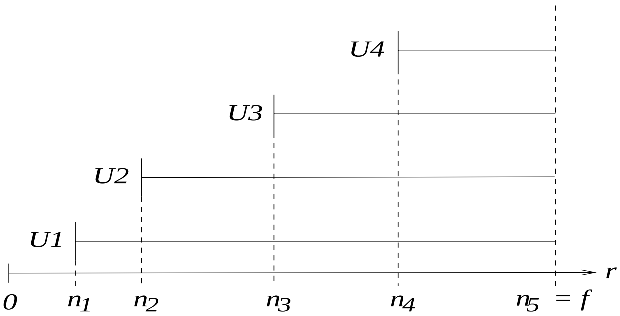

Figure 8: Illustrating the evaluation phase.

to n_i, we do not need to consider all 5 of them throughout $[n_2, n_5]$. Instead, we split $[n_2, n_5]$ into 3 sub-intervals, namely $[n_2, n_3]$, $[n_3, n_4]$ and $[n_4, n_5]$, and consider possibly fewer uncertainty intervals in each sub-interval. For example, in $[n_2, n_3]$, only U_1 and U_2 need to be considered.

3.3 Evaluation of EMinQ and EMaxQ

We can treat EMinQ and EMaxQ as special cases of ENNQ. In fact, answering an EMinQ is equivalent to answering an ENNQ with q equal to the minimum lower bound of all $U_i(t)$ in T. We can therefore modify the ENNQ algorithm to solve an EMinQ as follows: after the projection phase, we evaluate the minimum value of $l_i(t)$ among all uncertainty intervals. Then we set q to that value. We then obtain the results to the EMinQ after we execute the rest of the ENNQ algorithm. Solving an EMaxQ is symmetric to solving an EMinQ in which we set q to the maximum of $u_i(t)$ after the projection phase of ENNQ.

4. EVALUATING VALUE- QUERIES

In this section, we discuss how to answer the probabilistic value-based queries defined in Section 2.2.

4.1 Evaluation of VSingleQ

Evaluating a VSingleQ is simple, since by the definition of VSingleQ, only one object, T_k, needs to be considered. Suppose VSingleQ is executed at time t. Then the answer returned is the uncertainty information of $T_k.a$ at time t, i.e., $l_k(t)$, $u_k(t)$ and $\{f_k(x,t)|x \in [l_k(t), u_k(t)]\}$.

4.2 Evaluation of VSumQ and VAvgQ

Let us first consider the case where we want to find the sum of two uncertainty intervals $[l_1(t), u_1(t)]$ and $[l_2(t), u_2(t)]$ for objects T_1 and T_2. Notice that the values in the answer that have non-zero probability values lie in the range $[l_1(t) + l_2(t), u_1(t) + u_2(t)]$. For any x inside this interval, $p(x)$ (the pdf of random variable $X = T_1.a + T_2.a$) is:

$$\int_{max\{l_1(t), x-u_2(t)\}}^{min\{u_1(t), x-l_2(t)\}} f_1(y,t) f_2(x-y,t) dy \qquad (5)$$

The lower (upper) bound of the integration interval are evaluated according to the possible minimum (maximum) value of $T_1.a$.

We can generalize this result for summing the uncertainty intervals of $|T|$ objects by picking two intervals, summing them up using the above formula, and using the resulting interval to add to another interval. The process is repeated until we finish adding all the intervals. The resulting interval should have the following form:

$$[\sum_{i=1}^{|T|} l_i(t), \sum_{i=1}^{|T|} u_i(t)]$$

VAvgQ is essentially the same as VSumQ except for a division by the number of objects over which the aggregation is applied.

4.3 Evaluation of VMinQ and VMaxQ

To answer a VMinQ, we need to find a lower bound l, an upper bound u, and a pdf $p(x)$ where $p(x)$ is the pdf of the minimum value of a. Recall that (Section 3.3) to answer an EMinQ, we set q to be the minimum of the lower bound of the uncertainty intervals, and then obtain a bounding interval B in the bounding phase, within which we explore the uncertainty intervals to find the item with the minimum value. Interval B is exactly the interval $[l, u]$, since B determines the region where the possible minimum values are. In order to answer a VMinQ, we can use the first three phases of EMinQ (projection, pruning, bounding) to find B as the interval $[l, u]$. The evaluation phase is replaced by the algorithm shown in Figure 9.

1. $R \leftarrow \emptyset$
2. Sort the elements in S in ascending order of n_i, and rename the sorted elements in S as $T_1, T_2, \ldots, T_{|S|}$
3. $n_{|S|+1} \leftarrow f$
4. **for** $j \leftarrow 1$ to $|S|$ **do**
   (a) **for** $r \in [n_j, n_{j+1}]$ **do**
      i. $p \leftarrow 0$
      ii. **for** $i \leftarrow 1$ to j **do**
         $p \leftarrow p + pr_i(r) \cdot \prod_{k=1 \wedge k \neq i}^{j} (1 - P_k(r))$
      iii. $R \leftarrow R \cup (r, p)$
5. **return** $\{n_1, f, R\}$

Figure 9: Evaluation Phase of VMinQ.

Again, Steps 2 and 3 sort the objects in ascending order of n_i. After these steps, $B = [n_1, f]$ represents the range of possible values. Step 4 evaluates $p(r)$ for each value $r \in [n_1, f]$. Since we have sorted the uncertainty intervals in ascending order of n_i, we need not consider all $|T|$ objects throughout $[n_1, f]$. Specifically, for any value r in the interval $[n_j, n_{j+1}]$, we only need to consider objects $T_1, \ldots, T_j$ in evaluating $p(r)$. Hence for $r \in [n_j, n_{j+1}]$, $p(r)$ is given by the following formula:

$$p(r) = \sum_{i=1}^{j} (pr_i(r) \cdot \prod_{k=1 \wedge k \neq i}^{j} (1 - P_k(r))) \qquad (6)$$

The pair (r, p) represents $r, p(r)$. It is inserted into the set R in Step 4(a)iii. Finally, Step 5 returns the lower bound (n_1), upper bound (f) of the minimum value, and the distribution of the minimum values in $[n_1, f]$ (R). VMaxQ is handled in an analogous fashion.

5. QUALITY OF PROBABILISTIC RESULTS

In this section, we discuss several metrics for measuring the quality of the results returned by probabilistic queries. While other metrics (e.g., standard deviation) exist, the metrics described below measures the quality reasonably well. It is interesting to see that different metrics are suitable for different query classes.

5.1 Entity-Based Non-Aggregate Queries

For queries that belong to the entity-based non-aggregate query class, it suffices to define the quality metric for each (T_i, p_i) individually, independent of other tuples in the result. This is because whether an object satisfies the query or not is independent of the presence of other objects. We illustrate this point by explaining how the metric of ERQ is defined.

For an ERQ with query range $[l, u]$, the result is the best if we are sure either $T_i.a$ is completely inside or outside $[l, u]$. Uncertainty arises when we are less than 100% sure whether the value of $T_i.a$ is inside $[l, u]$. We are confident that $T_i.a$ is inside $[l, u]$ if a large part of $U_i(t)$ overlaps $[l, u]$ i.e., p_i is large. Likewise, we are also confident that $T_i.a$ is outside $[l, u]$ if only a very small portion of $U_i(t)$ overlaps $[l, u]$ i.e., p_i is small. The worst case happens when p_i is 0.5, where we cannot tell if $T_i.a$ satisfies the range query or not. Hence a reasonable metric for the quality of p_i is:

$$\frac{|p_i - 0.5|}{0.5} \quad (7)$$

In Equation 7, we measure the difference between p_i and 0.5. Its highest value, which equals 1, is obtained when p_i equals 0 or 1, and its lowest value, which equals 0, occurs when p_i equals 0.5. Hence the value of Equation 7 varies between 0 to 1, and a large value represents good quality. Let us now define the *score* of an ERQ:

$$\text{Score of an ERQ} = \frac{1}{|R|} \sum_{i \in R} \frac{|p_i - 0.5|}{0.5} \quad (8)$$

where R is the set of tuples $(T_i.a, p_i)$ returned by an ERQ. Essentially, Equation 8 evaluates the average over all tuples in R.

Notice that in defining the metric of ERQ, Equation 7 is defined for each T_i, disregarding other objects. In general, to define quality metrics for the entity-based non-aggregate query class, we can define the quality of each object individually. The overall score can then be obtained by averaging the quality value for each object.

5.2 Entity-Based Aggregate Queries

Contrary to an entity-based non-aggregate query, we observe that for an entity-based aggregate query, whether an object appears in the result depends on the existence of other objects. For example, consider the following two sets of answers to an EMinQ: $\{(T_1.a, 0.6), (T_2.a, 0.4)\}$ and $\{(T_1.a, 0.6), (T_2.a, 0.3), (T_3.a, 0.1)\}$. How can we tell which answer is better? We identify two important components of quality for this class: entropy and interval width.

Entropy. Let $X_1, \ldots, X_n$ be all possible messages, with respective probabilities $p(X_1), \ldots, p(X_n)$ such that $\sum_{i=1}^{n} p(X_i) = 1$. The entropy of a message $X \in \{X_1, \ldots, X_n\}$ is:

$$H(X) = \sum_{i=1}^{n} p(X_i) \log_2 \frac{1}{p(X_i)} \quad (9)$$

The entropy, $H(X)$, measures the average number of bits required to encode X, or the amount of information carried in X [11]. If $H(X)$ equals 0, there exists some i such that $p(X_i) = 1$, and we are certain that X_i is the message, and there is no uncertainty associated with X. On the other hand, $H(X)$ attains the maximum value when all the messages are equally likely, in which case $H(X)$ equals $\log_2 n$.

Recall that the result to the queries we defined in this class is returned in a set R consisting of tuples (T_i, p_i). We can view R as a set of messages, each of which has a probability p_i. Moreover, the property that $\sum_{i=1}^{n} p_i = 1$ holds. Then $H(R)$ measures the uncertainty of the answer to these queries; the lower the value of $H(R)$, the more certain is the answer.

Bounding Interval. Uncertainty of an answer also depends on another important factor: the bounding interval B. Recall that before evaluating one of these aggregate queries, we need to find B that indicates all possible values we have to consider. Then we consider all the portions of uncertainty intervals that lie within B. Note that the decision of which object satisfies the query is only made within this interval. Also notice that the width of B is determined by the width of the uncertainty intervals associated with objects; a large width of B is the result of large uncertainty intervals. Therefore, if B is small, it indicates that the uncertainty intervals of objects that participate in the final result of the query are also small. In the extreme case, when the uncertainty intervals of participant objects have zero width, then the width of B is zero too. The width of B therefore gives us a good indicator of how uncertain a query answer is.

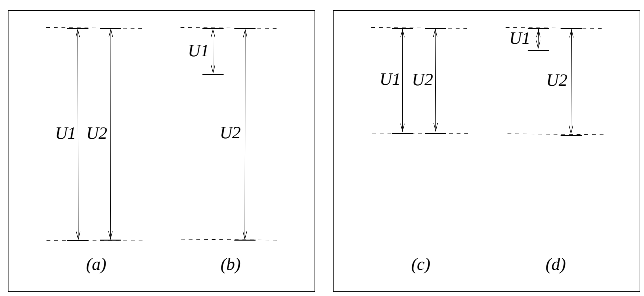

Figure 10: Illustrating how the entropy and the width of B affect the quality of answers for entity-based aggregate queries. The four figures show the uncertainty intervals ($U_1(t_0)$ and $U_2(t_0)$) inside B after the bounding phase. Within the same bounding interval, (b) has a lower entropy than (a), and (d) has a lower entropy than (c). However, both (c) and (d) have less uncertainty than (a) and (b) because of smaller bounding intervals.

An example at this point will make our discussions clear. Figure 10 shows four different scenarios of two uncertainty intervals, $U_1(t_0)$ and $U_2(t_0)$, after the bounding phase for an EMinQ. We can see that in (a), $U_1(t_0)$ is the same as $U_2(t_0)$. If we assume uniform distribution for both uncertainty intervals, both T_1 and T_2 will have equal probability of having the minimum value of a. In (b), it is obvious that T_2 has a much greater chance than T_1 to have the minimum value of a. Using Equation 9, we can observe that the answer in (b) enjoys a lower degree of uncertainty than (a). In (c) and (d), all the uncertainty intervals are halved of those in (a) and

(b) respectively. Hence (d) still has a lower entropy value than (c). However, since the uncertainty intervals in (c) and (d) are reduced, their answers should be more certain than those of (a) and (b). Notice that the widths of B for (c) and (d) are all less than (a) and (b).

The quality of entity-based aggregate queries is thus decided by two factors: (1) entropy $H(R)$ of the result set, and (2) width of B. Their *score* is defined as follows:

$$\text{Score of Entity, Aggr Query} = -H(R) \cdot \text{width of } B \quad (10)$$

Notice that the query answer gets a high score if either $H(R)$ is low, or the width of B is low. In particular, if either $H(R)$ or the width of B is zero, then $H(R) = 0$ is the maximum score.

5.3 Value-Based Queries

Recall that the results returned by value-based queries are all in the form $l, u, \{p(x) \mid x \in [l, u]\}$, i.e., a probability distribution of values in interval $[l, u]$. To measure the quality of such queries, we can use the concept of *differential entropy*, defined as follows:

$$\hat{H}(X) = -\int_{l}^{u} p(x) \log_2 p(x) dx \quad (11)$$

where $\hat{H}(X)$ is the differential entropy of a continuous random variable X with probability density function $p(x)$ defined in the interval $[l, u]$ [11]. Similar to the notion of entropy, $\hat{H}(X)$ measures the uncertainty associated with the value of X. Moreover, X attains the maximum value, $log_2(u - l)$ when X is uniformly distributed in $[l, u]$. When $u - l = 1$, $\hat{H}(X) = 0$. Therefore, if a random variable has more uncertainty than the uniform distribution in $[0, 1]$, it will have a positive entropy value; otherwise, it will have a negative entropy value.

We use the notion of differential entropy to measure the quality of value-based queries. Specifically, we apply Equation 11 to $p(x)$ as a measure of how much uncertainty is inherent to the answer of a value-based query. The lower the differential entropy value, the more certain is the answer, and hence the better quality is the answer. In particular, if there is a value y in $[l, u]$ such that the value of $p(y)$ is high, then the entropy will be low. We now define the *score* of a probabilistic value-based query:

$$\text{Score of a Value-Based Query} = -\hat{H}(X) \quad (12)$$

The quality of a value-based query can thus be measured by the uncertainty associated with its result: the lower the uncertainty, the higher score can be obtained as indicated by Equation 12.

6. IMPROVING ANSWER QUALITY

In this section, we discuss several update policies that can be used to improve the quality of probabilistic queries, defined in the last section. We assume that the sensors cooperate with the central server i.e., a sensor can respond to update requests from the sensor by sending the newest value to the server, as in the system model described in [9].

Suppose after the execution of a probabilistic query, some slack time is available for the query. The server can improve the quality of the answers to that query by requesting updates from sensors, so that the uncertainty intervals of some sensor data are reduced, potentially resulting in an improvement of the answer quality. Ideally, a system can demand updates from all sensors involved in the query; however, this is not practical in a limited-bandwidth environment. The issue is, therefore, to improve the quality with as few updates as possible. Depending on the types of queries, we propose a number of update policies.

Improving the Quality of ERQ The policy for choosing objects to update for an ERQ is very simple: choose the object with the minimum value computed in Formula 7, with an attempt to improve the score of ERQ.

Improving the Quality of Other Queries Several update policies are proposed for queries other than ERQ:

1. **Glb_RR.** This policy updates the database in a round-robin fashion using the available bandwidth i.e., it updates the data items one by one, making sure that each item gets a fair chance of being refreshed.

2. **Loc_RR.** This policy is similar to Glb_RR, except that the round-robin policy is applied only to the data items that are related to the query, e.g., the set of objects with uncertainty intervals overlapping the bounding interval of an EMinQ.

3. **MinMin.** An object with its lower bound of the uncertainty interval equal to the lower bound of B is chosen for update. This attempts to reduce the width of B and improve the score.

4. **MaxUnc.** This heuristic simply chooses the uncertainty interval with the maximum width to update, with an attempt to reduce the overlapping of the uncertainty intervals.

5. **MinExpEntropy.** Another heuristic is to check, for each $T_i.a$ that overlaps B, the effect to the entropy if we choose to update the value of $T_i.a$. Suppose once $T_i.a$ is updated, its uncertainty interval will shrink to a single value. The new uncertainty is then a point in the uncertainty interval before the update. For each value in the uncertainty interval before the update, we evaluate the entropy, assuming that $U_i(t)$ shrinks to that value after the update. The mean of these entropy values is then computed. The object that yields the minimum expected entropy is updated.

7. EXPERIMENTAL RESULTS

In this section, we experimentally study the relative behaviors of the various update policies described above, with respect to improving the quality of the query results. We will discuss the simulation model followed by the results.

7.1 Simulation Model

The evaluation is conducted using a discrete event simulation representing a server with a fixed network bandwidth ($\mathcal{B}$ messages per second) and 1000 sensors. Each update from a sensor updates the value and the uncertainty interval for the sensor stored at the server. Our uncertainty model is as follows: An update from sensor T_i at time t_{update} specifies the current value of the sensor, $T_i.a_{srv}$, and the rate, $T_i.r_{srv}$ at which the uncertainty region (centered at $T_i.a_{srv}$) grows. Thus at any time instant, t, following the update, the uncertainty interval ($U_i(t)$) of sensor T_i is given by $T_i.a_{srv} \pm T_i.r_{srv} \times (t - T_i.t_{update})$. The distribution of values within this interval is assumed to be uniform.

The actual values of the sensors are modeled as random walks within the normalized domain as in [9]. The maximum rate of change of individual sensors are uniformly distributed between 0 and R_{max}. At any time instant, the value of a sensor lies within its current uncertainty interval specified by the last update sent to the server. An update from the sensor is necessitated when a sensor is close to the edge of its current uncertainty region. Additionally, in order to avoid excessively large levels of uncertainty, an update is sent if either the total size of the uncertainty region or the time since the last update exceed threshold values.

The representative experiments presented considered either EMinQ or VMinQ queries only. In each experiment the queries arrive at the server following a Poisson distribution with arrival rate λ_q. Each query is executed over a subset of the sensors. The subsets are selected randomly following the 80-20 hot-cold distribution (20% of the sensors are selected 80% of the time). The cardinality of each set was fixed at $N_{sub} = 100$. The maximum number of concurrent queries was limited to $N_q = 10$. Each query is allowed to request at most N_{msg} updates from sensors in order to improve the quality of its result.

In order to study different aspects of the policies, query termination can be specified either as (i) a fixed time interval (T_{active}) after which the query is completed even if its requested updates have not arrived (due to network congestion) or (ii) when a target quality ($\mathcal{G}$) is achieved. Depending upon the policy, we study either the average achieved quality (score), the average size of the uncertainty region, or the average response time needed to achieve the desired quality. All measurements were made after a suitable warm-up period had elapsed. For fairness of comparison, in each experiment, the arrival of queries as well as the changes to the sensor values were identical.

Table 2 summarizes the major parameters and their default values. The simulation parameters were chosen such that average cardinality of the result sets achieved by the best update policies was between 3 and 10.

Param	Default	Meaning
$\mathcal{D}$	$[0, 1]$	Domain of attribute a
R_{max}	0.1	Maximum rate of change of a (sec^{-1})
N_q	10	Maximum # of concurrent queries
λ_q	20	Query arrival rate (query/sec)
N_{sub}	100	Cardinality of query subset
T_{active}	5	Query active time (sec)
$\mathcal{B}$	350	Network bandwidth (msg/sec)
N_{msg}	5	Maximum # of updates per query
N_{conc}	1	The # of concurrent updates per query

Table 2: Simulation parameters and default values.

7.2 Results

Due to limited space, we only show the most important experimental results. Interested readers are referred to our technical report [2] for more detailed discussions of our experiments. All figures in this section show averages.

Bandwidth. Figure 11 shows scores for EMinQ achieved by various update policies for different values of bandwidth.

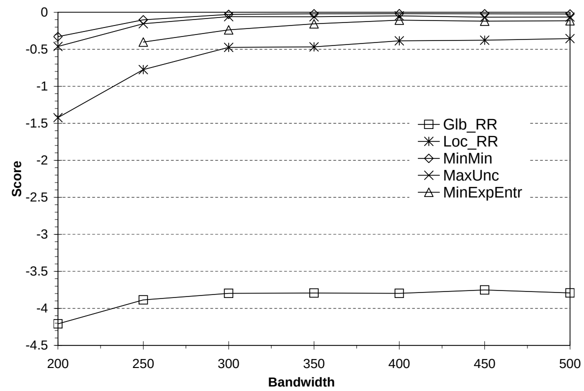

Figure 11: EMinQ score as function of $\mathcal{B}$

The quality metric in this case is negated entropy times the size of the uncertainty region of the result set. Figure 12 is analogous to Figure 11 but shows scores for VMinQ instead of EMinQ. The score for VMinQ queries is negated continuous entropy.

In Figures 11 and 12, the scores increase as bandwidth increases for all policies, approaching the perfect score of zero for EMinQ. This is explained by the fact that with higher bandwidth the updates requested by the queries are received faster. Thus for higher bandwidth the uncertainty regions for freshly updated sensors tend to be smaller than those using lower bandwidth. Smaller uncertainty regions translate into smaller uncertainty of the result set, and consequently higher score. The reduction in uncertainty regions with increasing bandwidth can be observed from Figure 13.

All schemes that favor updates for sensors being queried significantly outperform the only scheme that ignores this information: Glb_RR. The best performance is achieved by the MinMin policy, which updates a sensor with the lower bound of the uncertainty region $l_i(t)$ equal to the minimum lower bound among all sensors considered by the query. The MinExpEntropy policy showed worse results than the MinMin and MaxUnc policies in Figures 11 and 13 and worse results than those of the MinMin policy for VMinQ queries, Figure 12. When comparing the MinMin and MaxUnc policies, the better score of the MinMin policy is explained by the fact that the sensor picked for an update by the MinMin policy tends to have large uncertainty too – in fact, the uncertainty interval is at least as large as the width of the bounding interval. In addition the value of its attribute a tends to have higher probability of being minimum.

Response Time. Figure 14 shows response time as a function of available bandwidth for EMinQ. Unlike the other experiments, in this experiment a query execution is stopped as soon as the goal score $\mathcal{G}$ (-0.06) is reached. Once again the MinMin strategy showed the best results, reaching the goal score faster than the other policies. The difference in response time is especially noticeable for smaller values of bandwidth, where it is almost twice as good as the other strategies. Predictably, the response time decreases when more bandwidth becomes available.

Arrival Rate. Figures 15 and 16 show the scores achieved by EMinQ and VMinQ queries for various update policies as a function of query arrival rate λ_q. As λ_q increases from 5 to 25, more queries request updates and reduce the uncertainty

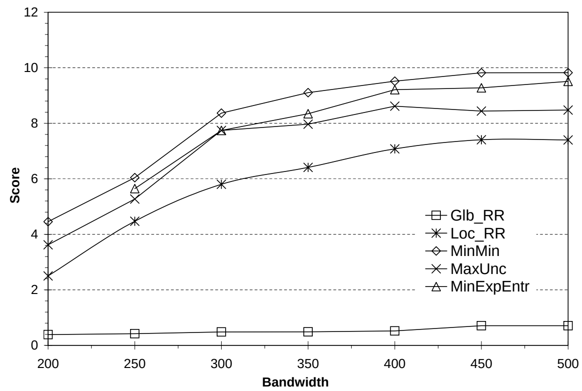

Figure 12: VMinQ score as function of $\mathcal{B}$

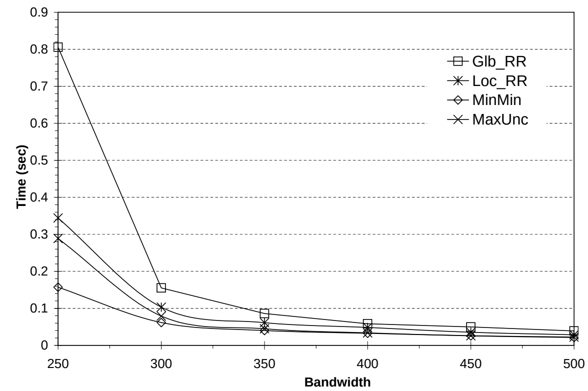

Figure 14: Response time as function of $\mathcal{B}$

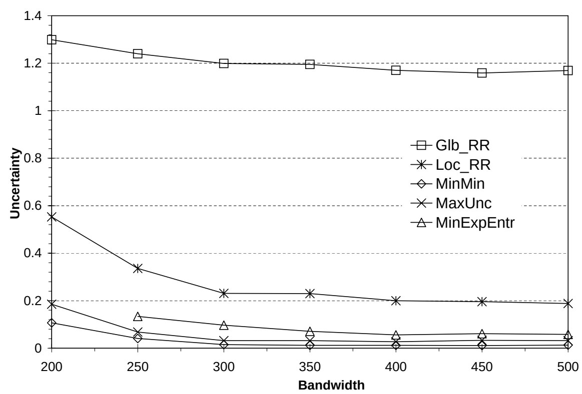

Figure 13: Uncertainty as function of $\mathcal{B}$

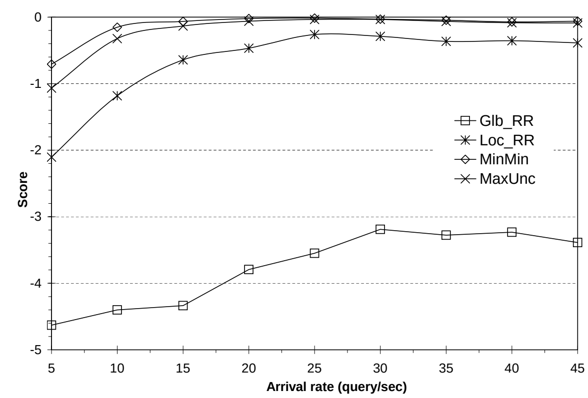

Figure 15: EMinQ score as function of λ_q

regions. As a result, the uncertainty decreases, which leads to better scores (Figure 17). When λ_q reaches 25 the entire network bandwidth is utilized. As λ_q continue to increase queries are able to send fewer requests for updates and receive fewer updates in time, leading to poor result quality and larger uncertainty.

We can observe from Figures 15, 16, and 17 that the relative performance of the various policies remains the same over a wide range of arrival rates ($\lambda_q \in [5, 45]$).

The experiments show that all policies that favor query-based updates achieve much higher levels of quality. For the queries considered, the MinMin policy gives the best performance. We plan to evaluate policies for all types of queries as part of future work.

8. RELATED WORK

Many researchers have studied approximate answers to queries based on a subset of data. In [13], Vrbsky et. al studied approximate answers for set-valued queries (where a query answer contains a set of objects) and single-valued queries (where a query answer contains a single value). An exact answer E can be approximated by two sets: a *certain set* C which is the subset of E, and a *possible set* P such that $C \cup P$ is a superset of E. Other techniques like sampling [5] and synopses [1] are used to produce statistical results. While these efforts investigate approximate answers based upon a subset of the (exact) values of the data, our work addresses probabilistic answers based upon all the (imprecise) values of the data.

The problem of balancing the tradeoff between precision and performance for querying replicated data was studied by Olston et. al [8, 7, 9]. In their model, the cache in the server cannot keep track of the exact values of sensor sources due to limited network bandwidth. Instead of storing the actual value in the server's cache, an interval for each item is stored, within which the current value must be located. A query is then answered by using these intervals, together with the actual values fetched from the sources. In [8], the problem of minimizing the update cost within an error bound specified by aggregate queries is studied. In [7], algorithms for tuning the intervals of the data items stored in the cache for best performance are proposed. In [9], the problem of minimizing the divergence between the server and the sources given a limited amount of bandwidth is discussed.

Khanna et. al [6] extend Olston's work by proposing an online algorithm that identifies a set of elements with minimum update cost so that a query can be answered within an error bound. Three models of precision are discussed. In the absolute (relative) precision model, an answer a is called α-precise if the actual value v deviates from a by not more than an additive (multiplicative) factor of α. The rank precision model is used to deal with selection problems which identifies an element of rank r: an answer a is called α-precise if the rank of a lies in the interval $[r - \alpha, r + \alpha]$.

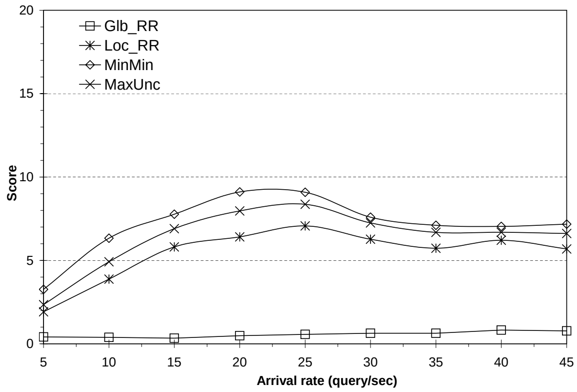

Figure 16: VMinQ score as function of λ_q

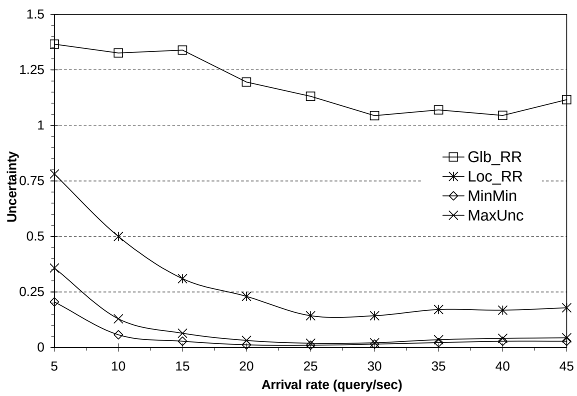

Figure 17: Uncertainty as function of λ_q

In all the works that we have discussed, the use of probability distribution of values inside the uncertainty interval as a tool for quantifying uncertainty has not been considered. Discussions of queries on uncertainty data were often limited to the scope of aggregate functions. In contrast, our work adopts the notion of probability and provides a paradigm for answering general queries involving uncertainty. We also define the quality of probabilistic query results which, to the best of our knowledge, has not been addressed.

Except [14] and [3], we are unaware of any other work that discusses the evaluation of a query answer in probabilistic form. These two studies only consider probabilistic range queries and nearest-neighbor queries in a moving-object database model. Velocity-constrained indexing was proposed in [10] for efficient indexing of moving objects.

9. CONCLUSIONS

In this paper we studied the problem of augmenting probability information to queries over uncertain data. We propose a flexible model of uncertainty, which is defined by (1) an lower and upper bound, and (2) a pdf of the values inside the bounds. We then explain, from the viewpoint of a probabilistic query, we can classify queries in two dimensions, based on whether they are aggregate/non-aggregate queries, and whether they are entity-based/value-based. Algorithms for computing typical queries in each query class are demonstrated. We present novel metrics for measuring quality of answers to these queries, and also discuss several update heuristics for improving the quality of results. The benefit of query-based updates was shown experimentally.

10. REFERENCES

[1] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *Proc. of the ACM SIGMOD 1999 Intl. Conf. on Management of Data*, pages 275–286, 1999.

[2] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. Technical Report TR 02-020, Department of Computer Science, Purdue University, October 2002.

[3] R. Cheng, S. Prabhakar, and D. V. Kalashnikov. Querying imprecise data in moving object environments. In *Proc. of the 19th IEEE Intl. Conf. on Data Engineering*, India, 2003.

[4] D.Pfoser and C.S.Jensen. Querying the trajectories of on-line mobile objects. In *ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE) 2001*, pages 66–73, 2001.

[5] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *Proc. of the ACM SIGMOD 1998 Intl. Conf. on Management of Data*, 1998.

[6] S. Khanna and W.C. Tan. On computing functions with uncertainty. In *20th ACM Symposium on Principles of Database Systems*, 2001.

[7] C. Olston, Boon Thau Loo, and J. Widom. Adaptive precision setting for cached approximate values. In *Proc. of the ACM SIGMOD 2001 Intl. Conf. on Management of Data*, 2001.

[8] C. Olston and J. Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *Proc. of the 26th Intl. Conf. on Very Large Data Bases*, 2000.

[9] C. Olston and J. Widom. Best-effort cache synchronization with source cooperation. In *Proc. of the ACM SIGMOD 2002 Intl. Conf. on Management of Data*, pages 73–84, 2002.

[10] S. Prabhakar, Y. Xia, D. Kalashnikov, W. Aref, and S. Hambrusch. Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects. *IEEE Transactions on Computers, Special section on data management and mobile computing*, 51(10), 2002.

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

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

[13] S. V. Vrbsky and J. W. S. Liu. Producing approximate answers to set- and single-valued queries. *The Journal of Systems and Software*, 27(3), 1994.

[14] O. Wolfson, P. Sistla, S. Chamberlain, and Y. Yesha. Updating and querying databases that track mobile units. *Distributed and Parallel Databases*, 7(3), 1999.