Query Selectivity Estimation for Uncertain Data

Sarvjeet Singh, Chris Mayfield, Rahul Shah*, Sunil Prabhakar,
and Susanne Hambrusch

Department of Computer Science, Purdue University
West Lafayette, IN 47907, USA
{sarvjeet,cmayfiel,rahul,sunil,seh}@cs.purdue.edu

Abstract. Applications requiring the handling of uncertain data have led to the development of database management systems extending the scope of relational databases to include uncertain (probabilistic) data as a native data type. New automatic query optimizations having the ability to estimate the cost of execution of a given query plan, as available in existing databases, need to be developed. For probabilistic data this involves providing selectivity estimations that can handle multiple values for each attribute and also new query types with threshold values. This paper presents novel selectivity estimation functions for uncertain data and shows how these functions can be integrated into PostgreSQL to achieve query optimization for probabilistic queries over uncertain data. The proposed methods are able to handle both attribute- and tuple-uncertainty. Our experimental results show that our algorithms are efficient and give good selectivity estimates with low space-time overhead.

1 Introduction

Recently there has been a surge in interest in managing probabilistic data in relational databases [1,2,3,4,5,6]. This interest is engendered by the needs of numerous applications including scientific data management, data integration, sensor databases, data cleaning, text processing and location-based services. The relational database model has very little support for uncertain data, limited to the use of NULL values. The nature of uncertainty in many applications is such that it is necessary to store alternative values for tuples, or attributes and process probabilistic queries over this data.

Several models have been proposed for extending the scope of relational databases to include uncertain (probabilistic) data as a native data type. These models define new semantics for query processing over uncertain data. The results of these queries are typically probabilistic in nature. Since results with a low probability of occurrence are generally less interesting than higher probability answers, an important new class of *threshold* queries has been identified [7]. These queries return only those answers that have a probability exceeding a threshold. While this thresholding weeds out less relevant answers, it also opens up possibilities for query optimization. There has been some recent work on efficient processing of threshold queries over uncertain data [8]. This work has largely focused on indexing methods to improve query performance.

* Work done while at Purdue University. Current affiliation: Louisiana State University, Baton Rouge, Louisiana, USA.

B. Ludäscher and Nikos Mamoulis (Eds.): SSDBM 2008, LNCS 5069, pp. 61–78, 2008.

The long-term goal for several projects is the development of novel database management systems that natively handle uncertain data. An important step in this direction is the development of automatic query optimization as is available in existing databases. Toward this end, an essential ingredient is the ability to estimate the cost of execution of a given query plan. For probabilistic data this would involve providing selectivity estimates for probabilistic operators. Currently, there is no work on providing such selectivity estimation functions for probabilistic data. With the availability of these estimation functions it is possible to use existing query optimization techniques that are already built into databases to handle the case of probabilistic data.

In this paper we address this problem and develop novel selectivity estimation functions for uncertain data. We also show how these functions can be integrated into PostgreSQL to achieve query optimization for probabilistic queries over uncertain data. Selectivity estimation for uncertain data needs to handle multiple values for each attribute and also novel query types with threshold values. Furthermore, an important type of uncertainty transforms a single attribute value to a continuous distribution – this is especially common in sensor databases [9]. The existing cost estimation methods are therefore not applicable for this domain.

The goal of this paper is to handle selectivity estimation for the two main types of uncertainty that have been proposed in recent work: *tuple uncertainty* [1,2] and *attribute uncertainty* [7]. To demonstrate the effectiveness of our selectivity estimation techniques, we have used an open-source database management system for uncertain data called Orion [3] which is built into PostgreSQL.

The major contributions of this paper are as follows:

- We have developed efficient algorithms for selectivity estimation of probabilistic threshold queries over uncertain data.
- We have implemented these algorithms in a real database system.
- Our experimental results show that the algorithms are efficient and provide good estimates for query selectivities.

The rest of this paper is organized as follows. Section 2 summarizes the related work done in this area. We formally describe the uncertainty model and probabilistic queries in Section 3. Our algorithms for selectivity estimation are presented in Section 4. We present the experimental results in Section 5, and Section 6 concludes this paper.

2 Related Work

There is a rich body of work on selectivity estimation for traditional relational database management systems. Most approaches for selectivity estimation on precise data use histograms. Poosala et al [10] proposed a taxonomy to capture all previously proposed histogram approaches. These approaches are not applicable for uncertain data because both the queries and the underlying data types for uncertain data differ greatly from traditional data and queries.

More recently, there has been a great deal of work on the development of models for representing uncertainty in databases. Two main approaches for modeling uncertain data have emerged in this field: Tuple uncertainty [1,2] and Attribute uncertainty [7].

Similar models have been proposed in moving-object environments [11] and in sensor networks [9]. Several systems that handle such uncertainty in data have been recently proposed (Orion [3], MayBMS [12], Mystiq [13], Trio [14], [4]). This probabilistic modeling of data has also been extended to semi-structured data [15] and XML [16].

Efficient evaluation of probabilistic range queries is discussed in [2,6,7,9,11]. Probabilistic nearest-neighbor queries are presented in [7,17]. An index called Probabilistic Threshold Index was proposed in [18] that can be used to efficiently execute some classes of probabilistic queries.

To best of our knowledge, the issue of selectivity estimation for queries over probabilistic data has not been addressed before.

3 Uncertainty Model

To model the uncertainty present in a data item, a data scheme known as the *Attribute uncertainty model* was proposed in [7]. This scheme assumes that individual attributes, as opposed to complete tuples, are uncertain. The attribute uncertainty model assumes that each data item can be represented by a range of possible values along with the distribution of values over this range. Formally, assume that each tuple of interest consists of an uncertain attribute a. If there are more than one uncertain attributes within the same tuple, they are assumed to be independent of each other. The domain of the uncertain attribute can be continuous (e.g. real-valued) or discrete (e.g. integer). The probabilistic uncertainty of a continuous attribute a consists of two components:

1. **Uncertainty Interval:** The *uncertainty interval* of an item a, denoted by U_a, is an interval $[l_a, r_a]$ where $l_a, r_a \in \Re, r_a \geq l_a$ and $a \in U_a$. The range of R_a of a is defined as $R_a = r_a - l_a$.
2. **Uncertainty pdf:** The *uncertainty pdf* of a, denoted by $f_a(x)$ is a *probability distribution function* (pdf) of a where $f_a(x) = 0$ if $x \notin U_a$.

In addition to the pdf $f_a(x)$, we can also define a *cumulative distribution function* (cdf) $F_a(x)$, which is defined as $F_a(x) = \int_{-\infty}^{x} f_a(x)dx$. Note that, similar to the continuous case, we can also define the pdf and cdf functions in case of a *discrete attribute* by replacing the integral with a sum in the above definitions.

The tuple uncertainty model [1,2,19] assumes that the complete tuple is uncertain. A probability value is attached to each tuple which represents the probability of that tuple being present in the database. In addition, multiple tuples can be grouped together to form an *x-tuple* [1]. The tuples present inside a *x-tuple* are called alternatives and they represent mutually exclusive values for the tuple.

The goal of this paper is to propose estimation solutions that are applicable to both models of uncertainty: attribute and tuple. For our purposes, we are interested in a single attribute at a time, a (real-valued or integer), for which we are estimating the selectivity. Thus, we can ignore the intra-tuple dependencies. We assume that the uncertainty in the data can be captured in terms of attribute uncertainty. In other words, for the attribute in question, we are able to generate a pdf (f_a) and cdf (F_a) for each tuple of the relation. This is directly available from the attribute uncertainty model. For the case of tuple uncertainty, there are two cases to consider. The first is if there are no x-tuples. In

this case, each tuple has a probability value associated with it and is independent of any other tuple. For this case, the pdf for each tuple is simply the single attribute value along with the associated tuple probability. In the second case, the x-tuple itself provides multiple alternatives for the given attribute along with associated probabilities. These are collapsed into a single attribute uncertainty (discrete) pdf.

3.1 Operators and Threshold Queries

A number of operators are defined in [8] for comparing uncertain values with both uncertain and certain (precise) values. This paper focuses on selection queries that compare an uncertain value with precise values. For these queries, we present the definitions for comparing uncertain with certain data. Operators between an uncertain value a and a certain value $v \in \Re$ can be defined as:

$$Pr(a < v) = \int_{-\infty}^{v} f_a(x)dx = F_a(v)$$
$$Pr(a > v) = 1 - F_a(v)$$

The set of queries that we consider in the paper are called *Probabilistic Threshold Range Queries* and were proposed in [18]. These queries are a variant of probabilistic queries where only answers with probability values over a certain threshold τ are returned. With this concept, all the operators discussed above can be changed into boolean predicates by adding a probability threshold to them.

4 Selectivity Estimation

In this section we describe various techniques that can be used for estimating the selectivity for a given probabilistic threshold operator.

4.1 Unbounded Range Queries

This approach is based on mapping the uncertain attribute values to a 2-D histogram and estimating the query result size by executing a 2-D box query on the histogram.

To understand the approach, let us consider an unbounded range query Q given by $a <_\tau x_0$, where τ is the probability threshold for the $>$ predicate. This query returns all uncertain items a such that $Pr(a < x_0) > \tau$. In terms of the cumulative distribution function $F_a(x)$, we get the following condition:

$$Pr(a < x_0) > \tau \Leftrightarrow \int_{-\infty}^{x_0} f_a(x)dx > \tau \Leftrightarrow F_a(x_0) > \tau \quad (1)$$

This follows from the definition of pdf and cdf functions.

Let us consider a 2D graph where we plot the cdf function F of all uncertain items. Figure 1 shows an example of this graph. The cdfs for three data items a, b, and c are shown. The range query Q given by Equation 1 can be translated into a (unbounded) box query $x < x_0$ and $y > \tau$ over this 2D plot (the shaded region in Figure 1). Items a and b satisfy the query as they intersect the shaded region.

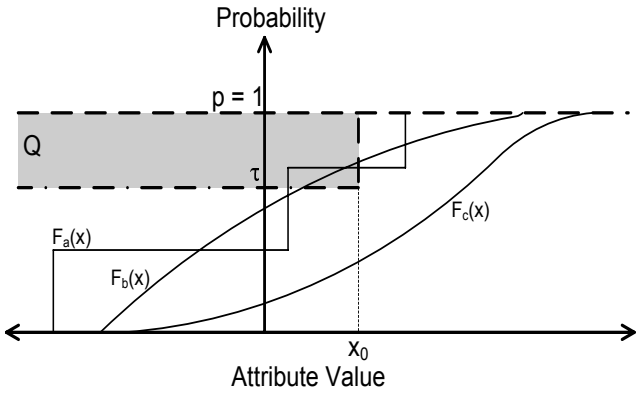

Fig. 1. Example plot for query $Q(x_0, \tau)$

Theorem 1. *All the items whose cdf function $F_a(x)$ lies in the box defined by query Q are part of the result of query Q. That is, $\forall a$, where the cdf function F_a lies in the box defined by query Q, we have $Pr(a < x_0) > \tau$.*

Proof. We observe that for any cdf F_a that lies in the box of query Q, we have $F_a(x) > \tau$ for some $x < x_0$. As F_a is a monotonically increasing function, we can deduce that $F_a(x_0) > F_a(x) > \tau$. Using 1, $P(a < x_0) > \tau$.

Now we state the following theorem without proof:

Theorem 2. *The total number of cdf lines that lie in the query box Q is equal to the number of lines crossing (intersecting) the vertical line-segment given by $\ell : x = x_0, \tau < y \leq 1$, which furthermore is equal to the number of lines crossing (intersecting) the horizontal ray $y = \tau, x < x_0$.*

The proof of this theorem follows from basic geometry and the monotonically increasing nature of cdf F.

Now finding all the items whose cdf function lies in the box defined by a query Q is equivalent to finding the total number of intersections of cdf lines with the vertical line-segment ℓ. To efficiently calculate this number we need to develop an approximation of the above technique. For this purpose, we define a *2-D grid* of histogram over the plot region. Given u_i, $0 \leq i < m$ as all the uncertain data items, we define

$$l = \min_i \left(l_{u_i}\right), r = \max_i \left(r_{u_i}\right)$$

where $[l_{u_i}, r_{u_i}]$ is the uncertainty interval of u_i. The plot region is bounded by 0 and 1 in the y (probability) direction and l, r in the x direction. The range R of the histogram is defined as $R = r - l$. The width of the histogram is given by the parameters δ_x and δ_p which represent the size of histogram along x and y (probability) axes respectively. A histogram bucket $H(x, y)$ covers the area given by the box $(x, y, x + \delta_x, y + \delta_p)$. The notations used are summarized in Table 1.

Table 1. Notations

Symbol	Meaning
f_a	Probability distribution function (pdf) of uncertain item a
F_a	Cumulative distribution function (cdf) of a
l_a, r_a	Left and right bounds of a's interval.
R_a	Range of a, $R_a = r_a - l_a$
u_i	All the uncertain data items ($0 \leq i \leq m$)
l, r	Leftmost and rightmost limits of all the uncertain intervals
R	Range of input data, $R = r - l$
δ_x, δ_p	Width of histogram bucket along x and y (probability) axis
H	Histogram structure for cost estimation

Definition 1. *The height of a histogram bucket $H(x, y)$ is the total number of cdf lines of uncertain items intersecting the box $(x, y, x + \delta_x, y + \delta_p)$.*

With this definition, we can now informally define the algorithm for calculating an approximation (upper-bound) of operator selectivity. Using Theorem 2 we see that the sum of individual histograms that cover the vertical line-segment ℓ gives a good approximation of the upper-bound of the result set size. The error in this approximation can be reduced by reducing the size of the histogram buckets. This extra accuracy comes at the cost of increased space overhead for storing the histogram structure.

If a cdf line has a large slope, it can contribute to more than one histogram in a given vertical window. This will result in over-estimation of the result size because the same cdf line will be counted multiple times. To prevent this, we propose a simple fix: If a cdf line intersects multiple (contiguous) histograms in a given vertical window, we only count its contribution in the *topmost* histogram. With this slight change, we will avoid counting the same line multiple times and obtain a tighter upper bound. Note that by adding the contribution of a given cdf line to the topmost histogram, we are guaranteed that there will be no false negatives. The algorithm for constructing this 2-D histogram is presented in Figure 2.

The algorithm presented in Figure 2 takes as input the uncertain data items from an attribute and the parameters δ_x and δ_p defining the width of each histogram inside the structure H. In addition to these values, it also takes the l and r values (defined earlier) which represent the spread of input data values. Depending on the attribute domain, these parameters can be provided by the user or the system can select them by random sampling. For a given uncertain item a, we start counting its contribution from its lower bound l_a and stop when we hit the upper-most bucket in the y-direction (Step 1(ii)). This small optimization saves a lot of computations as this step is repeated for all the input uncertain data items. Note that, for the correctness of our algorithm we *do* need to add the contributions to all the successive top buckets for item a. We take care of this *correction* in step 2 with just one pass over the entire histogram.

Given this histogram structure H, we can easily give an approximation for query result size. Figure 3 shows the algorithm for finding the selectivity estimate for query $Q(x, \tau) = a <_\tau x$.

Input

$u_i, 0 \leq i < m$: All the uncertain data items
δ_x, δ_p : Width of histogram along x and y axis
l, r : The left and right bounds for the histogram

Output

H: The histogram structure for the input data

0. Initialize H with all bucket heights $= 0$
1. **for** $a = u_0, u_1 \ldots, u_{m-1}$ **do**
   (i) **let** $x = \lfloor (l_a - l)/\delta_x \rfloor$; $p = 0$
   (ii) **while** $p < (1 - \delta_p)$
      (a) $p = F_a(l + (x + 1)\delta_x)$
      (b) $H\ (x, \lfloor p/\delta_p \rfloor)$++
      (c) x++
2. **for** $x = 0, 1, \ldots, \lfloor R/\delta_x \rfloor$
   (i) $H(x, \lfloor 1/\delta_p \rfloor)$ += $H(x - 1, \lfloor 1/\delta_p \rfloor)$
3. **return** H

Fig. 2. Algorithm for generating the histogram for unbounded range queries

Input

x_0, τ : Parameters of a query Q
H : Histogram structure
m : Total number of uncertain items
δ_x, δ_p : Width of histogram along x and y axis
l, r : The left and right bounds for the histogram

Output

An estimate (upper-bound) of query selectivity

1. **if** $x_0 < l$ **return 0**
2. **if** $x_0 > r$ **return 1**
3. $x = \lfloor (x_0 - l)/\delta_x \rfloor$
4. **let** $S = 0$
5. **for** $p = \lfloor \tau/\delta_p \rfloor, \ldots, \lfloor 1/\delta_p \rfloor$
   (i) $S = S + H(x, p)$
6. **return** (S/m)

Fig. 3. Algorithm for estimating query selectivity for unbounded range queries

Note that the above discussion applies to $a <_\tau x$ queries only. For unbounded range queries of the form $Q : a >_\tau x$, we have the following result:

$$a >_\tau x \Leftrightarrow Pr(a > x) > \tau \Leftrightarrow F_a(x) < 1 - \tau \tag{2}$$

Using Equation 2 we can see that if an uncertain item a *does not* satisfy the query $a <_{1-\tau} x$ (i.e. $F_a(x) \not> 1 - \tau$) then it will satisfy the query $a >_\tau x$. The algorithms presented in Figures 2 and 3 can therefore be used for $>_\tau$ queries with slight modifications. The selectivity of $>$ can be calculated by computing the selectivity of $<$ and using the fact that selectivity for $>_\tau$ is 1 - selectivity for $<_{1-\tau}$.

Theorem 3. *The time complexity of algorithm presented in Figure 2 is:*

$$\sum_{i=0}^{m-1} \left(\frac{R_{u_i}}{\delta_x} \right) + O\left(\frac{R}{\delta_x} \right)$$

Proof. The first terms comes from Step (1) in which we go through each item once for each uncertain item. Finally we add up all the contributions in the top histogram buckets in Step (2) which gives us the second term in the above expression.

4.2 General Range Queries

As discussed earlier, a general range query Q is expressed as $Pr(x_1 < a < x_2) > \tau$. This query returns all tuples such that:

$$\begin{aligned} Pr(x_1 < a < x_2) > \tau \Leftrightarrow \int_{x_1}^{x_2} f_a(x)dx > \tau \\ \Leftrightarrow F_a(x_2) - F_a(x_1) > \tau \end{aligned}$$

The previous section on unbounded range queries is a special case of the general range query where $x_1 = -\infty$ (or l) or $x_2 = \infty$ (or r).

We can extend the earlier solution to general range queries by adding another dimension to the histogram. In addition to the x-axis and y-axis representing x_2 (end-point of the range query) and the probability threshold τ respectively, we will now have a z-axis representing x_1 (or the beginning of range query).

The theoretical discussion of this selectivity estimation solution is similar to the unbounded case. In place of a 2-D curve, we will now have a 3-D curve for each uncertain item which is given by the function:

$$G_a(x_1, x_2) = \int_{x_1}^{x_2} f_a(x)dx = F_a(x_2) - F_a(x_1) \tag{3}$$

The range query Q will now translate to a box query given by $x < x_2$, $y > \tau$ and $z = x_1$. We can now state the following theorem for the 3-D curve:

Theorem 4. *Each item for which $G_a(x_1, x_2)$ intersects the box defined by query Q is part of the result of query Q. That is, $\forall a$, where the function G_a intersects the box defined by query Q, we have $Pr(x_1 < a < x_2) > \tau$.*

Proof. We observe that for any cdf F_a that lies in the box of query Q, we know that $G_a(x_1, x) > \tau$ for some $x < x_2$. This gives us that $G_a(x_1, x_2) > G_a(x_1, x) > \tau$. Using 3, we have $P(x_1 < a < x_2) > \tau$.

Similar to Theorem 2, we can prove that we can count the total number of items in the result set by counting the total number of intersections of function G_a with the line-segment $x = x_2$, $\tau < y \leq 1$ in the $z = x_1$ plane. The definition and construction of 3-D histogram is similar to the 2-D counterpart and is presented in Figure 4. The algorithm for estimating the answer size for a given query $Q(x_1, x_2, \tau)$ is presented in Figure 5.

Input

$u_i, 0 \leq i < m$: All the uncertain items
δ_x, δ_p : Width of histogram along x,z and y axis
l, r : The left and right bounds for the histogram

Output

H: The histogram structure for the input data

0. Initialize H, H_x, H_z, H_{xz} with all bucket heights $= 0$
1. **for** $a = u_0, u_1 \ldots, u_{m-1}$ **do**
   (i) **let** $x_{min} = \lfloor (l_a - l)/\delta_x \rfloor$, $x_{max} = \lfloor (r_a - l)/\delta_x \rfloor$
   (ii) **for** $z = x_{min}, \ldots, x_{max}$ **do**
   for $x = z, \ldots, x_{max}$ **do**
   (a) $p = G_a(l + z\delta_x, l + (x+1)\delta_x)$
   (b) **if** $(z = x_{min}) \wedge (x = x_{max})$
   $H_{xz}(x, \lfloor p/\delta_p \rfloor, z)$++
   (c) **else if** $(z = x_{min})$
   $H_z(x, \lfloor p/\delta_p \rfloor, z)$++
   (d) **else if** $(x = x_{max})$
   $H_x(x, \lfloor p/\delta_p \rfloor, z)$++
   (e) **else** $H(x, \lfloor p/\delta_p \rfloor, z)$++
2. **let** $x_{max} = \lfloor R/\delta_x \rfloor$
3. **for** $p = 0, \ldots, \lfloor 1/\delta_p \rfloor$
   (a) **for** $x = 0, \ldots, x_{max}$
   for $z = x_{max} - 1, x_{max} - 2, \ldots, 0$
   $H_z(x, p, z)$ += $H_z(x, p, z+1)$
   (b) **for** $z = 0, \ldots, x_{max}$
   for $x = 1, 2, \ldots, x_{max}$
   $H_x(x, p, z) += H_x(x-1, p, z)$
4. **for** $x = 0, \ldots, x_{max}$
   for $z = x_{max} - 1, x_{max} - 2, \ldots, 0$
   $H_{xz}(x, \lfloor 1/\delta_p \rfloor, z)$ += $H_{xz}(x, \lfloor 1/\delta_p \rfloor, z+1)$
5. **for** $z = 0, \ldots, x_{max}$
   for $x = 1, 2, \ldots, x_{max}$
   $H_{xz}(x, \lfloor 1/\delta_p \rfloor, z)$ += $H_{xz}(x-1, \lfloor 1/\delta_p \rfloor, z)$
6. **for** all x, z, p
   $H(x, z, p)$ += $H_z(x, p, z) + H_x(x, p, z) + H_{xz}(x, p, z)$
7. **return** H

Fig. 4. Algorithm for generating the histogram structure for general range queries

We can apply an optimization similar to the algorithm in Figure 2 by modifying only the local histogram area which is affected by an uncertain item and then propagating the effects globally by adding a post-processing step. This optimization helps in bringing down the running time of the algorithm significantly. To achieve this goal we keep three temporary histogram tables H_x, H_z and H_{xz} along with the main histogram structure H. For an uncertain item a, Step 1 adds the contribution of the item to the main histogram H, along with adding the contributions that are to be propagated globally to the temporary histograms. H_z and H_x store the contribution to the bins corresponding to $z = l_a$ and $x = r_a$ respectively, while H_{xz} stores the contribution to the bin

Input

x_1, x_2, τ : Parameters of a query Q
H : Histogram structure
m : Total number of uncertain items
δ_x, δ_p : Width of histogram bucket along x,z and y axis
l, r : The left and right bounds for the histogram

Output

An estimate (upper-bound) of query selectivity

1. **if** $x_1 < l$ $x_1 = l$
2. **if** $x_2 > r$ $x_2 = r$
3. **let** $x = \lfloor (x_2 - l)/\delta_x \rfloor$, $z = \lfloor (x_1 - l)/\delta_x \rfloor$
4. **let** $S = 0$
5. **for** $p = \lfloor \tau/\delta_p \rfloor, \ldots, \lfloor 1/\delta_p \rfloor$
   (i) $S = S + H(x, p, z)$
6. **return** (S/m)

Fig. 5. Algorithm for estimating query selectivity for general range queries

corresponding to $z = l_a$ and $x = r_a$. It is easy to see that the local contribution of the item a to H_z needs to be propagated to the plane given by $l_a \leq x < r_a$ and $z < l_a$ as for these values $Pr(z < a < x) = Pr(l_a < a < x)$ (Step 3a). Similarly, H_z needs to be propagated globally to the plane $l_a < z \leq r_a$ and $x > r_a$ as for this plane $Pr(z < a < x) = Pr(z < a < r_a)$ (Step 3b). In a similar fashion, H_{xz} is propagated to $z < l_a$ and $x > r_a$ (Step 4 and 5). Finally, we add all the temporary histograms to the main histogram to get the final histogram structure (Step 6).

Theorem 5. *The time complexity of algorithm presented in Figure 4 is:*

$$\sum_{i=0}^{m-1} \left(\frac{R_{u_i}^2}{2\delta_x^2} \right) + O\left(\frac{R^2}{\delta_x^2 \delta_p} \right)$$

Proof. By counting the number of loops. All the steps in Figure 4, except for Step 1, touch the cells only constant number of times. The number of loops in Step 1 gives the first summation.

4.3 General Range Queries Using Slabs

In Section 4.2 we discussed how the histogram construction technique can be extended to general range queries. While the accuracy of such an estimate is very good, the initial construction time and space trade-off is quadratic in terms of the range of the input data (R). In this section, we present another technique which has, in general, a lower accuracy than the previous technique but better space-time complexity.

In this algorithm, we partition the entire range of input data into slabs. Similar to histograms, the length of a slab is controlled by the input parameter δ_x. Each slab stores estimates of query selectivity for different values of p. A slab with end-points at $x = x_1, x_2$ stores the selectivity of a bounded range query $Q(x_1, x_2, \tau)$ for different

values of τ. Once again, the number of divisions (estimates) along the probability axis is controlled by δ_p. Note that, for a query that spans multiple slabs, we cannot just add the contributions of individual slabs. To solve this problem, we have a hierarchy of slabs. The size of slab at the bottom-most level of this hierarchy is exactly δ_x but as we go up the hierarchy the size increases exponentially until we reach the top-most slab, which encompasses the entire input region. At each level of the hierarchy there are two[1] sets of slabs, one starting at the midpoint of the other, so that we can get better estimates. We call these slabs A and B, respectively.

Formally, we have $\log(R/\delta_x)$ hierarchical levels, with each hierarchical level having two sets of slabs $A(i,j,p)$ and $B(i,j,p)$ where $j \leq \lceil \log_2(R/\delta_x) \rceil$.

Definition 2. *The slabs $A(i,j,p)$ and $B(i,j,p)$ cover the regions $\mathcal{R}_1 = [l + 2^j i\delta_x, l + 2^j(i+1)\delta_x]$ and $\mathcal{R}_2 = [l + 2^j(i+1/2)\delta_x, l + 2^j(i+3/2)\delta_x]$ respectively. The height of the slab $A(i,j,p)$ (or $B(i,j,p)$) is given by the number of uncertain items satisfying the bounded query $\mathcal{R}_1$ (or $\mathcal{R}_2$) with probability between $p\delta_p$ and $(p+1)\delta_p$.*

As mentioned earlier, each of these slabs stores the query answers for different values of query threshold τ. Thus, every $A(i,j)$ or $B(i,j)$ is an array of $\lfloor 1/\delta_p \rfloor$ values. The construction algorithm is presented in Figure 6. In Step 1, for each item, we find the slabs that are affected by the item and add the contribution of the item to the corresponding slabs.

Input

$u_i, 0 \leq i < m$: All the uncertain items
δ_x, δ_p : Parameters controlling width of divisions
l, r : The left and right bounds for the input region

Output

The slab structure for the input data

0. Initialize A and B with all buckets heights $= 0$
1. **for** $a = u_0, u_1, \ldots, u_{m-1}$ **do**
   (i) **for** $j = 0, 1 \ldots, \lceil \log_2(R/\delta_x) \rceil$ **do**
      (a) **let** $x_{min} = \lfloor (l_a - l)/(2^j\delta_x) \rfloor$,
         $x_{max} = \lfloor (r_a - l)/(2^j\delta_x) \rfloor$
      (b) **for** $x = x_{min} \ldots x_{max}$ **do**
         (A) **let** $p = G_a(l + x2^j\delta_x, l + (x+1)2^j\delta_x)$,
         (B) $A(x, j, \lfloor p/\delta_p \rfloor)$++
      (c) **let** $x_{min} = \lfloor (l_a - (l + 2^{j-1}\delta_x))/(2^j\delta_x) \rfloor$,
         $x_{max} = \lfloor (r_a - (l + 2^{j-1}\delta_x))/(2^j\delta_x) \rfloor$
      (d) **for** $x = x_{min} \ldots x_{max}$ **do**
         (A) $p = G_a(l + 2^j(x+1/2)\delta_x, l + 2^j(x+3/2)\delta_x)$
         (B) $B(x, j, \lfloor p/\delta_p \rfloor)$++
2. **return** A,B

Fig. 6. Algorithm for generating slabs

[1] In general, we can have more than two sets of slabs for each level of hierarchy which will further increase the accuracy of this technique.

Once we have this slab structure, we can get estimates by finding a pair of slabs that contains (over-estimate) and is contained (under-estimate) by the query region. With these estimates, we interpolate the estimates based on the the interval size to get the final estimate. The algorithm for finding the estimate is presented in Figure 7. In the algorithm, Step 1 picks j which corresponds to the slab size just smaller than the query. We have two additional functions *pickLB* and *pickUB*, which given the query limits and a level j, returns the slab that is contained inside and contains the query respectively. If these functions can not find any such slab at level j they return *null*. For $j < 0$, these functions simply return a slab with size 0 and all estimates are set to 0. In the case, these functions find more than one slab which satisfy the conditions of UB (LB) they return the one with minimum (maximum) estimate. This is done in order to get a tighter bound on the final estimate. The details of these functions are omitted due to space considerations. Steps 2 and 3 find the slabs and return them. Once we have a slab T_{LB} that bounds the answer from below and a slab T_{UB} that bounds the answer from above, we find the selectivity estimates of T_{LB} and T_{UB} in Step 6 and then finally in Step 7 we linearly interpolate the estimates based on the size of query and size of the two intervals returned. This gives us an estimate of the query result size.

Lemma 1. *For any query Q, the difference between the levels, from which T_{LB} and T_{UB} are picked up, is at most 2. Thus, the space covered by T_{UB} is at most 4 times that of T_{LB}.*

Proof. It follows from the cases of Figure 7. It remains to show that the *e*lse cases in Step 2(b) and Step 3(a),(b) are always successful in finding a slab. Note that the size of the slab at level j is less than the query interval. So a slab at level j could fit in the query. If this happens with the A slab being contained, then there is a slab at level $j+2$ that surely contains the query. This is because, an A slab at level $j+1$ contains at least one end-point of the query, and hence at level $j+2$, since an A slab and a B slab extend this A slab at level $j+1$ in different directions, at least one of the A slabs at level $j+2$ or B slabs at level $j+2$ will cover the entire interval. If at level j, the query covers a B slab, then it cuts two consecutive A slabs at level j and hence it is covered in either an A slab or a B slab at level $j+1$. If the query does not cover any slab at level j, then it again cuts two consecutive A slabs at level j. This means it is covered by a slab at level $j+1$. Also, it cuts at least one of these A slabs by more than half at the level j. Thus, there is an A slab at level $j-1$ which is contained in the query.

Theorem 6. *The time complexity of algorithm presented in Figure 6 is:*

$$O\left(\sum_{i=0}^{m-1}\left(\frac{R_{u_i}}{\delta_x}\right) + m\log\left(\frac{R}{\delta_x}\right)\right)$$

Proof. The above result directly follows from the following expression which is the total cost of Step 1.

$$\sum_{i=0}^{m-1}\sum_{j=0}^{\log(R/\delta_x)}\left\lceil\frac{R_{u_i}}{2^j\delta_x}\right\rceil$$

Input

x_1, x_2, τ : Parameters of a query Q
A, B : Slab structure
m : Total number of uncertain items
δ_x, δ_p : Parameters controlling width of divisions
l, r : The left and right bounds for the histogram

Output

An estimate of the query selectivity

1. **let** $j = \lfloor \log_2((x_2 - x_1)/\delta_x) \rfloor$
2. **if** (T = pickLB(x_1, x_2, j)) exists
   (a) $T_{LB} = T$
   (b) **if** (T = pickUB$(x_1, x_2, j+1)$) exists
   $T_{UB} = T$
   else T_{UB} = pickUB$(x_1, x_2, j+2)$
3. **else**
   (a) T_{LB} = pickLB$(x_1, x_2, j-1)$
   (b) T_{UB} = pickUB$(x_1, x_2, j+1)$
4. **let** $S_{min} = S_{max} = 0$, t_1 = length of T_{LB},
   t_2 = length of T_{UB}
5. **for** $p = \lfloor \tau/\delta_p \rfloor, \ldots, \lfloor 1/\delta_p \rfloor$
   (a) S_{min} += $T_{LB}(p)$, S_{max} += $T_{UB}(p)$
6. $S = S_{min} + (S_{max} - S_{min}) \times (x_2 - x_1 - t_1)/(t_2 - t_1)$
7. **return** (S/m)

Fig. 7. Algorithm for estimating query selectivity using slabs

Similarly, we can also show that the total space overhead is $O\left(R/\delta_x\right)$. Both these results are intuitive if we observe that the total cost/space is asymptotically bounded by number of slabs at the bottom-most level as the number of slabs at higher levels decrease exponentially.

5 Experimental Evaluation

We have implemented our statistics collection and selectivity estimation algorithms in *Orion*, a publicly available extension to PostgreSQL that provides native support for uncertain data [3]. To efficiently evaluate the queries discussed in this paper, Orion uses an indexing scheme known as *probabilistic threshold index* (PTI) introduced in [18]. This system not only allows us to validate the accuracy of our methods in a realistic runtime environment, it also gives additional insight into the overall effect our techniques have on query optimization in an industrial-strength DBMS.

5.1 Implementation

PostgreSQL measures the cost of query plans in disk page fetches (for simplicity, all CPU efforts are converted into disk I/Os). The optimizer generally estimates the cost of query plans by calculating the overall selectivity and multiplying it against the

estimated cardinality. In the common case of multiple predicates, individual selectivies are multiplied together, except for range queries where the dependence between the lower and upper bounds is simple to evaluate.

Virtually every numeric data type in PostgreSQL shares the same source code for cost estimation. Using this code base, we have built our implementation of the algorithms in Figures 2, 4, 3, and 5. Using the elegant framework PostgreSQL provides for new data management techniques, our implementation extends the functionality of Orion's UNCERTAIN data type by registering the optional callbacks for collecting statistics and estimating selectivity.

5.2 Methodology

To ensure correctness, we ran each experiment on a variety of queries and datasets, and then averaged the results. After populating the database with each test dataset, we first used `VACUUM ANALYZE` to generate the statistics in advance. The following experiments were conducted on a 1.6 GHz Pentium CPU with 512 MB RAM, running Linux 2.6.17, PostgreSQL 8.1.5, and Orion 0.1. Note that most of the resulting plots show the *relative error* of the selectivity estimates, i.e. the goal is to be as close to 0% as possible.

Synthetic Datasets. Each dataset consists of random "sensor readings," using a schema `Readings (rid, value)`. Without loss of generality, the uncertain values (i.e. reported from the sensors) are floating point numbers ranging from 0 to 1000, and the pdf for each uncertain value is a uniform distribution. The interval sizes are distributed normally, with midpoints distributed uniformly. We refer to our three main datasets as `Data-5`, `Data-50`, and `Data-100`; the numbers correspond to the average width of the uncertain value intervals.

Table 2 summarizes the control variables for the subsequent experiments. In particular, we show that our algorithms perform well without regard to dataset cardinality, and are reasonably robust to query selectivity and probabilistic threshold. In addition, we demonstrate the effect of increased precision as a trade-off between construction time and space versus the resulting accuracy of the selectivity estimates.

Example Query Plan. To illustrate the impact that correct estimates have on query optimization, we present the following example output from PostgreSQL. When no selectivity estimation function is available for a given predicate, PostgreSQL simply returns the default value of 1/3 for estimating unbounded range queries, and 0.005 for general range queries. In practice this estimate favors the use of unclustered indexes, such as PTI [18], to improve I/O performance:

Table 2. Summary of control variables

Variable	Default Value
Cardinality	250,000
Selectivity	2.5 %
Threshold	50 %
Precision	70 bins

```
SELECT * FROM Readings WHERE value < 750;
----------------------------------------
Bitmap Heap Scan on Readings
 (cost=742.33..4075.67 rows=66667 width=36)
 (actual=20379.348..20824.652 rows=153037)
 Recheck Cond: (value < 750::real)
->  Bitmap Index Scan on pti_value
 (cost=0.00..742.33 rows=66667 width=0)
 (actual=20378.677..20378.677 rows=153K)
 Index Cond: (value < 750::real)
```

With accurate estimates, the optimizer makes the correct decision, namely not to use the available PTI index:

```
(same query as before, but using our algorithms)
------------------------------------------------
Seq Scan on Readings
  (cost=0.00..5000.00 rows=164333 width=35)
  (actual=83.841..15545.401 rows=153037)
  Filter: (value < 750::real)
```

As shown in this example, accurate selectivity estimation saves the system thousands of disk fetches (i.e. 15545 total cost instead of 20825). In general, incorrect estimates may result in much higher losses of efficiency.

5.3 Results

We now evaluate the accuracy and performance of our cost estimation techniques for unbounded range queries using the 2D histogram structure introduced in Section 4.1 (see Figure 3), and general range queries using the 3D histogram discussed in Section 4.2 (see Figure 5).

Accuracy at Varying Selectivities: The first experiment verifies the accuracy of our algorithms, regardless of query selectivity. Figures 8 and 9 summarize the results using all three synthetic datasets. For clarity, we have only plotted one of them. The x-axis shows the selectivity of the query which was varied from high (1%) to low (100%). The y-axis shows the accuracy of the estimation as a percentage relative to the size of the exact result. Our algorithm significantly outperforms the baseline PostgreSQL estimate. As expected, high selectivity has a slight effect on the accuracy of our methods.

Accuracy at Varying Cardinalities: The next experiment studies the overall scalability of our algorithms, namely the impact of the size of the relation on the accuracy of the estimations. Figures 10 and 11 show the results for three representative queries. The x-axis shows the size of the table in number of tuples which was varied from 50,000 to 800,000. The results show that our approach is unaffected by the size of the dataset. This is in sharp contrast to the baseline PostgreSQL estimator (not shown) which is much more sensitive to the dataset size, particularly for smaller datasets.

Accuracy at Varying Thresholds: Figures 12 and 13 show the impact of query threshold on the accuracy of the estimates. The x-axis shows the threshold probability and the

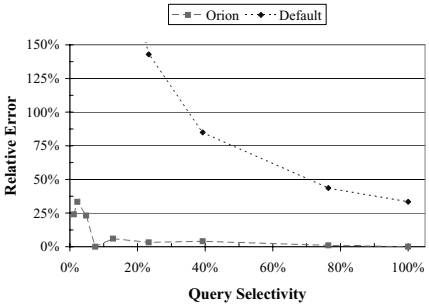

Fig. 8. Selectivities (2D)

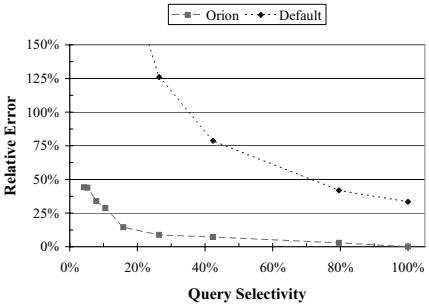

Fig. 9. Selectivities (3D)

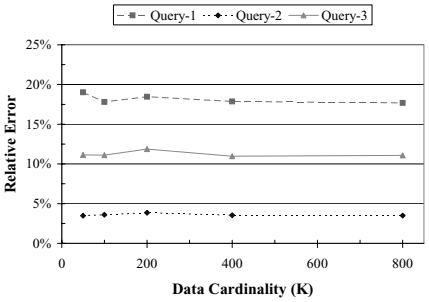

Fig. 10. Cardinalities (2D)

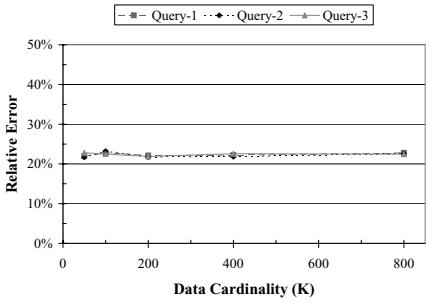

Fig. 11. Cardinalities (3D)

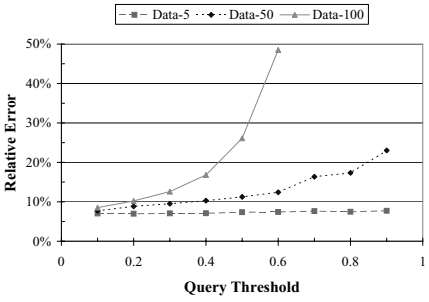

Fig. 12. Thresholds (2D)

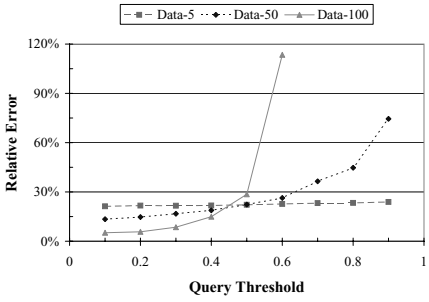

Fig. 13. Thresholds (3D)

y-axis shows the relative accuracy with respect to the correct answer size. Once again, we observe that our algorithm is much more robust than the baseline PostgreSQL estimator (not shown) that simply returns a constant selectivity. Our implementation shows slightly better accuracy for smaller thresholds, in part because larger thresholds result in additional tuples becoming part of the query answer, leading to overestimates. We can see that for highly selective queries, our algorithm is significantly better that the baseline and thus it is more likely to lead the optimizer into choosing a much more efficient plan.

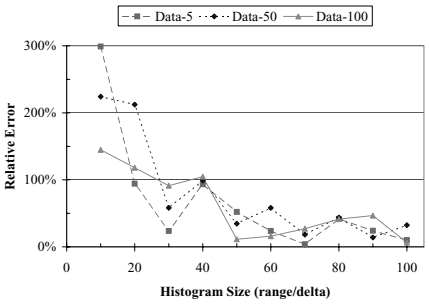

Fig. 14. Precision (2D)

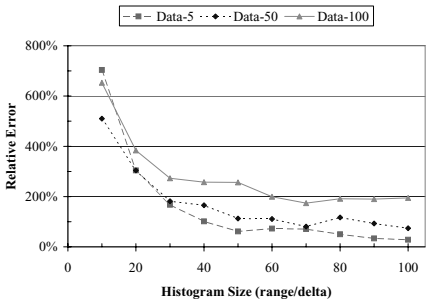

Fig. 15. Precision (3D)

Accuracy at Varying Precisions: Next we show the relationship between the size of the histograms and the resulting accuracy. Figures 14 and 15 summarize the results for each dataset. The x-axis shows the number of histogram buckets in each dimension, which was varied from 10 to 100. Clearly, both algorithms perform better with a more detailed histogram. Our algorithm outperforms the baseline for smaller histograms. As expected, we see that after a certain amount (i.e. 70, for these datasets and queries), larger histograms do not provide significant increase in accuracy.

Runtime Performance Overhead: The final set of experiments study the runtime performance of constructing the statistics and estimating the selectivity of a query. We have omitted figures for these findings because of limited space. As expected, the estimation times are constant and almost negligible (on the order of 15 ms). The histogram construction times scale linearly with respect to data cardinality, and grow a little more than linear as the requested number of buckets increases. For the bulk of our experiments, histogram construction only amounted to several hundred milliseconds.

6 Conclusions and Future Work

In this paper, we developed algorithms for computing selectivity estimates of probabilistic queries over uncertain data. The estimation techniques can be applied both to tuple uncertainty and attribute uncertainty models. These techniques were implemented in PostgreSQL and found to provide accurate estimates for uncertain data. The algorithms presented can be further improved by combining them with standard cost estimation techniques such as equi-depth binning and sampling. We showed both theoretically and empirically that our histogram construction algorithms are fast. The experiments show that they give very accurate estimation especially for less selective queries. For more selective queries, the accuracy is not quite as good, but is still much better than the baseline estimator.

Acknowledgements. This work was supported by NSF grants IIS 0242421, IIS 0534702, IIS 0415097, CCF-0621457 and AFOSR award FA9550–06–1–0099.

References

1. Benjelloun, O., Sarma, A., Halevy, A., Widom, J.: ULDBs: Databases with uncertainty and lineage. In: Proceedings of International Conference on Very Large Databases (2006)
2. Dalvi, N., Suciu, D.: Efficient query evaluation on probabilistic databases. In: Proceedings of International Conference on Very Large Databases (2004)
3. Orion (2008), http://orion.cs.purdue.edu/
4. Sen, P., Deshpande, A.: Representing and querying correlated tuples in probabilistic databases. In: Proceedings of IEEE International Conference on Data Engineering (2007)
5. Singh, S., Mayfield, C., Prabhakar, S., Shah, R., Hambrusch, S.: Indexing uncertain categorical data. In: Proceedings of IEEE International Conference on Data Engineering (2007)
6. Tao, Y., Cheng, R., Xiao, X., Ngai, W., Kao, B., Prabhakar, S.: Indexing multi-dimensional uncertain data with arbitrary probability density functions. In: Proceedings of the 31st Very Large Data Bases conference (2005)
7. Cheng, R., Kalashnikov, D.V., Prabhakar, S.: Evaluating probabilistic queries over imprecise data. In: Proceedings of ACM Special Interest Group on Management of Data (2003)
8. Cheng, R., Singh, S., Prabhakar, S., Shah, R., Vitter, J., Xia, Y.: Efficient join processing over uncertain data. In: Proceedings of International Conference on Information and Knowledge Management (2006)
9. Deshpande, A., Guestrin, C., Madden, S., Hellerstein, J., Hong, W.: Model-driven data acquisition in sensor networks. In: Proceedings of International Conference on Very Large Databases (2004)
10. Poosala, V., Ioannidis, Y., Haas, P., Shekita, E.: Improved histograms for selectivity estimation of range predicates. In: Proceedings of ACM Special Interest Group on Management of Data (1996)
11. Pfoser, D., Jensen, C.: Capturing the uncertainty of moving-objects representations. In: Proceedings of International Conference on Scientific and Statistical Database Management (1999)
12. Antova, L., Koch, C., Olteanu, D.: 10^10^6 worlds and beyond: Efficient representation and processing of incomplete information. In: Proceedings of 23rd International Conference on Data Engineering (2007)
13. Boulos, J., Dalvi, N., Mandhani, B., Mathur, S., Re, C., Suciu, D.: Mystiq: A system for finding more answers by using probabilities. In: Proceedings of ACM Special Interest Group on Management of Data (2005)
14. Widom, J.: Trio: A system for integrated management of data, accuracy, and lineage. In: Proceedings of the Second Biennial Conference on Innovative Data Systems Research (2005)
15. Nierman, A., Jagadish, H.V.: ProTDB: Probabilistic Data in XML. In: Proceedings of International Conference on Very Large Databases (2002)
16. Hung, E., Getoor, L., Subrahmanian, V.S.: PXML: A probabilistic semistructured data model and algebra. In: Proceedings of IEEE International Conference on Data Engineering (2003)
17. Ljosa, V., Singh, A.: APLA: Indexing arbitrary probability distributions. In: Proceedings of IEEE International Conference on Data Engineering (2007)
18. Cheng, R., Xia, Y., Prabhakar, S., Shah, R., Vitter, J.: Efficient indexing methods for probabilistic threshold queries over uncertain data. In: Proceedings of International Conference on Very Large Databases (2004)
19. Lakshmanan, L., Leone, N., Ross, R., Subrahmanina, V.: Probview: A flexible probabilistic database system. ACM Transactions on Database Systems 22(3), 419–469 (1997)