Submission no: 396

Efficient Join Processing over Uncertain Data

Reynold Cheng
Department of Computing
The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong
Email: csckcheng@comp.polyu.edu.hk

Yuni Xia   Sunil Prabhakar   Rahul Shah   Jeffrey Vitter
Department of Computer Science
Purdue University
West Lafayette, IN 47907-1398, USA
Email: {xia,sunil,rahul,jsv}@cs.purdue.edu

Abstract

In an uncertain database, each data item is modeled as a range associated with a probability density function. Previous works for this kind of data have focussed on simple queries such as range and nearest-neighbor queries. Queries that join multiple relations have not been addressed in earlier work despite the significance of joins in databases. In this paper, we address probabilistic join over uncertain data, essentially a query that augments the results with probability guarantees to indicate the likelihood of each join tuple being part of the result. We extend the notion of join operators, such as equality and inequality, for uncertain data.

We also study the performance of probabilistic join. We observe that a user may only need to know whether the probability of the results exceeds a given threshold, instead of the precise probability value. By incorporating this constraint, it is possible to achieve much better performance. In particular, we develop three sets of optimization techniques, namely item-level, page-level and index-level pruning, for different join operators. These techniques facilitate pruning with little space and time overhead, and are easily adapted to most join algorithms. We verify the performance of these techniques experimentally.

1 Introduction

There is ongoing research interest in systems that acquire information from the external world. Sensornets, for example, allow physical entities such as temperature, pressure and voltage to be collected through large numbers of inexpensive sensors [6]. Locationing devices like cell phones and GPS-equipped devices also allow phone users' and vehicle's locations to be obtained easily. The massive amounts of information collected about the physical world enable the development of novel applications that base their decisions on these physical data.

An important operation for these applications is join processing between relations where join predicates are evaluated on these measured attributes. An all-pairs join query is a well-known example of such a join: Given a relation with location information for moving objects, return all pairs of points that are within some distance, ϵ of each other.

Joins are also important for sensor data. Consider an application that stores the temperature and pressure recorded by various sensors. A join query can be used to determine pairs of sensors for which one sensor has a higher temperature while the other has a higher pressure (this may be anomalous behavior as pressure is expected to rise with temperature). As another example, consider a query that determines pressure and temperature values for points in a region. Suppose two sets of sensors (e.g., sensor dusts) were dispersed by aircraft into the region, where one set provides location and temperature, and the other provides location and pressure. An equality join over location attribute values produces temperature and pressure information for every location. As another example, an equality (self-) join query can be issued to evaluate the pairs of sensors that report the same value (temperature, pressure, etc.) from a set of sensors. The results of these join queries may be useful for further scientific studies, physical data correlation [6], weather forecasting, etc.

Unfortunately, joining "natural data" from the sensing instruments is not straightforward, due to the *uncertainty* inherent with the data obtained in the external dynamic environment. In particular, The entities like temperature and location values are continuously evolving. Since the information during the inter-arrival time of data samples is not provided to the system, there is uncertainty between the database value and the actual value. This problem can be aggravated by network issues, where data packets can be delayed or even lost, especially in a wireless network [1, 6]. As a result, the database values may have a large discrepancy compared with the actual values.

A direct consequence of data uncertainty is that join queries may produce incorrect results. To illustrate, consider a scientific application where an equality join query is issued over two sets of temperature values (obtained from two sensor networks in separate geographical regions) to discover the pairs of sensors that give the same temperature value. Figure 1(a) shows two tables, A and B, storing two attributes $(ID, Temp)$, which represents the temperature values $Temp$ recorded by sensors with names given by *ID*. Suppose we would like to perform an equality join over the temperature attributes to find out which pairs of entities in A and B match. The result is shown by the line joining the two entities. This result is incorrect if we consider the true values of the sensors given by Figure 1(b): since the actual value for A_1 is different from that of B_1, A_1 should not be paired with B_1. Instead, A_1 matches B_2, where both temperature values equal to 11^oF. Thus there is a *false positive* in the true result – (A_1, B_1) is wrongly returned to the user. Figure 1(b) also shows that A_2 should be matched with B_3. Consequently, (A_1, B_2) and (A_2, B_3) are not returned to the user, resulting in two *false negatives*. As we can see, the join result returned by the database is significantly different from the actual result. If this result is further processed by the application, the error may propagate in the analysis and wrong conclusions/decisions may be made.

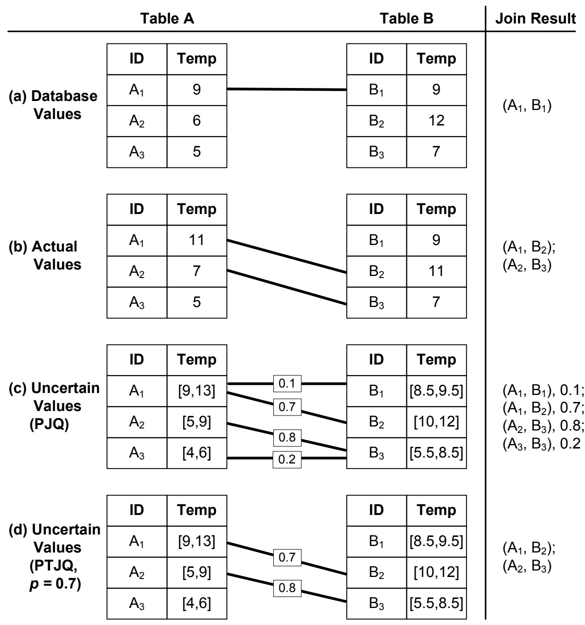

Figure 1. Illustrating join over uncertain data.

To avoid incorrectness in query answers, the idea of using an *uncertainty model* rather than a single numerical value to describe an item is proposed in [1]: each item is associated with a range of possible values and a probability density function (pdf) that describes the probability distribution of the value within the range. To address the above uncertainty problem, an uncertainty interval can be a fixed bound d, which is a result of negotiation between the database system and the sensor; if the system does not receive any update from the sensor, it can assume that the sensor's current value must be between $[v-d, v+d]$, where v is the value of the sensor last reported to the server [17]. The pdf of the sensor value within the range may be obtained through machine learning techniques [6]. By incorporating the notion of uncertainty into data values, imprecise, rather than exact, answers are generated. Each join-pair is associated with a probability to indicate the likelihood that the two tuples are matched. We use the term *Probabilistic Join Queries* (PJQ) to describe these types of joins over uncertain data.

Figure 1(c) illustrates the idea of PJQ. Each temperature attribute stores a range that encloses the data value, together with a pdf that describes the distribution (not shown here). Each tuple-pair is associated with a probability value that indicates the likelihood of the join. Notice that both (A_1, B_2) and (A_2, B_3) are now included in the result. In this example, therefore, the false negative problem vanishes. Also, we have a 0.7 and 0.8 confidence for these pairs. On the other hand, the false positive, (A_1, B_1), remains in the result, and a new false positive, (A_3, B_3), is introduced. However, both false positives are augmented with a relatively low probability (0.1 and 0.2 respectively), suggesting to the user that these two matches are less likely to occur.

How are these probability values computed? To answer this, we must understand the semantics of join operators for uncertainty. The notions of equality and inequality have to be extended to support uncertain data. We will address the new definitions of comparison operators for the uncertain data model. Furthermore, we demonstrate how it is possible to relax the requirements for comparison operators, in order to allow more flexibility in specifying accuracy requirements of joins over uncertainty.

Another dimension of our study deals with the performance issues of joins over uncertainty. We observe that although the answers probabilities are useful, it is not always necessary to know their exact values. Often the user is only concerned about whether the probability value exceeds a given threshold. We term the variant of PJQ which only returns tuple pairs when their probabilities exceed a certain threshold as *Probabilistic Threshold Join Queries* (PTJQ). An example of PTJQ is shown in Figure 1(d), where we assume the user is only interested in tuple pairs whose probabilities exceed threshold $p = 0.7$. As a result, the two pairs with low probability values (0.1 and 0.2) are not included in the answer. Compared with Figure 1(c), PTJQ returns fewer false negatives.

More importantly, there are various techniques to enhance the I/O and CPU performance of PTJQ. In particular, we develop three pruning techniques: (1) *item-level pruning*, where two uncertain values are pruned without evaluating the probability; (2) *page-level pruning*, where two pages are pruned without probing into the uncertain data stored in each page; and (3) *index-level pruning*, where all the data stored under a subtree are pruned. These techniques introduce little space and time overhead, and can be augmented to existing join algorithms easily.

As a summary of our contributions, we extend the semantics of join operators over exact, single-valued data to uncertain data. We present the concept of probabilistic join queries (PJQ) and illustrate how they can be evaluated. We illustrate how probabilistic threshold join queries (PTJQ), a variant of PJQ that constrains on the answers based on their probability values, can improve the join performance significantly based on various pruning techniques. We also perform evaluations to test our methods.

In Section 2, we define the uncertainty model of data assumed in this paper, and various notions of join operators over uncertainty. Section 3 presents item-level pruning techniques for each join operator. In Section 4, we study how the performance of join can be further improved through page-level and index-level pruning techniques. We present our experimental results in Section 5. Related work is discussed in Section 6, and Section 7 concludes the paper.

2 Comparing Uncertain Values

In this section we describe the uncertainty model, and definitions of comparison operators for uncertainty.

2.1 Probabilistic Uncertainty Model

To capture the uncertainty of dynamic entities such as temperature, pressure and location values, a data scheme known as *probabilistic uncertainty model* was proposed in [1]. This model assumes that each data item can be represented by a range of possible values and their distributions. Formally, assume each tuple of interest consists of a real-valued attribute a where join operations will be performed. Note that a is treated as a continuous random variable, and it is assumed that each uncertain attribute value is mutually independent. The *probabilistic uncertainty* of a consists of three components [1]:

Definition 1 *An* **uncertainty interval** *of a, denoted by $a.U$, is an interval $[a.l, a.r]$ where $a.l, a.r \in \Re$, $a.r \geq a.l$ and $a \in a.U$.*

Definition 2 *An* **uncertainty pdf** *of a, denoted by $a.f(x)$, is a probability distribution function of a, such that $\int_{a.l}^{a.r} a.f(x)dx = 1$ and $a.f(x) = 0$ if $x \notin a.U$.*

Definition 3 *An* **uncertainty cdf** *of a, denoted by $a.F(x)$, is a cumulative distribution function (cdf) of a, where $a.F(x) = \int_{a.l}^{x} a.f(y)dy$.*

Notice that $a.F(x) = 0$ if $x < a.l$ and $a.F(x) = 1$ if $x > a.r$.

The exact realization of this model is application-dependent. For example, in modeling sensor measurement uncertainty, $a.U$ is an error bound and $f(x)$ is a Gaussian distribution. In modeling moving objects, Wolfson et al. [17] suggested a bounded uncertainty model where each moving object only reports its location if its current location deviates from its reported location by more than d, so that at any point of time the uncertainty of the location value stored in the system has uncertainty of not more than d.

The specification of uncertain pdf is also application-specific. For convenience, one may assume that the uncertainty pdf $f(x)$ is a uniform distribution i.e., $f(x) = \frac{1}{a.r - a.l}$ for $a \in [a.l, a.r]$; essentially, this implies a "worst-case" scenario where we have no knowledge of which point in the uncertainty interval possesses a higher probability. In sensor networks, Deshpande et al. [6] assumed the reading of each sensor node is a Gaussian distribution parameterized with a mean and variance value. They also suggested that these Gaussian distributions can be constructed through machine learning algorithms, such as [11]. Another example pdf is a triangular distribution. Note that although the uncertainty model described here is presented for one-dimensional data, its concept can be extended to multiple dimensions.

2.2 Uncertainty Comparison Operators

Consider the equality of two uncertain-values a and b. Since a and b are not single values, traditional notions of comparison operators (such as equality and inequality) cannot be used. Due to the range of possible values for each data item it is not immediately obvious whether the two are equal in value or not. If there is no overlap in their range, clearly they cannot be equal. However, if there is an overlap, there is the possibility that the two could be equal. We are interested in finding the likelihood of this event. In this section, we extend the definitions of common comparison operators to support uncertain values. In particular, we express "imprecision" in these operators in terms of probability values.

To understand "equality" for uncertain data, consider Figure 2 where the overlap between $a.U$ and $b.U$ is $[a.l, b.r]$. Apparently, the probability a equals to b is just $\int_{a.l}^{b.r} a.f(x)b.f(x)dx$ (that is, a is considered to be equal to b if they are within infinitesimal distance from one another). By this definition, the probability of equality between two continuous random variables is always infinitesimally small. Also, given that the exact values for these data

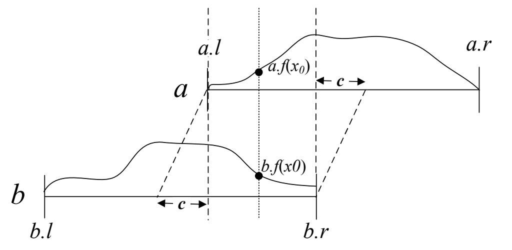

Figure 2. Comparing uncertain values.

items are not known, the user is more likely to be interested in them being very close in value rather than exactly equal. Naturally, how close they are should be determined by the user.

Based upon this observation, we define equality using a a user prescribed parameter, called *resolution* (c), as: a is equal to b if they are within c of each other i.e., $b-c \leq a \leq b+c$ or $a-c \leq b \leq a+c$:

Definition 4 Equality ($=_c$)*: Given a resolution c, a is equal to b with probability*

$$P(a =_c b) = \int_{-\infty}^{\infty} a.f(x) \cdot (b.F(x+c) - b.F(x-c))dx$$

Essentially, a is equal to b when $a = x_0$ if b is in the range $[x_0 - c, x_0 + c]$, with a probability of $b.F(x_0+c) - b.F(x_0-c)$, or $\int_{x_0-c}^{x_0+c} b.f(x)dx$. It can be seen that if c is zero, $P(a =_c b)$ becomes zero, which is consistent with our previous discussions that a and b have a zero probability of being exactly equal. This definition is thus more reasonable than the definition $\int_{a.l}^{b.r} a.f(x)b.f(x)dx$. Figure 2 illustrates this definition of equality, where we can see a and b only join in $[a.l-c, b.r+c]$.

Let $l_{a,b,c}$ be $\max(a.l-c, b.l-c)$ and $u_{a,b,c}$ be $\min(a.u+c, b.u+c)$. For the case that the two intervals are within distance c of each other, Definition 4 can be rewritten as:

$$P(a =_c b) = \int_{l_{a,b,c}}^{u_{a,b,c}} a.f(x)(b.F(x+c) - b.F(x-c))dx \quad (1)$$

where the overlap of $a.U$ and $b.U$ is given by $[l_{a,b,c}, u_{a,b,c}]$. We assert without proof that our definition of equality is symmetric i.e., $P(a =_c b)$ yields the same value as $P(b =_c a)$.

Notice that $P(a =_c b)$ is zero when $b.r + c < a.l$ or $a.r + c < b.l$. This indicates that a and b have no chance of being equal. Based upon the definition of equality, we can define **Inequality** as follows:

Definition 5 Inequality ($\neq_c$)*: Given a resolution c, a is not equal to b with probability $P(a \neq_c b) = 1 - \int_{-\infty}^{\infty} a.f(x) \cdot (b.F(x+c) - b.F(x-c))dx$.*

To address the question "Is a greater than' b?", let us look at Figure 2. In $[b.r, a.r]$, b cannot be larger than a, since $b.f(x)$ is 0 when $b > b.r$. Thus if a is within $[b.r, a.r]$, it is larger than b with probability $\int_{b.r}^{a.r} a.f(x)dx$, or $1 - a.F(b.r)$. At any point x_0 inside $[a.l, b.r]$, a is larger than b with a probability $a.f(x_0)b.F(x_0)$, where $b.F(x_0)$ is the probability that b is less than x_0. Therefore, in $[a.l, b.r]$, the probability that a is larger than b is given by $\int_{a.l}^{b.r} a.f(x)b.F(x)dx$. There is no need to consider $[b.l, a.l]$, because b is always less than a when b is in this region. To sum up, the probability that a is larger than b in Figure 2 is:

$$\int_{a.l}^{b.r} a.f(x)b.F(x)dx + 1 - a.F(b.r)$$

Upon considering all possible scenarios of overlap between $a.U$ and $b.U$, we obtain the definition of ">":

Definition 6 Greater than ($>$)*: $a > b$ with probability $P(a > b)$*

$$= \begin{cases} \int_{\max(a.l,b.l)}^{b.r} a.f(x)b.F(x)dx + 1 - a.F(b.r) & a.l \leq b.r < a.r \\ \int_{\max(a.l,b.l)}^{a.r} a.f(x)b.F(x)dx & b.l \leq a.r \leq b.r \end{cases}$$

For the case that a lies entirely to the left of b, i.e. $a.r < b.l$, $P(a > b) = 0$. Also, for the case that a lies entirely to the right of b, i.e. $a.l \geq b.r$, $P(a > b) = 1$.

Note that in a continuous-valued domain, $P(a > b)$ is the same as $P(a \geq b)$ because a can never be exactly equal to b. In the sequel we will not discuss $a \geq b$. In a similar manner as $<$, we can also define **Less than ($<$)**.

We can see from that comparison over uncertainty is imprecise. The degree of imprecision, represented by probability values, indicates the confidence of the comparison result. For example, if $P(a > b) = 0.01$, then a is unlikely to be greater than b. It is worth mentioning that the definitions of comparisons for uncertainty with continuous uncertainty pdfs can be extended to support discrete pdfs, as discussed in our technical report [4].

2.3 Comparing Uncertainty with Certainty

Some situations may require the join of uncertain values with "certain" values. For example, a user can join the current locations of people with locations of buildings (where the locations are fixed), in order to find out which persons are in which buildings. In general, operators between an uncertain value a and a certain value $v \in \Re$ can be defined as:

$$\begin{aligned} P(a =_c v) &= \int_{v-c}^{v+c} a.f(x)dx = a.F(v+c) - a.F(v-c) \\ P(a \neq_c v) &= 1 - P(a =_c v) = 1 - a.F(v+c) + a.F(v-c) \\ P(a > v) &= 1 - a.F(v) \\ P(a < v) &= a.F(v) \end{aligned}$$

which can be treated as special cases for the definitions of uncertainty operators.

2.4 Probabilistic Join Queries

We can now formulate the join problem. Suppose we have two tables R and S containing m and n tuples respectively. Both tables contain an uncertain attribute upon which the join will be performed. We name the uncertain attribute of the ith row as R_i for table R, and as S_i for table S. Then the Probabilistic Join Query (PJQ) is defined as follows.

Definition 7 *Given an uncertainty comparator θ_u (where θ_u is any one of $=_c, \neq_c, >, <$), a* **Probabilistic Join Query (PJQ)** *returns all tuples $(R_i, S_j, P(R_i \theta_u S_j))$ where $i = 1, \ldots, m$, $j = 1, \ldots, n$ and $P(R_i \theta_u S_j) > 0$.*

Essentially, a PJQ returns join pairs with a non-zero probability of meeting the join condition. In this paper, we study a variant of PJQ called *Probabilistic Threshold Join Query* (PTJQ). It has an additional constraint that only join pairs whose probabilities exceed a user-defined threshold is returned – which makes sense when a user is only interested in results that meets his confidence requirement.

Definition 8 *Given an uncertainty comparator θ_u (where θ_u is any one of $=_c, \neq_c, >, <$), a* **Probabilistic Threshold Join Query (PTJQ)** *returns all tuples (R_i, S_j) such that $i = 1, \ldots, m$, $j = 1, \ldots, n$, and $P(R_i \theta_u S_j) > p$, where $p \in [0, 1]$ is called the probability threshold.*

A PTJQ only returns join pairs that have probabilities higher than p. Another difference from PJQ is that PTJQ only returns the pairs, (R_i, S_j), but not the actual probability values. In the sequel, we will explain how these two differences are exploited for performance improvement.

2.5 Processing Joined Tuples

So far we have defined the semantics of join over uncertain attributes for two tables. In the rest of the paper, we will investigate efficient evaluation techniques for PTJQs. Now, what if the joined result is further joined with another table? Here we discuss how our model can be extended to address this problem. In particular, we explain how to use PJQ answers, as well as the evaluation and performance issues of PTJQ result processing.

Suppose table R has an uncertain attribute a, which is to be joined with attribute b of S. Suppose the attribute $S.b$ of the joined tuple (i.e.,$R_i \theta_u S_j$) is further joined with attribute c of another table T. Recall from the definition of PJQ that the probability R joins S is also augmented to the result. We must take this probability into account in order to further process this joined relation.

Now consider a relation X created by joining R and S. Every tuple in X has an extra attribute called *exist-prob* which represents the probability that such a tuple exists in X.

For any tuple X_k (i.e., kth tuple of X) created by $R_i \theta_u S_j$, X_k.*exist-prob* $= P(R_i \theta_u S_j)$. The probability that X_k joins tuple T_m (i.e., the mth tuple of T) is then

$$X_k.\textit{exist-prob} \times T_m.\textit{exist-prob} \times P(X_k.b \theta_u T_m.c) \quad (2)$$

Thus, by augmenting the column *exist-prob* to every table created, the join result can be further passed to another join operator in a query plan. By default, in "certain" relations, *exist-prob* is equal to 1 for every row. Note that this definition of tuple uncertainty is widely used in uncertain databases [5, 12].

For PTJQ, we may not get the exact probability values. All we know about the returned tuple X_k is that $p \leq P(R_i \theta_u S_j) \leq 1$. If some tuple was eliminated by PTJQ, then we are sure that the probability that this tuple will produce further result is certainly less than p. For the tuples which are outputted by PTJQ, we need to compute the corresponding probability values. We will see in further sections that this still gives us the advantage of pruning.

3 Evaluating PTJQ with Interval Join

To evaluate a PTJQ, common methods like block-nested-loop join and indexed-loop can be used. The advantage of these algorithms is that they have been implemented in typical database systems, and so the system requires little modification to support joins over uncertain data. However, we will demonstrate that these join techniques can in fact be improved by a number of novel techniques.

Figure 3 illustrates a possible approach of using traditional join algorithms for processing uncertainty. As shown in Step 2, the main idea is to join the uncertainty intervals with an interval-join algorithm, and store the possible candidates are stored in a set, C. Subsequently, the pdf/cdf information is used to calculate the probability of each candidate pair, and those that have probability greater than p are retained in the result (Step 3). In the rest of this section, we examine these two steps in more details.

The exact method used in Step 2 depends on the type of the comparison operator. For equality over two uncertain intervals $R_i.U$ and $S_j.U$, we can eliminate intervals that do not overlap after considering the resolution c (i.e., pairs that satisfy $R_i.r + c < S_j.l$ or $S_j.r + c < R_i.l$). According to Definition 4, these tuples have zero chance of being paired up. Thus, any I/O-efficient *overlap join* algorithms over intervals (e.g., [8]) can be used. For the case of $>$, we can immediately eliminate (R_i, S_j) if $R_i.r < S_j.l$. In general, based on the uncertainty operator and uncertainty intervals, we may derive pruning conditions and choose an efficient I/O join algorithm to facilitate pruning.

```
Input
    $R, S$ /* tables containing common uncertainty attributes */
    $\theta_u$ /* uncertainty join operator */
    $p$ /* probability threshold of PTJQ */

Output
    $(R_i, S_j)$ that satisfies $P(R_i \theta_u S_j) > p$

Begin
    1. Let $A \leftarrow \phi$ /* $A$ is the answer of PTJQ */
    2. Let $C \leftarrow \{(R_i, S_j)|$ where $(R_i, S_j)$ are results returned
       by an interval join algorithm over $R_i.U$ and $S_j.U$ $\}$
       ( For $=_c$ and $\neq_c$, join over $[R_i.l-c, R_i.r+c]$, $[S_j.l-c, S_j.r+c]$)
    3. $\forall (R_i, S_j)$ in $C$
           i. if $P(R_i \theta_u S_j) > p$ then $A \leftarrow A \bigcup (R_i, S_j)$
End
```

Figure 3. Evaluating a PTJQ with an interval join.

3.1 Item-Level Pruning

The set C of candidate pairs (R_i, S_j), produced in Step 2, is further refined in Step 3. The refinement process can be done by directly computing the join probability, $P(R_i \theta_u S_j)$ for every pair of (R_i, S_j); only those larger than p are retained. The exact way of computing the this probability depends on the type of uncertainty pdf. For uniform pdf, a closed-form formula can be derived. For Gaussian distribution, the join probability may be implemented by a table lookup. For an arbitrary pdf, $P(R_i \theta_u S_j)$ may not be in closed-form; the join probability can be computed with (relatively expensive) numerical integration methods.

We develop a set of techniques to facilitate the evaluation of Step 3. These methods do not compute $P(R_i \theta_u S_j)$ directly. Instead, they establish *pruning conditions* that can be checked easily to decide whether (R_i, S_j) satisfy the query. They are applicable to any kind of uncertainty pdf, and do not require the knowledge of the specific form of $P(R_i \theta_u S_j)$. They are thus convenient for developing an uncertain database system that supports a wide range of uncertainty pdfs. Moreover, they form the basis of discussions of other pruning techniques in later sections. We term these techniques "item-level-pruning", since pruning is performed based on testing a pair of data items. Let us now discuss the pruning criteria for each operator.

For **Equality** and **Inequality**, we establish the following lemma:

Lemma 1 *Suppose a and b are uncertain-valued variables and $a.U \cap b.U \neq \phi$. Let $l_{a,b,c}$ be $\max(a.l - c, b.l - c)$ and $u_{a,b,c}$ be $\min(a.r + c, b.r + c)$. Then,*

- *$P(a =_c b)$ is at most*
$$\min(a.F(u_{a,b,c}) - a.F(l_{a,b,c}), b.F(u_{a,b,c}) - b.F(l_{a,b,c})) \quad (3)$$
- *$P(a \neq_c b)$ is at least*
$$1 - \min(a.F(u_{a,b,c}) - a.F(l_{a,b,c}), b.F(u_{a,b,c}) - b.F(l_{a,b,c})) \quad (4)$$

Lemma 1 enables us to quickly decide whether a candidate pair $(R_i, S_j) \in C$ should be included into or excluded from the answer, since uncertainty cdfs are known and Equations 3 and 4 can be computed easily. For equality, the lemma allows us to prune away (R_i, S_j) when Equation 3 is less than p; for inequality, we can immediately claim that (R_i, S_j) is the answer when Equation 4 is larger than p. The proof of Lemma 1 is detailed in [4].

For **Greater than** and **Less than**, we have the following Lemma 2.

Lemma 2 *Suppose a and b are uncertain-valued variables. Then, for $a > b$,*

1. *If $a.l \leq b.r < a.r$, $P(a > b) \geq 1 - a.F(b.r)$.*
2. *If $a.l \leq b.l \leq a.r$, $P(a > b) \leq 1 - a.F(b.l)$.*

Again, the proof of Lemma 2 is described in [4]. To understand how this lemma facilitates pruning for $>$, notice that we can immediately include (R_i, S_j) in the answer if $R_i.l \leq S_j.r < R_i.r$ and $1 - R_i.F(S_j.r) \geq p$, since by the first rule of the lemma $P(R_i > S_j)$ has to be larger than p. Observe that (R_i, S_j) can also be included in the answer if $R_i.l > S_j.r$. On the other hand, the second rule of the lemma allows (R_i, S_j) to be excluded from the answer, if the right side expression of $P(a > b)$ has probability value less than p. Notice that (R_i, S_j) can also be excluded from the answer if $R_i.r < S_j.l$. The rules for $<$ in Lemma 2 can be used for pruning in a similar manner.

Given that the pdfs of the uncertain values are known, the above lemmata allow us to perform a constant-time check to decide whether $P(R_i \theta_u S_j)$ has to be evaluated. Thus, for a small overhead, we may be able to avoid the evaluation of actual probabilities in Step 3, which can be expensive. From now on, we assume that checks based on the above lemmata are performed to process the predicate $P(R_i \theta_u S_j)$ in Step 3. In Section 5, we experimentally examine the effectiveness of the framework presented in Figure 3, where we study two common interval join algorithms: block nested loop join (BNLJ) and indexed nested loop join (INLJ).

Notice that the interval-join operation, performed in Step 2, can generate a lot of candidate pairs that are actually not part of the answer (i.e., their probabilities are less than p) The key problem with Step 2 is that it uses uncertainty intervals as the only pruning criterion. In the next section, we examine algorithms that use *both* uncertainty intervals and uncertainty pdfs for pruning, so that a smaller candidate set is produced. In some of these methods, the I/O performance is improved too.

4 Uncertainty-based Joins

Interval joins may not be the best solution because they do not utilize uncertainty pdfs. We now present join algorithms that are tailored for uncertainty. We discuss how to prune at the page level for different uncertainty operators, and how this page-level pruning can be realized in join algorithms.

The discussion focuses on the **equality** ($=_c$) and **greater than** ($>$) operators. The other operators are similar to these and are thus not discussed in detail.

4.1 The Uncertainty Bounds

For database joins like the block-nested-loop join and the indexed-loop-join, the unit of retrieval is a page. Suppose we are given two pages, one from R and the other from S. To perform a join between the uncertain values contained in these two pages, a simple approach is to consider all pairs of values in the two pages. This can be time-consuming, because a page of a modest size can contain many uncertain values[1]. Our goal is "page-level" pruning: with an additional small storage overhead, it can avoid examining the page contents.

The idea of using a small overhead to facilitate the pruning of uncertain values was first proposed in [3] to answer probabilistic threshold range queries – essentially a range query where only uncertain data items that satisfy it with a probability higher than a user-defined threshold are reported. The main idea is to augment some tighter bounds (x-bound) in each node in an interval R-tree. Each x-bound is a pair of bounds that are calculated based on the properties of the uncertainty pdfs associated with the entries stored in that node. Since an x-bound is potentially tighter than the Minimum Bounding Rectangle (MBR), the pruning power can be increased. In this paper, we borrow the idea of x-bound to facilitate page-level joins. Based on the definition of x-bounds for a tree node in [3], we generalize the definition of x-bound for a page:

Definition 9 *Given $0 \leq x < 1$, an* **x-bound** *of a page B consists of two values, called left-x-bound ($B.l(x)$), and right-x-bound ($B.r(x)$). For every uncertain value a stored in B, two conditions must hold:*

- *If $a.l < B.l(x)$, then $a.F(B.l(x)) \leq x$.*
- *If $a.r > B.r(x)$, then $1 - a.F(B.r(x)) \leq x$.*

Essentially, we require that every uncertain attribute stored in a page must have no more than a probability of x of being outside either the left-x-bound or the right-x-bound. We also assume that x-bounds are "tight", i.e., the left-x-bounds (right-x-bounds) are pushed to the right (left) as much as possible. To illustrate, Figure 4 shows a page storing two uncertain attributes, a and b. As we can see, a has a probability less than 0.1 and 0.3 of lying to the left of the left-0.1-bound and left-0.3-bound respectively, i.e., $a.F(B.l(0.1)) \leq 0.1$ and $a.F(B.l(0.3)) \leq x$. Similarly, a cannot have a probability of over 0.3 of being outside the right-0.3-bound. Finally, all the uncertainty intervals must be fully enclosed by the 0-bound, which is akin to the MBR of an index node.

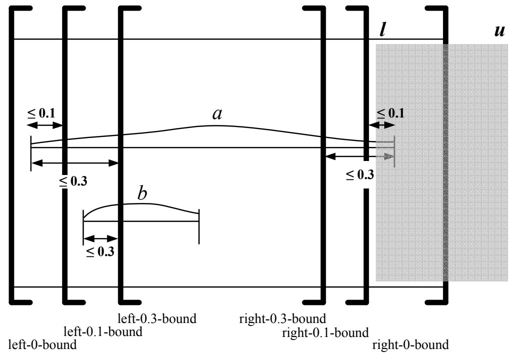

Figure 4. Pruning with x-bounds.

The major purpose of the x-bound is to facilitate pruning for probabilistic threshold range queries. Suppose a range query has a lower bound l, upper bound u and probability threshold p. As shown in Figure 4, if p is larger than 0.4, we are immediately guaranteed that none of the uncertain attributes can satisfy the query: each attribute has a probability of less than 0.3 of being located inside $[l, u]$. We will explain how x-bounds are used to prune in order to process joins effectively.

The implementation of uncertain items and x-bounds in a page is shown in Figure 5. For pdf and cdf, we store the symbol of the type of the distribution, and the parameters relevant to that distribution. For example, if the pdf is Gaussian, then the pdf can be a pair of values (mean, variance), and the cdf may be approximated by a histogram. To implement the x-bounds, we store a table V on the same page, where V_i is a tuple of the form (l, r) for storing the left-W_i-bound and right-W_i-bound. The values of W_i's ($i = 1, \ldots, |W_i|$) are stored in an external table W, sorted in ascending order of W_i's. Our join algorithms require 0-bounds to be stored, with W_1 equal to 0, and $[V_1.l, V_1.r]$ representing the position of the 0-bound. Figure 5 shows the implementation of x-bounds for the example in Figure 4. The total space cost of V and W is $O(|W|)$, which is usually small since only a few x-bounds are stored.

To insert an item to the page, we first compute the x-

[1]For example, if an uncertain attribute uses 8 bytes to store its uncertainty interval, 8 bytes to specify the uniform uncertainty pdf and cdf, a $4K$ page can store 256 items. Joining values in two pages then requires examining $256^2 = 65536$ pairs.

bound of the item. This is usually an inexpensive one-time cost. If the uncertainty pdf is a standard distribution (e.g., uniform), the x-bounds are readily obtained. For an arbitrary pdf (e.g., represented by a histogram), its x-bounds can be derived by scanning the histogram once. Then x-bound of the page is then expanded to accommodate the new item. Readers are referred to [3] for further maintenance details.

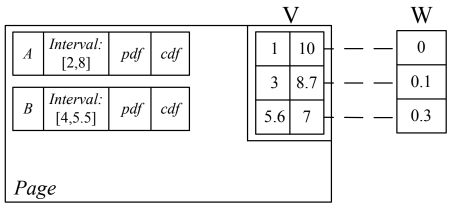

Figure 5. Implementing x-bounds in a page.

```
Input
    [l, u] /* Lower and upper bound of range query Q */
    p /* probability threshold of range query */
    B /* Page with table B.V */
    W /* Global table storing values of x for x-bounds */
Output
    FALSE: All intervals in B are guaranteed to fail Q,
    TRUE otherwise.
(a) CheckLeft($l, u, p, B, W$) /* prune using left-$x$-bounds */
    1. for $i = 1, \ldots, |W|$ do
        (i) if $u < B.V_i.l$ and $W_i < p$ then
            (a) return FALSE
    2. return TRUE
(b) CheckRight($l, u, p, B, W$) /* prune using right-$x$-bounds */
    1. for $i = 1, \ldots, |W|$ do
        (i) if $l > B.V_i.r$ and $W_i < p$ then
            (a) return FALSE
    2. return TRUE
```

Figure 6. `CheckLeft` and `CheckRight`

Given a page B with uncertainty tables, we now present two algorithms (Figure 6) to decide if any uncertain attributes have a probability higher than p of satisfying a range query. Algorithm `CheckLeft` checks the range query against left-x-bounds while Algorithm `CheckRight` employs right-x-bounds for checking. They use the idea illustrated in Figure 4 for pruning, and we state without proof the following lemma.

Lemma 3 *Given a range query Q with interval $[l, u]$ and probability threshold p, if* `CheckLeft` *or* `CheckRight` *returns* `FALSE`*, no uncertain attribute in B can satisfy Q with probability higher than p.*

These two checking routines form the fundamental building blocks for the page-level join operators. They are usually very efficient since only a few x-bounds need to be stored and W is small.

```
Input
    $B_R$ /* Page (with uncertainty bounds) from table $R$ */
    $B_S$ /* Page (with uncertainty bounds) from table $S$ */
    $W$ /* Global table storing values of $x$ for $x$-bounds */
    $c$ /* Resolution of equality */
    $p$ /* probability threshold of equality join */
Output
    (i) PRUNE: $\forall R_i \in B_R, S_j \in B_S$, it is certain that $P(R_i =_c S_j) < p$,
    (ii) CHECK otherwise.
EquiJoin($B_R, B_S, W, c, p$)
    1. if (NOT(CheckLeft($B_R.V_1.l - c, B_R.V_1.r + c, p, B_S, W$))) or
       ( NOT(CheckRight($B_R.V_1.l - c, B_R.V_1.r + c, p, B_S, W$)))
       then return PRUNE
    2. if (NOT(CheckLeft($B_S.V_1.l - c, B_S.V_1.r + c, p, B_R, W$))) or
       NOT(CheckRight($B_S.V_1.l - c, B_S.V_1.r + c, p, B_R, W$)))
       then return PRUNE
    3. return CHECK
```

Figure 7. Page Level Join for Equality.

4.2 Page-Level Equality Join

Using `CheckLeft` and `CheckRight`, a page-level equality join can be constructed easily. Figure 7 illustrates **EquiJoin**, which returns `PRUNE` to indicate that two given pages from R and S do not contain any join pairs with probability over p of being equal, in which case the two pages can be pruned without further investigation. **EquiJoin** returns `CHECK` to indicate that there is a possibility that some pairs satisfying the conditions exist which results in a pairwise evaluation of the values in the pages R and S.

EquiJoin applies two sets of criteria. The first test (Step 1) uses `CheckLeft` and `CheckRight` on page B_S (of table S), using the 0-bound of page B_R (extended with resolution c) to form a range query. In other words, the range query with the interval $[B_R.V_1.l-c, B_R.V_1.r+c]$ is checked against B_S using left- and right-x-bounds. If `CheckLeft` or `CheckRight` returns `FALSE`, by Lemma 3 no uncertain attribute in B_S is in $[B_R.V_1.l - c, B_R.V_1.r + c]$ with a probability higher than p. **EquiJoin** then returns `PRUNE` to indicate that these pages cannot be joined. If Step 1 does not return `PRUNE`, **EquiJoin** uses another set of tests in Step 2, which exchanges the role of B_R and B_S.

The correctness of **EquiJoin** hinges on the four test conditions. Following lemma establishes the correctness of `CheckLeft` in step 1. The other three conditions use the same principles and their proofs are skipped.

Lemma 4 *If* `CheckLeft` *of Step 1 in* **EquiJoin** *returns* `FALSE`*, then for every uncertain values R_i, S_j in B_R, B_S, $P(S_j =_c R_i) < p$.*

Proof : We give a brief proof here, the detailed proof can be found in [4]. Consider any two uncertain values R_i, S_j from B_R, B_S respectively. From Lemma 1, we know that $P(S_j =_c R_i) \leq S_j.F(u_{R_i,S_j,c}) - S_j.F(l_{R_i,S_j,c})$ which we shall show to be less than p. From Lemma 3, we know that no attribute in B_S satisfies the range query formed by $[B_R.V_1.l - c, B_R.V_1.r + c]$ with probability greater than p. Further, any uncertainty interval $R_i.U$ in B_R must be enclosed by $[B_R.V_1.l, B_R.V_1.r]$, and therefore $R_i.r + c \leq B_R.V_1.r + c$. According to Step 1(i) of `CheckLeft` there must be some q such that $B_R.V_1.r + c < B_S.V_q.l$ and $W_q < p$. Therefore,

$$R_i.r + c < B_S.V_q.l \qquad (5)$$

As shown in Figure 8, none of the uncertainty intervals in B_S crosses the line $B_S.V_q.l$ with a fraction of more than W_q. Since S_j lies in B_S, this implies S_j satisfies range query $[R_i.l - c, R_i.r + c]$ with probability $< p$. The overlap of S_j with this range query is given by interval $[l_{R_i,S_j,c}, u_{R_i,S_j,c}]$. Thus, $S_j.F(u_{R_i,S_j,c}) - S_j.F(l_{R_i,S_j,c}) < p$. □

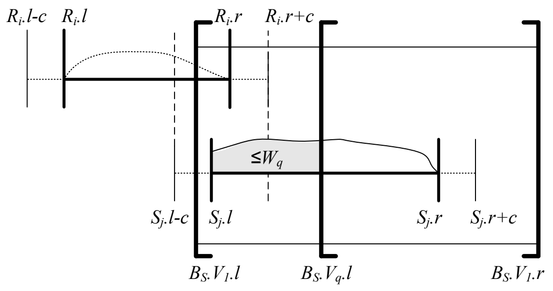

Figure 8. Illustrating the correctness of **EquiJoin**.

Thus Step 1's `CheckLeft` prunes pages correctly. For the remaining criteria, the proofs are skipped due to lack of space. By calling four small testing routines, **EquiJoin** can identify pruning opportunities by using x-bounds of the pages quickly.

4.3 Page-Level Join for "Greater than"

We have developed a page-level pruning algorithm for "$>$" called **GTJoin**. As illustrated in Figure 9, **GTJoin** returns three possible answers. The first type of answer, called `PRUNE`, signals to the caller of **GTJoin** that no interval pairs in the pages concerned have a probability of p or more of being joined (Step 1). The second type of answer, called `INCLUDE`, does the opposite: it informs the user that *every* pair of intervals from B_R and B_S join with probability higher than p, and these pairs can be inserted to the answer without hesitation (Step 2). The final kind of answer, `CHECK`, is returned when neither the conditions in

```
Input
    $B_R$ /* Page (with uncertainty bounds) from table $R$ */
    $B_S$ /* Page (with uncertainty bounds) from table $S$ */
    $W$ /* Global table storing values of $x$ for $x$-bounds */
    $p$ /* probability threshold of $>$ join */
Output
    (i)PRUNE:$\forall R_i \in B_R, S_j \in B_S$,it is certain that $P(R_i > S_j) < p$;
    (ii)INCLUDE:$\forall R_i \in B_R, S_j \in B_S$,it is certain that $P(R_i > S_j) \geq p$;
    (iii) CHECK otherwise.
GTJoin($B_R, B_S, W, p$)
    1. if (NOT(CheckRight($B_S.V_1.l, B_S.V_1.r, p, B_R, W$))) or
          ( NOT(CheckLeft($B_R.V_1.l, B_R.V_1.r, p, B_S, W$)))
       then return PRUNE
    2. if (NOT(CheckRight($B_R.V_1.l, B_R.V_1.r, 1-p, B_S, W$))) or
          ( NOT(CheckLeft($B_S.V_1.l, B_S.V_1.r, 1-p, B_R, W$)))
       then return INCLUDE
    3. return CHECK
```

Figure 9. Page Level Join for $R_i > S_j$.

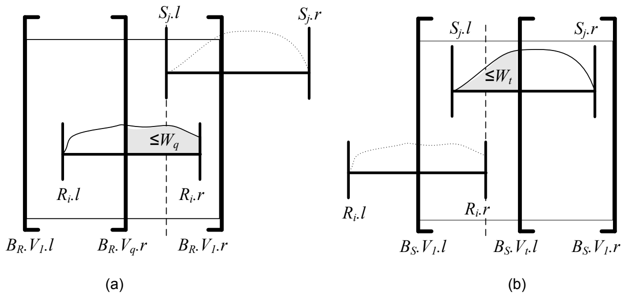

Figure 10. Pruning pages (for $>$).

Step 1 nor those in Step 2 is satisfied. This implies that all pairs must be checked for possible inclusion in the result.

Intuitively, Step 1 first forms a range query by using the 0-bounds of B_S and query it against the right x-bounds of page B_R, by using `CheckRight`. Figure 10(a) illustrates this. If there exists some q such that $B_S.V_1.l \geq B_R.V_q.r$ and $W_q < p$, the page pairs can be pruned (the proof in [4]). If this test fails to prune, another test based on `CheckLeft` is performed, where the range query is formed by the 0-bounds of B_R, querying against the left x-bounds of B_S. The scenario is shown in Figure 10(b).

We can summarize that the function of `CheckRight` and `CheckLeft` of Step 1 is to test whether $P(R_i > S_j) < p$, and if so, "throw away" B_R and B_S. Step 2 performs the opposite: it establishes the conditions in which every pair of items in B_R and B_S can be placed in the answer. Specifically Step 2 verifies the condition $P(S_j > R_i) < 1 - p$, which can be easily achieved by modifying the parameters in Step 1. Since $P(R_i > S_j) = 1 - P(S_j > R_i)$, if any of the two conditions in Step 2 are satisfied, we can conclude that $P(R_i > S_j) \geq p$. **GTJoin** then returns `INCLUDE` to indicate that all combinations of (R_i, S_j) can be inserted to the answer without probing.

Similar to **EquiJoin**, **GTJoin** requires little time as it only calls four small checking subroutines. With this little overhead, the savings can be significant as illustrated in our experiments.

4.4 Uncertainty-enhanced Joins

The page-level pruning techniques can be used to improve the performance of interval or spatial join algorithms that retrieve data in units of pages. Whenever two data pages are compared in the join algorithms, uncertainty tables can be read first, and with our pruning techniques, probing into actual values in the pages can be avoided. Of course, **GTJoin** may not prevent the retrieval of intervals when `INCLUDE` is returned – however, it still improves performance because we can simply add the Cartesian product of the intervals from the two pages to the answer without computing the actual probabilities.

We further illustrate our techniques by studying the example of the Block-Nested-Loop Join (**BNLJ**). In this algorithm, the two relations to be joined are organized as lists of unordered pages. Each page read from the outer relation is matched with each page from the inner relation iteratively, which can be slow because we have to check each pair of intervals from both relations. However, by augmenting each page with an uncertainty table, we can speed up this matching process by using **EquiJoin** or **GTJoin**. We denote the version of **BNLJ** where uncertainty tables are augmented as **Uncertainty-based Block-Nested-Loop Join** (**U-BNLJ** for short). We will compare the performance differences experimentally between these two join algorithms in Section 5. Other page-based join algorithms, such as interval hash join and sort-merge-join, can be enhanced in a similar manner and the details are skipped here.

4.5 Index-level Join

Although uncertainty tables can be used to improve the performance of page-based joins, they do not improve I/O performance, simply because the pages still have to be loaded in order to read the uncertainty tables. However, we can extend the idea of page-level pruning to improve I/O performance, by organizing the pages in a tree structure. Conceptually, each tree node still has an uncertainty table, but now each uncertainty interval in a tree node becomes a Minimum Bounding Rectangle (MBR) that encloses all the uncertainty intervals stored in that MBR. Page-level pruning now operates on MBRs instead of uncertainty intervals. The correctness of these algorithms can be shown easily, by using the fact that each MBR tightly encloses the intervals within the subtree, and arguments similar to Lemma 4.

An implementation of uncertainty tables in the index level is the the Probability Threshold Index (PTI) [3], originally designed to answer probability threshold range queries. It is essentially an interval R-Tree, where each intermediate node is augmented with uncertainty tables. Specifically, for each child branch in a node, PTI stores *both* the MBR and the uncertainty table V of each child. We can use PTI to improve join performance in the framework of the Indexed-Nested-Loop-Join (**INLJ**), by constructing a PTI for the inner relation. The 0-bound of each page from the outer relation is then treated as a range query and tested against the PTI in the inner relation. All pages that are retrieved from the PTI are then individually compared with the page from where the range query is constructed, and our page-level pruning techniques can then be used again to reduce computation efforts.

We denote the version of **INLJ** where PTI is used in place of an interval index as **Uncertainty-based Indexed-Loop Join**, or **U-INLJ** for short. We present the performance results of **INLJ** and **U-INLJ** in the next section.

5 Experiment results

We have evaluated the performance of our pruning methods for the equality operator. We will present the simulation model followed by the results.

5.1 Simulation Model

We simulate the scenario where two sets of sensors were scattered in a region, with one set provides location (one-dimensional) and temperature, and the other provides location and pressure. We consider an equality join query over location attribute values, which produces temperature and pressure information for every location.

Two tables of uncertain data are generated, where the uncertainty pdf is uniform for both datasets. For the first table, uncertainty intervals are uniformly distributed in $[0, 10000]$. The length of each interval is normally distributed with a mean μ of 5 and deviation σ of 1. For the other table, intervals are uniformly distributed in $[5000, 15000]$, and the length is normal with $\mu = 10$ and $\sigma = 2$. Each disk page stores up to 50 tuples. We study the performance of joins over these two tables by evaluating the number of tuple-pair candidates output from the join algorithms (N_{pair}) for item-level pruning, and the number of pairs where probability evaluation has to be performed (N_{prob}).

5.2 Results

Page-Level Pruning Figure 11 shows that **U-BNLJ** performs substantially better than **BNLJ** in N_{pair}. This is because **U-BNLJ** performs page-level pruning while **BNLJ**

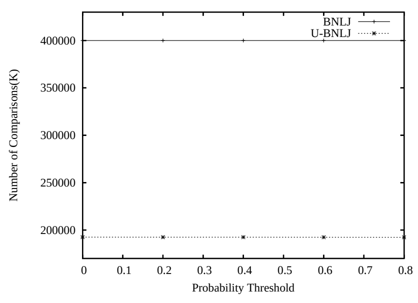

Figure 11. BNLJ and **U-BNLJ**

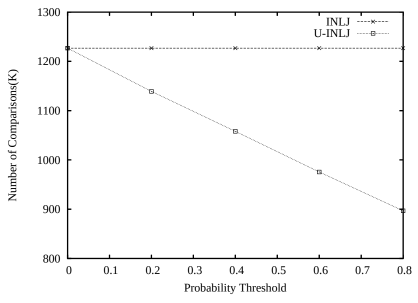

Figure 12. INLJ and **U-INLJ**

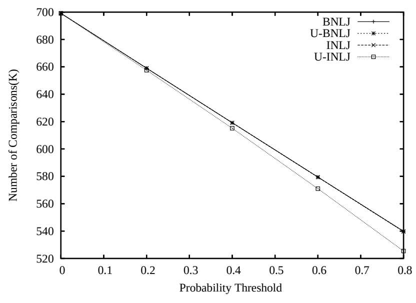

Figure 13. N_{prob} vs p

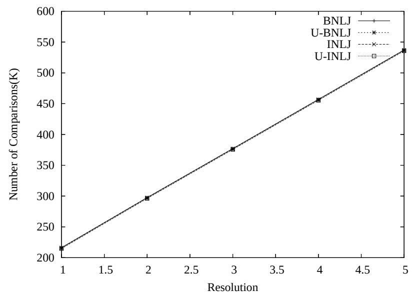

Figure 14. N_{prob} vs c

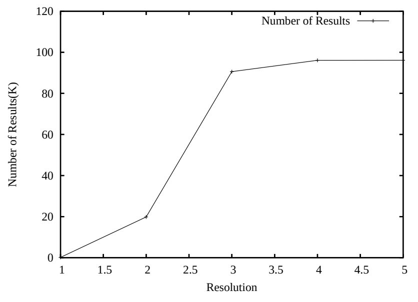

Figure 15. No. of results vs c

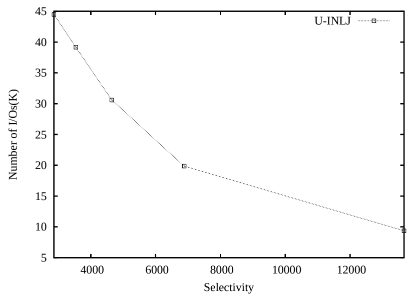

Figure 16. Selectivity on **U-INLJ**

does not. However, **U-BNLJ** does not benefit much from large values of p. Since intervals are stored randomly, intervals in each disk page can be widely spread. Consequently all the x-bounds are close to the 0-bound, and the page-level join cannot exploit p effectively.

Index-Level Pruning The above problem can be alleviated by organizing intervals in a better way, for example, with an index. Figure 12 shows that both **INLJ** and **U-INLJ** address a much better performance in N_{pair} than **BNLJ** and **U-BNLJ**. Further, **U-INLJ** exploits p much better than **INLJ** as uncertainty bounds are used effectively.

Item-Level Pruning Figure 13 shows the number of pairs that we have to compute probability (N_{prob}) for the four joins. We see that the four graphs almost coincide. This means regardless of how many tuple-pairs are produced, the final number of intervals that have to be evaluated is almost the same. This implies our item-level pruning techniques can eliminate a large portion of false positives regardless of the join algorithm. The computational effort due to probability evaluation is reduced significantly.

The effect of **Resolution** for the equality operator is illustrated in Figure 14. We observe that N_{prob} increases with c. With a larger value of c, the uncertainty interval of each tuple is expanded significantly and thus the chance for pruning is reduced. However, the increase in c implies more relaxation of “equality”, potentially returns more answers. This is illustrated in Figure 15. Interestingly, the growth of number of answers saturates as $c > 3$. This indicates that c does not need to be large in order to obtain all possible matches.

Selectivity We also test the effect of join selectivity on **U-INLJ**. Figure 16 shows that **U-INLJ** benefits from high selectivity. When a join is highly selective, **U-INLJ** requires less traversal over the tree, and thus fewer pages need to be retrieved.

Greater Than We present an interesting result for $>$ in Figure 17. We observe that **U-INLJ** does not have the same behavior as in Figure 12. Here N_{pair} does not show a sharp drop as p increases. Recall that in the page-level join for $>$, `INCLUDE` may be returned. When p is very low, there is a high chance for objects to be directly included in the answer. Hence N_{pair} is low when p is low.

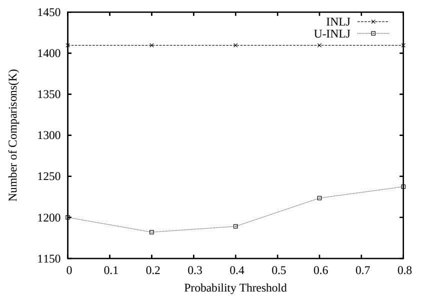

Figure 17. INLJ and **U-INLJ** (for $>$)

6 Related Work

The model for managing uncertain data discussed in this paper is based on [1]. Similar models are proposed

Level	Savings	Applicability	Algorithms
Item	Computation	$=_c, \neq_c, >, <$	BNLJ, INLJ
Page	Computation	$=_c, >, <$	U-BNLJ
Index	I/O & computation	$=_c, >, <$	U-INLJ

Table 1. Pruning Methods for Uncertainty Joins.

in moving-object environments [17, 7] and in sensor networks [6]. Recently, the UDBMS [2] and the Trio System [16] has been proposed to handle such kind of uncertainty. The discussions of uncertainty in other data types can be found in [18]. Another representation of data uncertainty is a "probabilistic database", where each tuple is associated with a probability value to indicate the confidence of its presence [5]. Probabilistic databases have also been recently extended to semi-structured data [12] and XML [10].

Probabilistic queries are classified as value-based (return a single-value) and entity-based (return a set of objects) in [1]. Probabilistic join queries belong to the entity-based query class. Evaluation of probabilistic range queries is discussed in [7, 17, 1, 5]. Nearest-neighbor queries are discussed in [1]. In [1, 5],aggregate value-queries evaluation algorithms are presented. To our best knowledge, probabilistic join queries have not been addressed before. Also these works did not focus on the efficiency issues of probabilistic queries. Although [3] did examine the issues of query efficiency, their discussions are limited to range queries for one-dimensional uncertain data. Recently in [14] the indexing solution for probabilistic range queries has been extended to support uncertain data in high-dimensional space.

There is a rich vein of work on interval joins, which are usually used to handle temporal and one-dimensional spatial data. Different efficient algorithms have been proposed, such as nested-loop join [9], partition-based join [13], and index-based join [19]. Recently the idea of implementing interval join on top of a relational database is proposed in [8]. All these algorithms do not utilize probability distributions within the bounds during the pruning process, and thus potentially retrieve many false candidates. We demonstrated how our ideas can be applied easily to enhance these existing interval join techniques.

7 Conclusions

Uncertainty management is an emerging topic and has attracted research interest in recent years. Indeed, as pointed out in the Lowell Database meeting [15], DBMSs should support imprecision that arises in data acquired by scientific instruments. We identified an important issue in managing data imprecision: the extension of comparison operators for uncertainty and the joining of uncertain-valued attributes. We illustrate how pruning can be achieved at different granularity: item level, page level, and index level (see Table 1). With only a small overhead, these techniques can improve join performance significantly. We intend to extend this work to address join queries over multi-dimensional uncertainty.

References

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

[2] R. Cheng, S. Singh, and S. Prabhakar. UDBMS: A database system for managing constantly-evolving data. In *In Proc. VLDB*, 2005.

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

[4] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter. Effi cient join processing over uncertain data. Technical Report CSD TR# 05-004, Dept. of CS, Purdue University, 2005.

[5] N. Dalvi and D. Suciu. Effi cient query evaluation on probabilistic databases. In *Proc. VLDB*, 2004.

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

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

[8] J. Enderle, M. Hampel, and T. Seidl. Joining interval data in relational databases. In *Proc. SIGMOD*, 2004.

[9] H. Gunadhi and A. Segev. Query processing algorithms for temporal intersection joins. In *Proc. ICDE*, 1991.

[10] E. Hung, L. Getoor, and V. S. Subrahmanian. PXML: A probabilistic semistructured data model and algebra. In *ICDE*, 2003.

[11] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[12] A. Nierman and H. V. Jagadish. ProTDB: Probabilistic Data in XML. In *VLDB*, 2002.

[13] M. Soo, R. Snodgrass, and C. Jensen. Effi cient evaluation of the valid-time natural join. In *Proc. ICDE*, 1994.

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

[15] The Lowell Database Research Self-Assessment Meeting. Lowell massachusetts. May 2003.

[16] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *Proc. CIDR*, 2005.

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

[18] A. Yazici, A. Soysal, B. Buckles, and F. Petry. Uncertainty in a nested relational database model. *Elsevier Data and Knowledge Engineering*, 30, 1999.

[19] D. Zhang, V. Tsotras, and B. Seeger. Effi cient temporal join processing using indicies. In *Proc. ICDE*, 2002.