Efficient Join Processing over Uncertain Data[*]

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

Sarvjeet Singh
Department of Computer Science
Purdue University
West Lafayette, Indiana, USA
sarvjeet@cs.purdue.edu

Sunil Prabhakar
Department of Computer Science
Purdue University
West Lafayette, Indiana, USA
sunil@cs.purdue.edu

Rahul Shah
Department of Computer Science
Purdue University
West Lafayette, Indiana, USA
rahul@cs.purdue.edu

Jeffrey Scott Vitter
Department of Computer Science
Purdue University
West Lafayette, Indiana, USA
jsv@cs.purdue.edu

Yuni Xia
Department of Computer and Information Science
Indiana University - Purdue University Indianapolis
Indianapolis, Indiana, USA
yxia@cs.iupui.edu

ABSTRACT

In many applications data values are inherently uncertain. This includes moving-objects, sensors and biological databases. There has been recent interest in the development of database management systems that can handle uncertain data. Some proposals for such systems include attribute values that are uncertain. In particular, an attribute value can be modeled as a range of possible values, associated with a probability density function. Previous efforts for this type of data have only addressed 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 join queries over uncertain data. We propose a semantics for the join operation, define probabilistic operators over uncertain data, and propose join algorithms that provide efficient execution of probabilistic joins. The paper focuses on an important class of joins termed probabilistic threshold joins that avoid some of the semantic complexities of dealing with uncertain data. For this class of joins we develop three sets of optimization techniques: item-level, page-level, and index-level pruning. 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.

Categories and Subject Descriptors: H.2.4 Database Management: Systems

General Terms: Algorithms, Performance

Keywords: Uncertainty Management, Joins, Imprecise data

[*]Parts of this work were supported by Army Research Office through grant DAAD19–03–1–0321 and National Science Foundation through grants 0534702, 0242421 and 0415097.

1. INTRODUCTION

While traditionally, databases have required data to be modeled in terms of precise values, there are many applications where uncertainty, or imprecision in values is inherent or desirable [4, 5, 14]. Consider, for example, scientific applications that record measurements taken from sensors or other devices. These measurements are many times inexact, with known degrees of errors. GPS devices that give location information are known to have a Gaussian distribution of error around the reported value. Similarly, microarray data in biological experiments are known to have a Lorentzian distribution of error. Sometimes, errors are introduced in order to achieve scalability. Consider the case of sensor databases. It is infeasible (due to resource constraints such as batteries and bandwidth) to continuously monitor every single change in value for every sensor. One solution to this problem while limiting the degree of error is to allow each sensor to not send updates unless the value has changed significantly, or a specified amount of time has elapsed. In this model, called the dead-reckoning apporach [15], the value of the sensor is correctly modeled as a range around the last reported value. Finally, in a Location-Based Service application, users may wish to provide approximate, imprecise locations in order to preserve their privacy.

Given the need for managing uncertain data, several researchers have recently proposed the development of database sytems that manage uncertain data [4, 14, 5, 1]. There are two broad types of data uncertainty that is defined in these works: *tuple-uncertainty* and *attribute-uncertainty*. Tuple uncertainty refers to the uncertainty that a given tuple is part of a database [4]. The tuple itself is no different than regular database tuples. Attribute uncertainty refers to the uncertainty in the value of a given attribute [14, 1].

Incorporating uncertainty into databases brings about many challenges including issues of query semantics, evaluation, and efficiency. The problem of the semantics of query processing and efficient evaluation of queries for tuple uncertainty have been discussed in earlier work [4]. There has also been some work on simple types of queries (range and nearest-neighbors only) for databases with attribute uncertainty [1, 2]. To the best of our knowledge, there is prior work on more complex queries in the specific area of uncertain attribute data. In this paper, we address the issue of joins over databases with uncertain attributes.

As a more concerete example, 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 represent 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 determine which pairs of entities in A and B record the same temperatures. Joining pairs are shown connected by a line in (a). 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 invalid conclusions/decisions may be made.

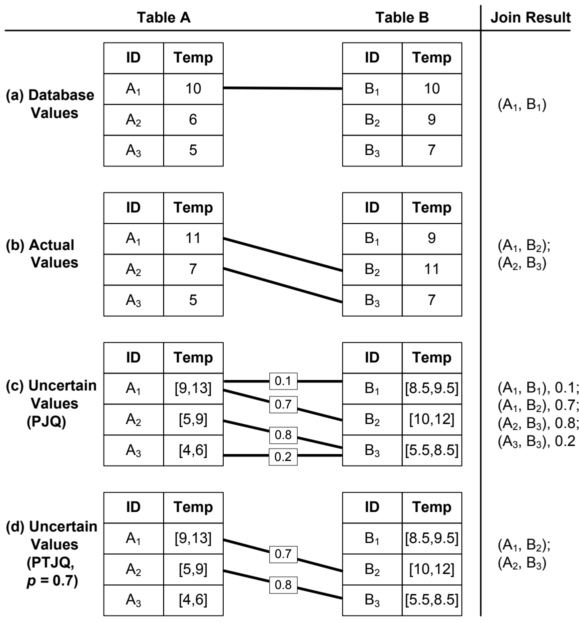

Table A
Table B
Join Result
(a) Database Values
ID | Temp: A1 10, A2 6, A3 5
ID | Temp: B1 10, B2 9, B3 7
(A1, B1)
(b) Actual Values
ID | Temp: A1 11, A2 7, A3 5
ID | Temp: B1 9, B2 11, B3 7
(A1, B2); (A2, B3)
(c) Uncertain Values (PJQ)
ID | Temp: A1 [9,13], A2 [5,9], A3 [4,6]
0.1
0.7
0.8
0.2
ID | Temp: B1 [8.5,9.5], B2 [10,12], B3 [5.5,8.5]
(A1, B1), 0.1; (A1, B2), 0.7; (A2, B3), 0.8; (A3, B3), 0.2
(d) Uncertain Values (PTJQ, p = 0.7)
ID | Temp: A1 [9,13], A2 [5,9], A3 [4,6]
0.7
0.8
ID | Temp: B1 [8.5,9.5], B2 [10,12], B3 [5.5,8.5]
(A1, B2); (A2, B3)

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 was 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 [15]. The pdf of the sensor value within the range may be obtained through machine learning techniques [5]. 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 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, 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 have 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 answer 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. Moreover, the result of a query must return a table as a result. In a regular database model (with no uncertain attributes) the presence of a join pair in the result is boolean depending upon whether or not the join condition is satisfied. For join conditions over uncertain data, the result is generally not boolean, but probabilistic. We propose to solve this problem by associating with each condition over uncertain data, a cut-off threshold. If the probability that the join pair meets the join condition exceeds the threshold, it is included in the result, otherwise the pair is not included. This threshold can either be user-specified or a system parameter.

We term this variant of probabilistic join queries, 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 0.01) are not included in the answer. Compared with Figure 1(c), PTJQ returns fewer false negatives.

In this paper we focus on threshold joins and develop various techniques for the efficient (in terms of I/O and CPU cost) algorithms for 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 data stored in each page; and (3) *index-level pruning*, where all the data stored under a subtree is pruned. These techniques incur a small 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. 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 two components

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 $x \notin a.U$.*

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. [15] 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. [5] 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 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

In order to evaluate join conditions over uncertain attributes, it is first necessary to define operators for this data type. 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 would like to determine the likelihood of them being equal. In this section, we extend the definitions of common comparison operators

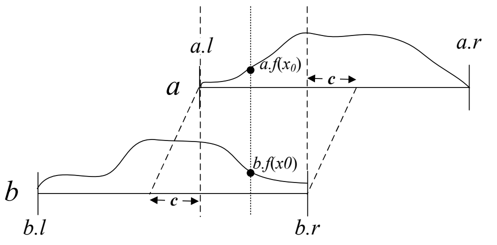

Figure 2: Comparing uncertain values.

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 $[a.l, b.r]$. A first thought is that the probability a equals to b is simply $\int_{a.l}^{b.r} a.f(x)b.f(x)dx$. However, this is incorrect: both $a.f(x)$ and $b.f(x)$ are continuous functions, thus the probability that a and b are equal to x_0 is zero. Consequently, the probability of equality is always zero, and a and b can never be equal.

Given that the exact values for these data 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 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 3. **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 $\int_{x_0-c}^{x_0+c} b.f(x)dx$. 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 3 can be rewritten as:

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

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 4. **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 5. **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, we can redefine $<$ as follows.

DEFINITION 6. **Less than ($<$)**: *$a < b$ with probability $P(a < b)$*

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

Again, for the case that a lies entirely to the left of b, i.e. $a.r < b.l$, $P(a < b) = 1$. Also, for the case that a lies entirely to the right of b, i.e. $a.l \geq b.r$, $P(a < b) = 0$. Also, since $P(a < b)$ is the same as $P(a \leq b)$, and so we will not discuss $a \leq b$.

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.

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 on 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 alongwith the associated probability. Notice that the probability returned by the join is in effect the probability of the correspoding tuple being part of the join result table. Thus the result of a PJQ over a table with uncertain attribute data is a table with tuple uncertainty. Since the model we have considered for uncertainty does not incorporate tuple uncertainty, this result falls outside the model. This is not desirable since we would like to have a closed model in order to enable query composibility.

There are two alternatives for addressing this problem. The first is to treat the probability simply as another attribute of the query result. The new attribute is intrinsically defined with the domain of probability values. This requires users to either be aware that a new attribute will be added, or to explicitly add the probability attribute in their SELECT clauses. The second alternative is not to generate tuples with probabilistic values. Instead, each tuple is either part of the result or not. In this case, we have to convert each probabilitic comparison operator into a boolean comparison operator. This is achieved through the specification of a cut-off threshold probability. With this minor change, we define a join to be *Probabilistic Threshold Join Query* (PTJQ). It has an additional constraint that only join pairs whose probabilities exceed a user-defined threshold are returned.

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.

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, hence the system requires little modification to support joins over uncertain data. However, we will demonstrate that these join techniques can 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 3, these tuples have zero chance of being paired up. Thus, any I/O-efficient *overlap* *join* algorithms over intervals (e.g., [7]) can be used. For $>$, we can immediately eliminate (R_i, S_j) if $R_i.r < S_j.l$, and we can derive similar conditions for $<$. 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 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*
$$\min(a.F(u_{a,b,c}) - a.F(l_{a,b,c}), b.F(u_{a,b,c}) - b.F(l_{a,b,c})) \tag{2}$$
- *$P(a \neq_c b)$ is at least*
$$1 - \min(a.F(u_{a,b,c}) - a.F(l_{a,b,c}), b.F(u_{a,b,c}) - b.F(l_{a,b,c})) \tag{3}$$

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 2 and 3 can be computed easily. For equality, the lemma allows us to prune away (R_i, S_j) when Equation 2 is less than p; for inequality, we can immediately claim that (R_i, S_j) is the answer when Equation 3 is larger than p. The proof of Lemma 1 is detailed in [3].

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

LEMMA 2. *Given uncertain-valued variables a and b:*

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

Again, the proof of Lemma 2 is described in [3]. 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 the price of 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 from R and S respectively. 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 a 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 [2] 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 [2], 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 $\int_{a.l}^{B.l(x)} a.f(y)dy \leq x$.*
- *If $a.r > B.r(x)$, then $\int_{B.r(x)}^{a.r} a.f(y)dy \leq x$.*

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., $\int_{a.l}^{B.l(0.1)} a.f(y)dy \leq 0.1$ and $\int_{a.l}^{B.l(0.3)} a.f(y)dy \leq x$. 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.

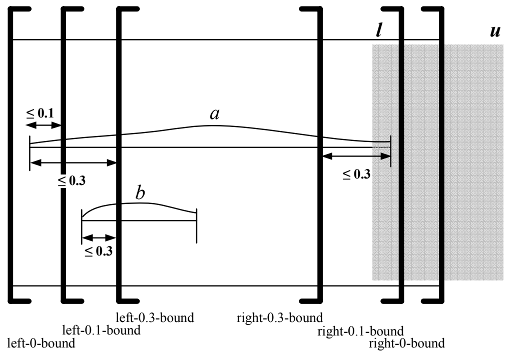

Figure 4: Pruning with x-bounds.

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.

We now discuss the implementation of uncertain items and x-bounds in a page. 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. 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-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. The x-bound of the page is then expanded to accommodate the new item.

Given a page B with uncertainty tables, we now present two algorithms (Figure 5) 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
    $[l, u]$ /* 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 5: `CheckLeft` **and** `CheckRight`

[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.

4.2 Page-Level Equality Join

Using CheckLeft and CheckRight, a page-level equality join can be constructed as follows. Figure 6 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.

```
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 6: **Page Level Join for Equality.**

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 range query is now constructed by using the 0-bound of B_S, and tested against the uncertainty bounds in B_R. Again, **EquiJoin** returns PRUNE if either CheckLeft or CheckRight is FALSE. If none of these tests work, **EquiJoin** concludes that it cannot prune the pages (Step 3).

The correctness of **EquiJoin** hinges on the four test conditions. Below, we establish the correctness when the first testing procedure in Step 1, namely CheckLeft, returns FALSE on pages B_R and B_S. The other three conditions use the same principles and their proofs are skipped. We begin with the following lemma.

LEMMA 4. *If* CheckLeft *of Step 1 in* **EquiJoin** *returns* FALSE, *then for every uncertain value S_j in B_S, its probability of satisfying the range query formed by any uncertainty interval of R_i stored in B_R extended with c, i.e., $[R_i.l - c, R_i.u + c]$, must be less than p.*

PROOF. From Lemma 3, we know that no attributes in B_S satisfies the range query formed by $[B_R.V_1.l - c, B_R.V_1.r + c]$ with probability higher 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,

$$R_i.r + c < B_S.V_q.l \tag{4}$$

As shown in Figure 7, none of the uncertainty intervals in B_S crosses the line $B_S.V_q.l$ with a fraction of more than W_q. This implies no values in B_S can satisfy $[R_i.l - c, R_i.r + c]$ with probability higher than p. □

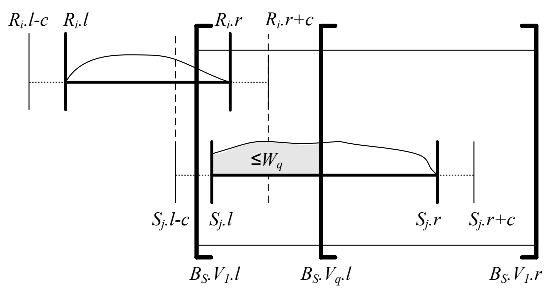

Figure 7: **Illustrating the correctness of EquiJoin.**

For any R_i and S_j stored in pages B_R and B_S, the intersection between $[R_i.l - c, R_i.r + c]$ and $[S_j.l - c, S_j.r + c]$ is given by $[l_{R_i,S_j,c}, u_{R_i,S_j,c}]$, where $l_{R_i,S_j,c}$ is $\max(R_i.l - c, S_j.l - c)$ and $u_{R_i,S_j,c}$ is $\min(R_i.r + c, S_j.r + c)$. The following lemma can be derived.

LEMMA 5. *If* CheckLeft *of Step 1 in* **EquiJoin** *returns* FALSE,

$$S_j.F(u_{R_i,S_j,c}) - S_j.F(l_{R_i,S_j,c}) < p \tag{5}$$

PROOF. Recall from Lemma 4 that S_j with uncertainty interval $[S_j.l, S_j.r]$ satisfies range query $[R_i.l - c, R_i.r + c]$ with a probability less than p. This implies the cumulative probability in the overlap region of $S_j.U$ and $[R_i.l - c, R_i.r + c]$ is less than p, i.e.,

$$S_j.F(\min(R_i.r + c, S_j.r)) - S_j.F(\max(R_i.l - c, S_j.l)) < p \tag{6}$$

We now make the following claims.

Claim 1:

$$S_j.F(\max(R_i.l - c, S_j.l)) = S_j.F(l_{R_i,S_j,c}) \tag{7}$$

PROOF. There are two cases:

(1) $R_i.l - c \geq S_j.l$. Then $R_i.l - c \geq S_j.l - c$, and hence $\max(R_i.l - c, S_j.l)$ is equal to $\max(R_i.l - c, S_j.l - c)$, and thus Equation 7 is correct.

(2) $R_i.l - c < S_j.l$. Then $S_j.F(\max(R_i.l - c, S_j.l)) = S_j.F(S_j.l) = 0$. Moreover, $\max(R_i.l - c, S_j.l - c)$ is either $R_i.l - c$ or $S_j.l - c$; the latter is illustrated in Figure 7. Since $R_i.l - c$ and $S_j.l - c$ are less than $S_j.l$, by Definition 2, both $S_j.F(R_i.l - c)$ and $S_j.F(S_j.l - c)$ are equal to 0. Therefore, Equation 7 is correct. □

Claim 2:

$$S_j.F(\min(R_i.r + c, S_j.r)) = S_j.F(u_{R_i,S_j,c}) \tag{8}$$

PROOF. Recall from Equation 4 that $R_i.r + c$ must be to the left of the left-W_q-bound, as illustrated in Figure 7. Moreover, as $W_q < 1$, $S_j.r$ must be to the right of $B_S.V_q.l$; otherwise the entire interval $S_j.U$ is on the left of the left-W_q-bound, implying that $\int_{S_j.l}^{B_S.V_q.l} S_j.f(y)dy$ is 1, which is larger than W_q and violates Definition 9. Hence, $R_i.r + c$ is less than $S_j.r$, which in turn cannot be larger than $S_j.r + c$. This means $\min(R_i.r + c, S_j.r)$ is the same as $\min(R_i.r + c, S_j.r + c)$, and thus Equation 8 is correct. □

Based on Equations 7 and 8, the left hand side of Equation 6 is the same as

$$S_j.F(\min(R_i.r + c, S_j.r + c)) - S_j.F(\max(R_i.l - c, S_j.l - c))$$

Thus Lemma 5 holds. □

We now prove the correctness of **EquiJoin**. Suppose Step 1 `CheckLeft` returns `FALSE`. 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 is less than p according to Lemma 5. Thus `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 efficiently prune using the page x-bounds.

4.3 Page-Level Join for "Greater than"

We have developed a page-level pruning algorithm for "$>$" called **GTJoin**. As illustrated in Figure 8, **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 Step 1 nor those in Step 2 are satisfied. This implies that all pairs must be checked for possible inclusion in the result.

```
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 8: **Page Level Join for $R_i > S_j$.**

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 9(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. 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 9(b).

The role 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

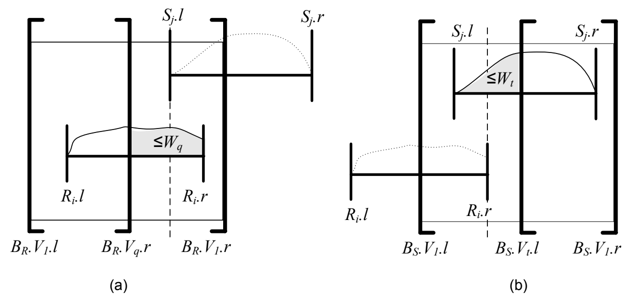

Figure 9: Pruning pages (for $>$).

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 pairs 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) [2], 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. EXPERIMENTAL 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

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 10 shows that **U-BNLJ** performs substantially better than **BNLJ** in N_{pair}. This is because **U-BNLJ** performs page-level pruning while **BNLJ** 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 with an index. Figure 11 shows that both **INLJ** and **U-INLJ** have 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 12 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 13. 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, increase in c implies more relaxation of “equality”, potentially returns more answers. This is illustrated in Figure 14. 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 15 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 16. We observe that **U-INLJ** does not have the same behavior as in Figure 11. 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.

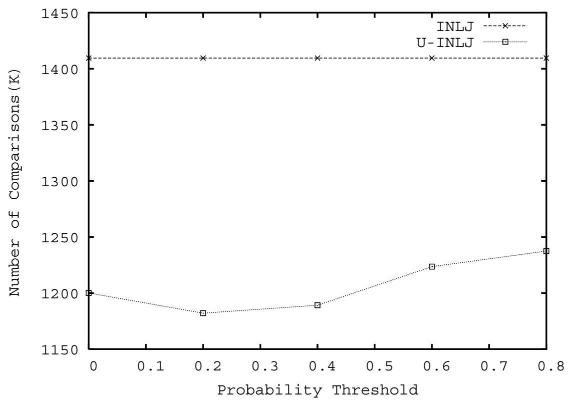

Figure 16: **INLJ and U-INLJ (for $>$)**

6. RELATED WORK

The model for managing uncertain data discussed in this paper is based on [1]. Similar models are proposed in moving-object environments [15, 6] and in sensor networks [5]. Recently, the Trio System [14] has been proposed to handle such uncertainty. Discussion of uncertainty in other data types can be found in [16]. 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 [4]. Probabilistic databases have also been recently extended to semi-structured data [12] and XML [9].

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 [6, 15, 1, 4]. Nearest-neighbor queries are discussed in [1]. In [1, 4],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 [2] did examine the issues of query efficiency, their discussions are limited to range queries.

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 [8], partition-based join [13], and index-based join [17]. Recently the idea of implementing interval join on top of a relational database is proposed in [7]. All these algorithms do not utilize probability distributions within the bounds during the pruning pro-

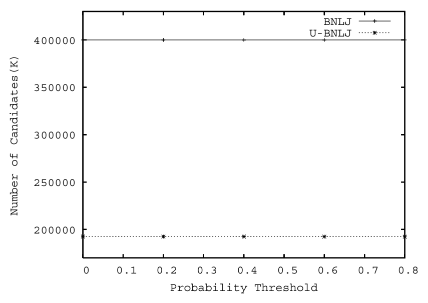

Figure 10: BNLJ and U-BNLJ

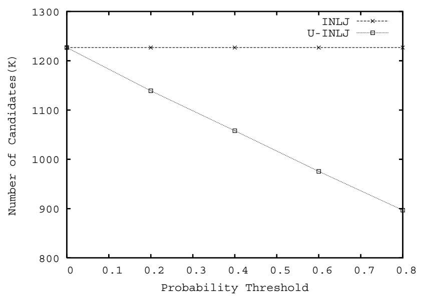

Figure 11: INLJ and U-INLJ

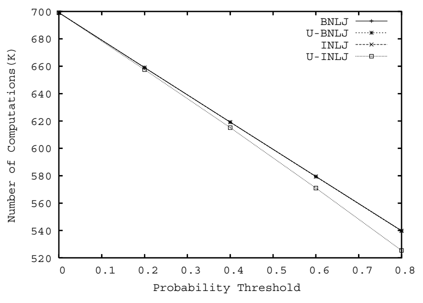

Figure 12: N_{prob} vs p

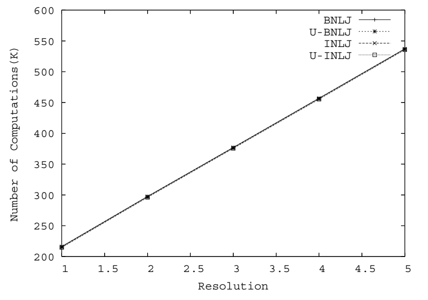

Figure 13: N_{prob} vs c

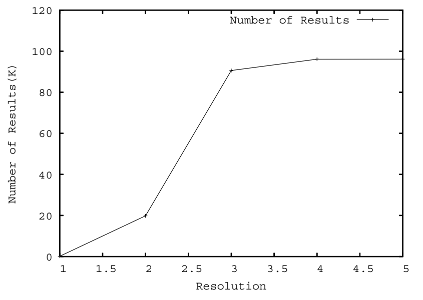

Figure 14: No. of results vs c

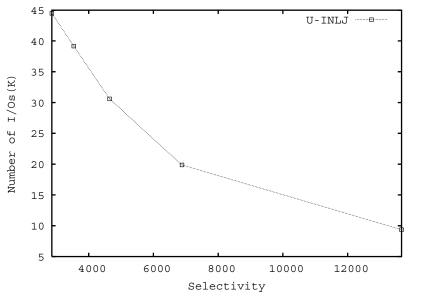

Figure 15: Selectivity on U-INLJ

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.

cess, 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 [10], 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. Joining uncertainty can be costly, and we discussed numerous techniques to reduce the cost. We illustrate how pruning can be achieved at different granularity: item level, page level, and index level. Their properties are summarized in 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.

8. REFERENCES

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

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

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

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

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

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

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

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

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

[10] The Lowell Database Research Self-Assessment Meeting. Lowell massachusetts. May 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. Efficient evaluation of the valid-time natural join. In *Proc. ICDE*, 1994.

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

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

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

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