2005

# Efficient Join Processing over Uncertain Data Technical Report

Reynold Cheng

Yuni Xia

Sunil Prabhakar
*Purdue University*, sunil@cs.purdue.edu

Rahul Shah

Jeffrey S. Vitter
*Kansas University*, jsv@ku.edu

## Report Number:
05-004

# EFFICIENT JOIN PROCESSING OVER UNCERTAIN DATA

**Reynold Cheng**
**Yuni Xia**
**Sunil Prabhakar**
**Rahul Shah**
**Jeffrey Scott Vitter**

**Department of Computer Sciences**
**Purdue University**
**West Lafayette, IN 47907**

**CSD TR #05-004**
**March 2005**

# Efficient Join Processing over Uncertain Data
# Technical Report

Reynold Cheng Yuni Xia Sunil Prabhakar Rahul Shah Jeffrey Scott Vitter

Department of Computer Science, Purdue University, West Lafayette, IN 47907-2066, USA
Email: {ckcheng,xia,sunil,rahul,jsv}@cs.purdue.edu

## Abstract

In database systems that collect information about the external environment, such as temperature and location values, it is often infeasible to obtain accurate information due to measurement and sampling errors, and resource limitations. Queries evaluated over these inaccurate data can potentially yield incorrect results. To avoid these problems, the idea of using uncertainty models (such as an interval associated with a probability density function) instead of a single value for modeling a data item has been explored in recent years. These works 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 join queries over uncertain data. As with other queries over uncertain data, these joins return probabilistic answers. A *probabilistic Join Query* (PJQ) augments the results with probability guarantees to indicate the likelihood of each join tuple being part of the result. Traditional join operators, such as equality and inequality, need to be extended to support uncertain data. In this paper, we present the notion of equality and inequality operators for uncertainty. We also introduce the concept of "approximation" in these comparison operators.

Although PJQs are more informative than traditional joins, they are expensive to evaluate. To overcome this problem, we observe that often it is only necessary to know whether the probability of the results exceeds a given threshold, instead of the precise probability value. By incorporating this constraint into PJQ, 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. Extensive simulation results show that these techniques improve the performance of joins significantly.

## 1 Introduction

There is a lot of ongoing research interest in studying 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 [4]. Locating devices such as cell phones and GPS-equipped handhelds also allow cell 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.

One such application involves the use of *join* operations over "external data". In weather data analysis, for example, it may be interesting to find those times during a particular day two regions have the same temperature. This involves an equality join over temperature values from the two areas. Joins over temperature can also be found in coloring of maps for displaying temperature distribution, where regions with approximately the same temperature are assigned the same color. Joins are found in location-based applications too. For instance, a join query can be used to determine for each moving object, its closest neighbor from among another set of moving objects.

In sensor networks where enormous number of sensors are deployed in a large area, one may want to extract information about the sensor. For example, if the sensor network is used to monitor the temperature of different areas, and we would like to know which areas show the same temperature, we can perform a "self-join" over the whole set of sensor readings. Joins are also useful in error and event detection. For example, due to the low cost of sensors, we can deploy multiple sensors in the same monitoring region to obtain more reliable readings [11]. To find out if there are any unexpected events (e.g., a faulty sensor or a fire), one can perform an equality join among the sensors. If the redundant sensor readings in the region cannot be joined, there are possibly some problems in the region, prompting further investigation. As another example, suppose we know that normally sensor $A$ yields a reading at least as high as sensor $B$. We may perform a "$\geq$" join and if we find that $A$ cannot be matched with $B$, this may indicate there are some problems with either $A$ or $B$. In general, given that some "rules" governing the relationship between the sensors are known, we may perform join queries periodically to identify faulty sensors and surprising events.

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. Data sources such as temperature and pressure sensors provide inherently inaccurate data due to imperfect design of measurement devices. Moreover, while current technologies only allow data to be acquired in a discrete manner, entities such as temperature and location values are continuously changing with time. "Sampling uncertainty" is created, where the information during the inter-arrival time of data samples is not provided to the system. The problem of uncertainty can be aggravated by network issues, where data packets can be delayed or even lost, especially in a wireless network. Hence the database system is only able to get stale and inaccurate versions of the actual values [1, 4].

Data uncertainty can lead to incorrect results for join queries. To illustrate, let us look at Figure 1(a) which shows two tables, $A$ and $B$, storing two attributes $(ID, Temp)$, representing 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). As we can see, since the temperature value of $A_1$ 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 $B_3$, but this is not found from the table instance in Figure 1(a). Consequently, $(A_1, B_2)$ and $(A_2, B_3)$ are not returned to the user, resulting in two *false negatives*.

To avoid drawing incorrect conclusions due to inaccuracy of data, 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

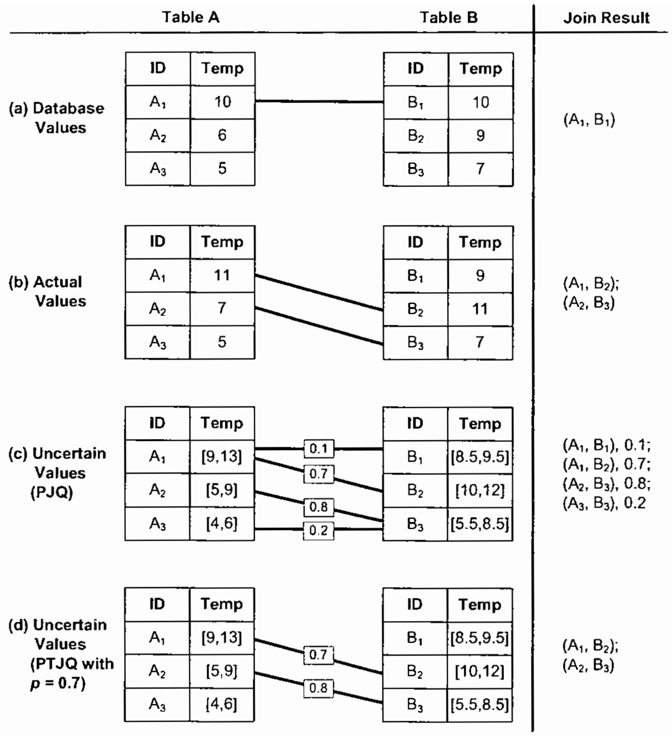


Figure 1: Uncertainty and Join. The link between the tuples indicate a match between the temperature attribute. (a) Join operators use stored temperature values in the database. (b) Join operators use actual temperature values. (c) Probabilistic join use uncertainty models of temperature values to produce probabilistic results. The number on the link represents the probabilistic confidence of the match. (d) Probabilistic threshold join returns tuples with probability at least 0.7.

function (pdf) that describes the probability distribution of the value within the range. By incorporating the notion of uncertainty into data values, imprecise, rather than exact, answers are generated. In particular, each join-pair is associated with a probability to indicate the likeliness the two tuples are matched. We use the term *Probabilistic Join Queries* (PJQ) to describe these types of joins over uncertain data.

To have a better understanding of PJQ and how it improves the quality of answers, let us look at Figure 1(c). 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 likeliness of the join. Notice that both $(A_1, B_2)$ and $(A_2, B_3)$ are now included in the result, in contrast to the situation in Figure 1(a) where these pairs are excluded. In this example, therefore, the false negative problem vanishes, amid that 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.

An interesting question is: how are the 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. The various definitions of join operators allow probability values of each pair of tuples being joined to be computed.

As illustrated in the example in Figure 1(c), PJQ provides stronger guarantees on the answers as compared to traditional joins which do not consider uncertainty. Unfortunately, this advantage does not come without cost; as we will illustrate shortly, these probability values have to be evaluated through costly integration operations. This is much more expensive than traditional joins that only manipulate single attribute values. There is thus a need to reduce the cost of computation of PJQ. An important observation is 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 0.01) are not included in the answer. Compared with Figure 1(c), PTJQ returns fewer false negatives.

More importantly, apart from removing tuple pairs with low confidence, PTJQ can be more efficiently computed than PJQ. This can be achieved through three 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.

The rest of this paper is organized as follows. 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, in detail, the uncertainty data model assumed in this paper. We then present the definitions of comparison operators over uncertainty, based on which probabilistic join queries are defined.

### 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. We treat $a$ as a continuous random variable. The *probabilistic uncertainty* of $a$ consists of two basic components [1]:

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

**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 $x \notin a.U$.*

For our purpose, we also define **uncertainty cdf**:

**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. [12] 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. [4] 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 [9]. Note that although the uncertainty model described here is presented for one-dimensional data, the model and our algorithms can be extended to multiple dimensions.

### 2.2 Uncertainty Comparison Operators

Consider the equality of two uncertain-valued attributes, $a$ and $b$, which are modeled with probabilistic uncertainty. 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 to support uncertain-valued attributes. In particular, we express "imprecision" in these operators in terms of probability values.

Let us examine in detail what "equality" for uncertain data means. Consider the scenario in 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 $a - c \leq b \leq a + c$:

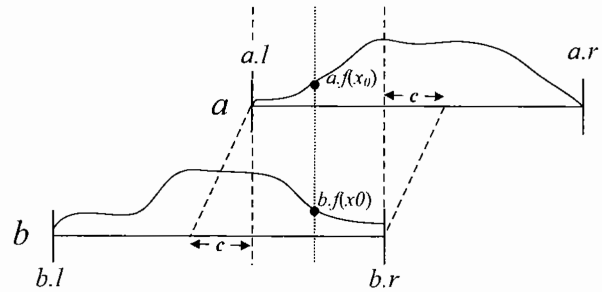


Figure 2: Illustrating Comparison Operations for uncertain values $a$ and $b$.

**Definition 4 Equality ($=_c$):** *Given a resolution $c$, $a$ is equal to $b$ with probability denoted by $P(a =_c b)$, where $P(a =_c b) = \int_{-\infty}^{\infty} a.f(x) \cdot (b.F(x+c) - b.F(x-c))dx$.*

Essentially, $a$ is considered to be equal to the value of $b$ 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 the region $[a.l - c, b.r + c]$. 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 if the uncertainty intervals of $a$ and $b$ have non-zero overlap, their overlap 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 for the case that we are fully confident that $a$ and $b$ cannot be joined. This happens 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 denoted by $P(a \neq_c b)$, where*

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

Now we examine another interesting question: when is $a$ "greater than" $b$? Let us look at Figure 2 again. In the region $[b.r, a.r]$, $b$ has a zero chance of being 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 the region $[a.l, b.r]$, $a$ is only 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 $\int_{a.l}^{b.r} a.f(x)b.F(x)dx$. We do not need to consider the region $[b.l, a.l]$, since $a$ has zero chance of being located in that region, and $a$ is never less than $b$. 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 following definition for ">":

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

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

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. $a.l \ge b.r$, $P(a > b) = 1$.

Note that in a continuous-valued domain, $P(a > b)$ is the same as $P(a \ge b)$ because $a$ can never be exactly equal to $b$. In our subsequent discussions we will not discuss $a \ge b$.

In a similar manner, we can redefine $<$ as follows.

**Definition 7 Less than (<)**: *$a < b$ with probability $P(a < b)$*

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

Once 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. $a.l \ge b.r$, $P(a > b) = 0$.

Again, $P(a < b)$ is the same as $P(a \le b)$, and so we will not discuss $a \le b$.

We can see from these definitions of comparators, comparison is imprecise: they return probability values. However, these probability values indicate the confidence of the comparison result. For example, if $P(a > b) = 0.01$, it indicates $a$ only has a small chance of being greater than $b$.

Before we continue our discussions, it is worth notice that our definitions of comparisons for uncertainty with continuous uncertainty pdfs can be extended to support discrete pdfs.

### 2.3 Comparing Uncertainty with Certainty

In some situations, we may want to join uncertain values with attribute values with no uncertainty. For example, a user may want to join the current locations of moving objects with locations of buildings whose locations are fixed. In general, operators between an uncertain value $a$ and a certain value $v \in \Re$ can be defined as follows:

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

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

### 2.4 Probabilistic Join Queries

Once the comparison operators for uncertainty are defined, we can 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 $i$th 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 8** *Given an uncertainty comparator $\theta_u$ (where $\theta_u$ 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. Although this probabilistic result is correct and more informative that a potentially incorrect result from a traditional join, it involves expensive operations – especially in the process of finding the probabilities of the join pairs using our uncertainty comparators. As pointed out earlier, users may only be interested in join pairs whose probabilities exceed a user-defined *probabilistic threshold.* Using this extra constraint, we show in the subsequent sections that it is possible to evaluate probabilistic joins efficiently, in terms of both computation and I/O. We call this variant of PJQ, defined below, the *Probabilistic Threshold Join Query* (PTJQ).

**Definition 9** *Given an uncertainty comparator $\theta_u$ (where $\theta_u$ 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. As we will see, these two modifications are critical to enhance the performance of join operations.

## 3 Evaluating PTJQ with Interval Join

An initial attempt to evaluate a PTJQ is to use existing join methods such as the block nested loop join, indexed loop join and hash join. The advantage of using these join methods is that many of them have been well implemented in typical database systems, and so the system requires little modification to support joins over uncertain data. Unfortunately, these join methods do not support uncertainty well and they need to be changed.

Figure 3 illustrates a possible approach of using traditional join algorithms for processing uncertainty. For Step 2, the idea is to first ignore the uncertainty pdf and cdf information of the data items. Only the uncertainty intervals are joined using interval-join algorithms, and 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). Since Steps 2 and 3(i) affect the efficiency of the process significantly, they merit further discussion.

Let us use equality as an example to illustrate the details of Step 2. Given uncertain intervals $R_i.U$ and $S_j.U$, we can eliminate intervals which do not overlap after considering the resolution $c$ (i.e., pairs that satisfy $R_i.r + c < S_j.l$ or $S_j.r + c < R_i.l$), since 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., [6]), 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

After Step 2, we obtain a list of candidate pairs $(R_i, S_j)$ based on their uncertainty intervals. We are not sure, however, whether they are in the answer of PTJQ because their probabilities, $P(R_i \theta_u S_j)$, may not be higher than $p$. We could simply compute $P(R_i \theta_u S_j)$ according to their definitions and check whether they are larger than $p$. Unfortunately, this can involve expensive operations. In particular, when the uncertainty pdfs are not simple algebraic expressions, one may need to use numerical methods to perform the integration operations, which can be computationally expensive if a high level of precision is required. In this section, we discuss how we can avoid this costly operation for each comparison operator defined in Section 2.2. These techniques exploit the characteristics of PTJQ and the probability threshold $p$ for efficiency.

The first technique to improve the computation time is to perform a *partial integration*. Suppose we have to perform a numerical integration over an interval $[u, v]$ to find $P(R_i \theta_u S_j)$. This operation typically involves successively computing pieces of the integral over a sequence of small contiguous subintervals. Since the user is not concerned with the actual probability, there is no need to compute the exact value. Instead, once the partial sum of the pdfs over sub-intervals exceeds $p$, we can immediately conclude that $(R_i \theta_u S_j)$ is in the result.

The next set of computation-based techniques exploit the different nature of join operators defined in Section 2.2. It is potentially more powerful than partial integration by providing a chance to skip the integration altogether. We term these techniques "item-level-pruning", since pruning is performed based on testing a pair of data items. Each operator has a separate set of pruning criteria.

**Equality.** To evaluate Step 3 efficiently for equality (Definition 4), we establish the following lemma:

**Theorem 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*

$$\min(a.F(u_{a.b.c}) - a.F(l_{a.b.c}), b.F(u_{a.b.c}) - b.F(l_{a.b.c}))$$

**Proof:** Since $a$ and $b$ overlap at interval $[l_{a.b.c}, u_{a.b.c}]$, from Equation 1 we have

$$\begin{aligned} P(a =_c b) &= \int_{l_{a.b.c}}^{u_{a.b.c}} a.f(x)(b.F(x+c) - b.F(x-c))dx \\ &\leq \int_{l_{a.b.c}}^{u_{a.b.c}} a.f(x)dx \\ &= a.F(u_{a.b.c}) - a.F(l_{a.b.c}) \end{aligned}$$

Similarly, we have $P(b =_c a) \leq b.F(u_{a.b.c}) - b.F(l_{a.b.c})$. Since $P(a =_c b)$ is equal to $P(b =_c a)$, $P(a =_c b)$

cannot be larger than the minimum of $a.F(u_{a.b.c}) - a.F(l_{a.b.c})$ and $b.F(u_{a.b.c}) - b.F(l_{a.b.c})$. Thus the lemma holds.

This lemma enables us to quickly decide which candidate pairs $(R_i, S_j) \in C$ should be included in the answer. Specifically, from the uncertainty cdfs of $R_i.F(x)$ and $S_j.F(x)$, we can obtain in constant time the values of $R_i.F(u_{R_i,S_j,c}) - R_i.F(l_{R_i,S_j,c})$ and $S_j.F(u_{R_i,S_j,c}) - S_j.F(l_{R_i,S_j,c})$. If any of these two values is less than $p$, we can immediately conclude from Lemma 1 that $P_c(R_i, S_j) < p$, and so $(R_i, S_j)$ cannot be part of the answer.

**Inequality.** For inequality, we have the following lemma:

**Theorem 2** *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 $P(a \neq_c b)$ is at least*

$$1-\min(a.F(u_{a,b,c})-a.F(l_{a,b,c}), b.F(u_{a,b,c})-b.F(l_{a,b,c}))$$

which is a direct result of Lemma 1 and Definition 5. Again, this lemma provides us an opportunity for rapid pruning: if the right side of Lemma 2 is larger than $p$, we can immediately include $(R_i, S_j)$ without evaluating $P(R_i, S_j)$. As a reminder, if $R_i.U$ and $S_j.U$ do not overlap, $(R_i, S_j)$ is an answer for $\neq_c$ immediately.

**Greater than.** Lemma 3 illustrates three inequalities for the operator $>$.

**Theorem 3** *Suppose $a$ and $b$ are uncertain-valued variables. Then.*

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

**Proof :** Lemma 3.1 is a direct result from Definition 6. For Lemma 3.2, when $a.l \leq b.l \leq a.r$, $P(a < b)$ is equal to $a.F(b.l) + \int_{b.l}^{\min(a.r,b.r)} a.f(x)(1 - b.F(x))dx$ (Definition 7) , which is larger than or equal to $a.F(b.l)$. Since $P(a > b) = 1 - P(a < b)$, $P(a > b)$ must be smaller than or equal to $1 - a.F(b.l)$.

These three statements offer opportunities for speeding up calculations, depending on the relative positions of $a.U$ and $b.U$. Specifically, 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 Lemma 3.1 $P(R_i > S_j)$ has to be larger than $p$. Notice that $(R_i, S_j)$ can also be included in the answer if $R_i.l > S_j.r$.

On the other hand, Lemma 3.3 allows $(R_i, S_j)$ to be excluded from the answer, if its corresponding right side expressions has probability value less than $p$. Observe that $(R_i, S_j)$ can also be excluded from the answer if $R_i.r < S_j.l$.

**Less than.** Finally, we state without proof Lemma 4 for operator $<$. It is a direct consequence of Lemma 3.

**Theorem 4** *Suppose $a$ and $b$ are uncertain-valued variables. Then.*

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

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 expensive evaluation of actual probabilities in Step 3. From now on, we assume that checks based on the above lemmata and partial integration 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 compare two common interval join algorithms: block nested loop join (BNLJ) and indexed nested loop join (INLJ).

Although the lemmata developed for Step 3 provide potential speedup in computation performance, they offer limited improvement. In particular, 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 $p$), affecting the performance of Step 3. The key problem with Step 2 is that it uses uncertainty intervals as the only pruning criterion. In the next section, we examine join 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

The performance of Step 2 in Figure 3 is essential to the overall performance since it eliminates some I/O operations. As explained above., interval joins may not be the best solution because they do not utilize uncertainty pdfs. In this section, we discuss join algorithms that are tailored for uncertainty. We first discuss how to prune at the page level for different uncertainty operators. Next we study how this page-level pruning can be realized in different 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

In typical database join algorithms, such as block-nested-loop join and indexed-loop-join, the unit of retrieval is a page. Suppose we are given two instances of 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 uncertain values contained 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 is possible to avoid examining the intervals of $R$ and $S$.

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. Their 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 10** *Given $0 \le x < 1$, 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 attribute $a$ stored in $B$, two conditions must be satisfied:*

- *If $a.l < B.l(x)$, then $\int_{a.l}^{B.l(x)} a.f(y)dy \le x$.*
- *If $a.r > B.r(x)$, then $\int_{B.r(x)}^{a.r} a.f(y)dy \le x$.*

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 better understand the concept of $x$-bound, let us take a look at Figure 4 that 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., $\int_{a.l}^{B.l(0.1)} a.f(y)dy \le 0.1$ and $\int_{a.l}^{B.l(0.3)} a.f(y)dy \le x$. 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.

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 $[l, u]$. Here it is worth mentioning that the method of using $x$-bounds for data

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

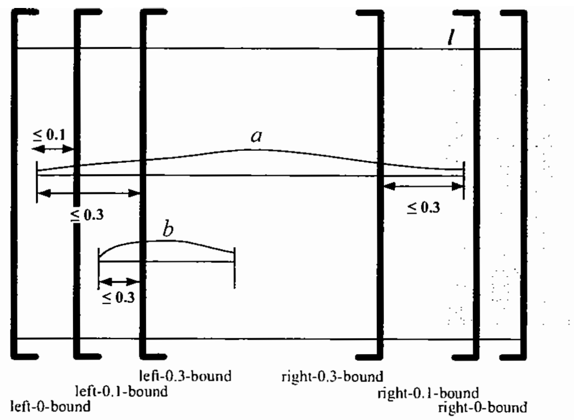


Figure 4: 0-bound, 0.1-bound and 0.3-bound. A range query $[l, u]$ with $p = 0.4$ is also shown.

pruning does not assume that the uncertainty pdfs of all data belong to the same type. It is thus more flexible compared with variance-based clustering, another method proposed in [2] which assumes homogeneous uncertainty pdfs. As we will see soon, $x$-bounds can be used to prune in order to process joins effectively.

Implementing the $x$-bounds for a page is simple. 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. Inserting and deleting uncertain data to and from the page requires expanding and shrinking of $x$-bounds, respectively. This can be achieved as described in [2].

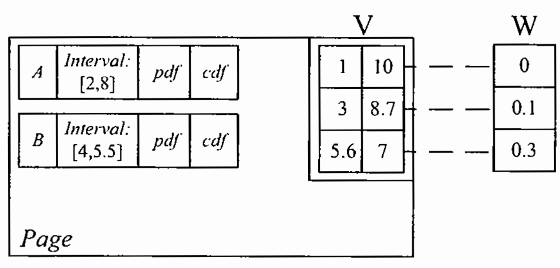


Figure 5: Implementing $x$-bounds in a page.

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.

**Theorem 5** *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 6: Algorithms for deciding whether a page $B$ can be pruned for a range query. (a) CheckLeft uses left-$x$-bounds for pruning. (b) CheckRight uses right-$x$-bounds for pruning.

## 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 5 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.

---

**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.

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. In the rest of this section, 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.

**Theorem 6** *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 5, 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,

$$R_i.r + c < B_S.V_q.l \tag{2}$$

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 $W_q$. This implies no values in $B_S$ can satisfy $[R_i.l - c, R_i.r + c]$ with probability higher than $p$.

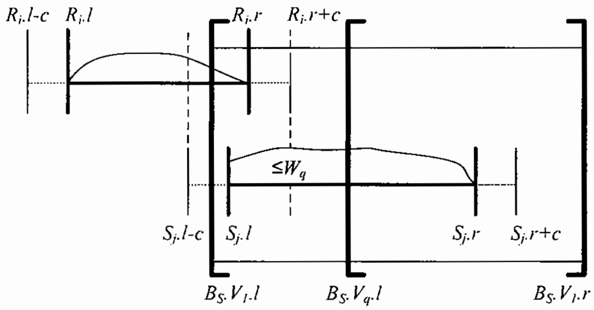


Figure 8: Illustrating the correctness of **EquiJoin**.

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.

**Theorem 7** *If* `CheckLeft` *of Step 1 in* **EquiJoin** *returns* `FALSE`, *then*

$$S_j.F(u_{R_i.S_j.c}) - S_j.F(l_{R_i.S_j.c}) < p \tag{3}$$

**Proof :** Recall from Lemma 6 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.,

$$S_j.F(\min(R_i.r + c, S_j.r)) - S_j.F(\max(R_i.l - c, S_j.l)) < p \tag{4}$$

We now make the following claims.

**Claim 1:**

$$S_j.F(\max(R_i.l - c, S_j.l)) = S_j.F(l_{R_i.S_j.c}) \tag{5}$$

**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 5 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 8. 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 5 is correct.

**Claim 2:**

$$S_j.F(\min(R_i.r + c, S_j.r)) = S_j.F(u_{R_i.S_j.c}) \tag{6}$$

**Proof :** Recall from Equation 2 that $R_i.r + c$ must be to the left of the left-$W_q$-bound, as illustrated in Figure 8. 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 10. 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 6 is correct.

Based on Equations 5 and 6, the left hand side of Equation 4 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 7 holds. It is now easy to prove the correctness of **EquiJoin**. Suppose Step 1's `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 7. 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 Step 1 nor those in Step 2 is satisfied. This implies that all pairs must be checked for possible inclusion in the result.

Although **GTJoin** also uses primitives `CheckLeft` and `CheckRight`, it is different from **EquiJoin** in the way these primitives are used. To understand how the algorithm works, let us first examine `CheckRight` of Step 1. We will prove the correctness of **GTJoin** through Lemma 8.

**Theorem 8** *When* `CheckRight` *of Step 1 returns* `FALSE`, *any uncertain value $R_i$ in $B_R$ must have a probability of less than p for being greater than any $S_i$ in $B_S$.*

**Proof :** If `CheckRight` is `FALSE`, according to Lemma 5 no uncertain value in $B_R$ satisfies the range query constructed by the 0-bound of $B_S$ (i.e,$[B_S.V_1.l, B_S.V_1.r]$) with probability higher than $p$. Also, there exists some $q$ such that $B_S.V_1.l \geq B_R.V_q.r$

**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 $P(R_i > S_j)$
(ii)INCLUDE:$\forall R_i \in B_R, S_j \in B_S$,it is certain that $P(R_i > S$
(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 $R_i > S_j$.

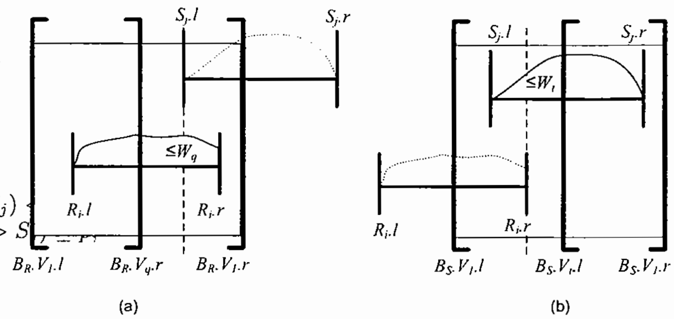


Figure 10: Pruning pages for $>$, using (a) right-$x$-bounds of $B_R$, and (b) left-$x$-bounds of $B_S$.

and $W_q < p$. Since the lower bound of any uncertainty interval $S_j$ in $B_S$ is not less than $B_S.V_1.l$, $S_j.l \geq B_R.V_q.r$. Figure 10(a) illustrates the situation [2]. We can see that the overlap region between any $R_i$ and $S_j$, i.e., $[S_j.l, R_i.r]$, addresses a total probability of not more than $W_q$ for $R_i$ i.e., $1 - R_i.F(S_j.l) < W_q$, which is less than $p$. This implies that the cumulative probability of $R_i$ from the $R_i.l$ to $S_j.l$ is larger than $1-p$, i.e.,

$$R_i.F(S_j.l) > 1 - p \qquad (7)$$

Since $R_i.l \leq S_j.l \leq R_i.r$, according to Lemma 3.2, we have

$$\begin{aligned} P(R_i > S_j) &\leq 1 - R_i.F(S_j.l) \\ &\leq 1 - (1-p) \quad \text{(by Equation 7)} \\ &= p \end{aligned}$$

Therefore Lemma 8 holds.

Another test criterion in Step 1 applies CheckLeft, where the roles of $B_R$ and $B_S$ are switched, and the range query is formed by the 0-bounds of $B_R$. Figure 10(b) illustrates a typical scenario where this criterion is applied to. Its proof is skipped due to space limitation.

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** only needs to call four small checking routines.

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

So far we have discussed different pruning criteria for comparing two pages using uncertainty bounds resident on the pages. These techniques can be applied to traditional interval or spatial join algorithms to improve performance on processing uncertain data, which 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 prevented. 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 now illustrate how our techniques can be applied to a simple join algorithm: the Block-Nested-Loop Join (**BNLJ**). In this algorithm, the two relations to be joined are organized as a list of unordered pages. For each page read from the outer relation, it is matched with each page from the inner relation in an iterative manner, 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 al-

[2] The case when $S_j$ does not overlap with $R_i$ i.e., $S_j.l > R_i.r$, is not shown in Figure 10. In that case $R_i$ is certainly less than $S_j$.

gorithms 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 join algorithms, they do not improve the I/O performance, simply because the pages still have to be loaded in order to read the uncertainty tables on the pages. However, we can extend the idea of page-level pruning to have a better 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 Maximum 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 6.

An implementation of uncertainty tables in the index level is the Probability Threshold Index (PTI) [2], originally designed to answer range queries over uncertain data with probabilistic thresholds. It is essentially an interval R-Tree, where each intermediate node is augmented with uncertainty tables. Sepcifically, 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 have implemented **U-INLJ** and found that it is experimentally better than **INLJ**, as described in the next section.

## 5 Experiment results

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

### 5.1 Simulation Model

We generated two tables of uncertain data, 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 $\mu$ of 5 and deviation $\sigma$ 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 probability evaluations performed ($N_{prob}$). Notice that each "probability evaluation" is expensive because of the costly integration operation involved in finding the probability – which is done when pruning techniques fail. Ideally $N_{prob}$ should be small.

### 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** does not. However, **U-BNLJ** does not benefit much from large values of $p$. Since intervals are randomly stored, intervals in each disk page can be widely spread. Consequently the $x$-bounds are close to the boundary, 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 illustrates that both **INLJ** and **U-INLJ** address a much better performance in $N_{pair}$ than **BNLJ** and **U-BNLJ**. Further, **U-INLJ** exploits the probability threshold $p$ 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 $N_{prob}$ 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 mores answers. This is illustrated in Figure 15. Interestingly, the growth of number of answers saturate as $c > 3$. This indicates $c$ does not need to be

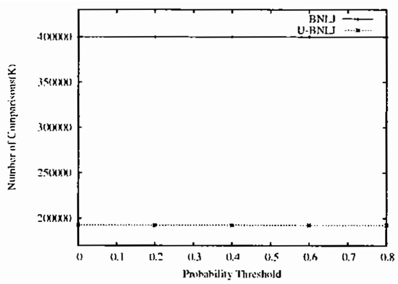


Figure 11: **BNLJ** and **U-BNLJ**

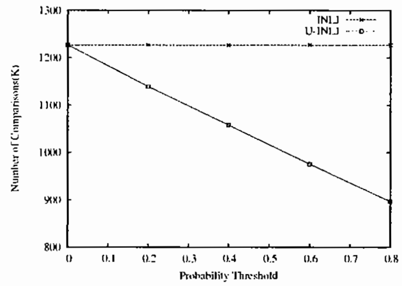


Figure 12: **INLJ** and **U-INLJ**

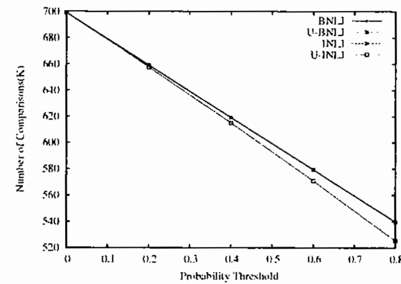


Figure 13: $N_{prob}$ vs $p$

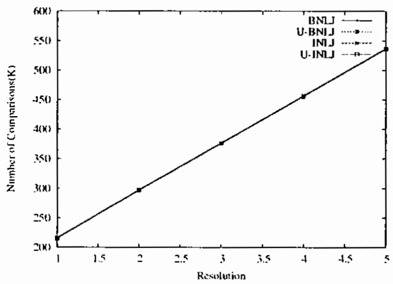


Figure 14: $N_{prob}$ vs $c$

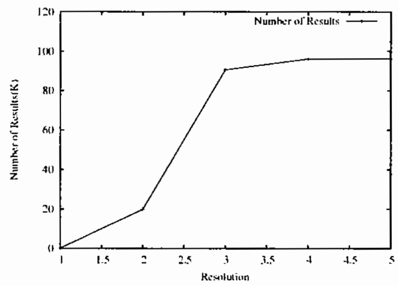


Figure 15: No. of results vs $c$

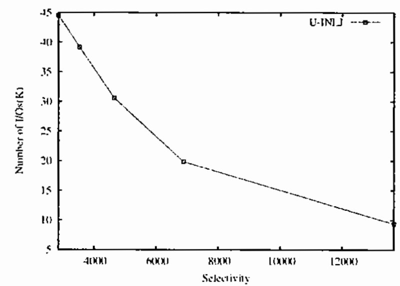


Figure 16: Effect of Selectivity on **U-INLJ**

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 less number of pages need to be retrieved.

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

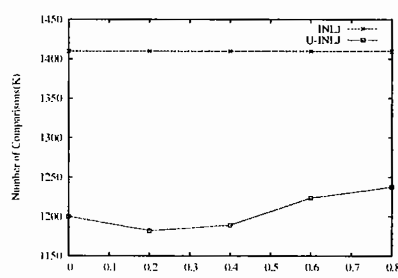


Figure 17: **INLJ** and **U-INLJ** (for $>$)

## 6 Related Work

The data uncertainty model assumed in this paper is based on the work of [1]. While the uncertain intervals are time-varying functions in their paper, we assume the lengths of uncertain intervals are time-independent. Uncertainty models can also be found in moving-object environments [12, 5], and more recently in sensor networks [4]. The discussions of uncertainty in other data types can be found in [13]. Another representation of data uncertainty is the "probabilistic database", where each tuple is associated with a probability value to indicate the confidence of its presence [3].

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 entity-based query class. Evaluation of probabilistic range queries can be found in [5, 12, 1, 3]. Nearest-neighbor queries are discussed in [1]. In [1, 3],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 in interval join, which are usually used to handle temporal and one-dimensional spatial data. Different efficient algorithms have been proposed, such as nested-loop join [7], partition-based join [10], and index-based join [14]. Recently the idea of implementing interval join on top of a relational database is proposed in [6]. All these algorithms do not utilize probability distributions within the bounds during the pruning process,

| 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.

and thus potentially retrieve a lot of 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 [8], 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.

## References


[1] R. Cheng, D. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, June 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. of the 30th Intl. Conf. on Very Large Data Bases*, 2004.

[3] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *Proc. of the 30th Intl. Conf. on Very Large Data Bases*, 2004.

[4] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *Proc. of the 30th Intl. Conf. on Very Large Data Bases*, 2004.

[5] D.Pfoser and C. Jensen. Capturing the uncertainty of moving-objects representations. In *Proceedings of the SSDBM Conference*, pages 123–132, 1999.

[6] J. Enderle, M. Hampel, and T. Seidl. Joining interval data in relational databases. In *Proc. of the 2004 ACM SIGMOD Intl. Conf. on Management of Data*, 2004.

[7] H. Gunadhi and A. Segev. Query processing algorithms for temporal intersection joins. In *Proc. of the Intl. Conf. on Data Engineering*, 1991.

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

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

[10] M. Soo, R. Snodgrass, and C. Jensen. Efficient evaluation of the valid-time natural join. In *Proc. of the Intl. Conf. on Data Engineering*, 1994.

[11] H. Wang, K. Yao, G. Pottie, and D. Estrin. Entropy-based sensor selection heuristic for localization. In *3rd Intl. Workshop on Inofrmation Processing in Sensor Networks (IPSN'04)*, 2004.

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

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

[14] D. Zhang, V. Tsotras, and B. Seeger. Efficient temporal join processing using indicies. In *Proc. of the Intl. Conf. on Data Engineering*, 2002.