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 Shab  Jeffrey Scott Vitter
Department of Computer Science, Purdue University, West Lafayette, IN 47907-2066, USA
Email: {ckcheng,xia,sunil,rallLlsv}@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 focused 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 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. Sensors, for example, can perceive 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 used in coloring of maps for displaying temperature distribution. Regions with approximately the same temperature are assigned the same color. Join queries are also found in location-based applications too. For instance, a join query may 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, 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 ob-
tain 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 possible noise problems in the region, prompting further investigation. An other example, suppose we know that normally sensor A yields a reading at least as high as sensor B. We may perform a "2:" 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 gen-eral, 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 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, the system. The problem of uncertainty can be aggregated 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 A1 is different from that of B1, A1 should not be paired with B1. Instead, A1 matches B2, where both temperature values equal 11°F. Thus there is a false positive in the true result — (A1, B3) is wrongly returned to the user. Figure 1(b) also shows that A3 should be matched with B3, but this is not found from the table instance in Figure 1(a). Consequently, (A1, B3) and (A3, B3) 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 just a numerical value to describe an item is proposed in [1]. Each item is associated with a range of possible values and a probability density function (pdf) that describes the probability distribution of the value within the range. By incorporating the notion of uncertainty into data values, incorrect, rather than exact, answers are generated. In particular, each join-pair is associated with a probability to indicate the likelihood 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 likelihood of the join. Notice that both (A1, B2) and (A1, B3) 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, and that we have a 0.7 and 0.8 confidence for these pairs. On the other hand, the false positive, A1, B3, remains in the result, and a

![Figure 1: Uncertainty and Join.](image-url)

<table>
<thead>
<tr>
<th>Type</th>
<th>Table A</th>
<th>Table B</th>
<th>Join Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID Temp</td>
<td>A1</td>
<td>B1</td>
<td>A1, B1</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A1</td>
<td>Temp B1</td>
<td>A1</td>
<td>B1</td>
</tr>
<tr>
<td>(ID, Temp)</td>
<td>(ID, Temp)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>38</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>42</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>44</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>46</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>48</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>54</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>56</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>58</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>62</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>66</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>68</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>70</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>72</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>74</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>76</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>78</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>82</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>84</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>86</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>88</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>90</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>92</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>94</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>96</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>98</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

By using 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 likelihood of the join. Notice that both (A1, B2) and (A1, B3) 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, and that we have a 0.7 and 0.8 confidence for these pairs. On the other hand, the false positive, A1, B3, remains in the result, and a
The various definitions of join operators allow proba-
stand the semantics of join operators for uncertainty.
The notions of equality and inequality have to be ex-
tended to support uncertain data. We will address
the new definitions of comparison operators for the uncer-
tain data model. Furthermore, we demonstrate how
it is possible to relax the requirements for comparison
operators, in order to allow more flexibility in speci-
fying accuracy requirements of joins over uncertainty.
The various definitions of join operators allow proba-
bility 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 com-
pared to traditional joins which do not consider uncer-
tainty. Unfortunately, this advantage does not come
without cost; as we will illustrate shortly, these prob-
bability values have to be evaluated through costly in-
tegration operations. This is much more expensive
than traditional joins that only manipulate single at-
tribute values. Often the user is only concerned about whether the proba-

probability value exceeds a given threshold. We term the
"new" variant of PJQ which only returns tuple pairs when
their probabilities exceed a certain threshold as Prob-
abilistic 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 proba-
bilities exceed threshold $p = 0.7$. As a result, the two
pairs with low probability values (0.1 and 0.01) are not in-
cluded in the answer. Compared with Figure 1(c),
PTJQ returns fewer false negatives.

Moving 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 integrated into existing join algorithms eas-
ily.

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 as-
sumed in this paper, and various notions of join oper-
ators over uncertainty. Section 3 presents item-level
pruning techniques for each join operator. In Sec-
3.2 Comparing Uncertain Values

In this section, we describe, in detail, the uncertainty
model assumed in this paper. We then present
the definitions of comparison operators over uncer-
tainty, based on which probabilistic join queries are de-
fined.

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 $a$ and its distribution $F_a(x)$.

$F_a(x)$ is a probability distribution function of $a$, such that $F_a(x) = \int_{-\infty}^{x} f_a(y)dy$. Notice that $F_a(x) = 0$ if $x < a$ and $F_a(x) = 1$ if $x > a$.

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 $a.F(x) = \int_{-\infty}^{x} a.f(y)dy$.

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

The exact realization of this model is application-de-
dependent. For example, in modeling sensor measure-
ment uncertainty, $a.U$ is an error bound and $f_a(y)$ 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 in 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 \( f(U) \) is a uniform distribution i.e., \( f(x) = \frac{1}{2b-2a} \) for a \( x \in [a, b] \). 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 [5]. 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 \( a \) and \( b \) is \([a, b] \). A first thought is that the probability \( a \) equals to \( b \) is simply \( \int_a^b f(x) f(y) dx dy \). However, this is incorrect: both \( f(x) \) and \( f(y) \) are continuous functions, thus the probability that \( a \) and \( b \) are equal to \( z \) 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 should be determined by the user. Based upon this observation, we define equality using a parameter, called resolution (\( \varepsilon \)), as: \( a \) is equal to \( b \) if they are within \( \varepsilon \) of each other, i.e., \( b - \epsilon \leq a \leq b + \epsilon \) or \( a - \epsilon \leq b \leq a + \epsilon \).

\[ P(a = b) = \int_{a-\epsilon}^{a+\epsilon} \int_{b-\epsilon}^{b+\epsilon} f(x) f(y) dx dy \]

\[ P(a \neq b) = 1 - P(a = b) \]

To illustrate another interesting question when a is smaller than b let us look at Figure 2 again. In the region \([b, r, a]\), b has a zero chance of being larger than a, since \( f(z) \) is 0 when \( b > r \). Thus, if a is within \([b, r, a]\), it is larger than b with probability \( \int_a^b f(x) dx \), or 1 - \( a \cdot P(b) \). At any point \( x_0 \) inside the region \([a, b, h] \), a is only larger than b if \( x_0 \) is greater than b.
with a probability \( a.F(v)F(x)dx \). Where \( F(x) \) is the probability that \( x \) is less than \( x \). Therefore, in \([a, b], \) the probability that \( a \) is larger than \( b \) is given by \( \int_{b}^{a} a.F(v)F(x)dx \). We do not need to consider the region \([b, a], \) 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:

\[
\int_{b}^{a} a.F(v)F(x)dx + 1 - a.F(b) = 0.5
\]

Upon considering all possible scenarios of overlap between \( a.U \) and \( b.U \), we obtain the following definition for \( \sim \):

**Definition 6 Greater than (\( \succ \)):** \( a \succ b \) with probability \( P(a \succ b) = \frac{1}{2} + \frac{1}{2} a.F(b) \)

For the case that \( a \) lies entirely to the left of \( b, i.e. a < b \), \( P(a < b) = 0 \). Also, for the case that \( a \) lies entirely to the right of \( b, i.e. a > b \), \( P(a > b) = 1 \). Note that in a continuous-valued domain, \( P(a > b) \) is the same as \( P(a < b) \) because \( a \) can never be exactly equal to \( b \). In our subsequent discussions we will not discuss a \( a \succ b \).

In a similar manner, we can redefine \( < \) as follows.

**Definition 7 Less than (\( \prec \)):** \( a \prec b \) with probability \( P(a \prec b) = \frac{1}{2} - \frac{1}{2} a.F(b) \)

Once again, for the case that \( a \) lies entirely to the left of \( b, i.e. a < b \), \( P(a < b) = 1 \). Also, for the case that \( a \) lies entirely to the right of \( b, i.e. a > b \), \( P(a > b) = 0 \). Again, \( P(a = b) = 0 \), and so we will not discuss \( a \equiv b \).

We can see from these definitions of comparators, comparison in 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 \( a \) has only a small chance of being greater than \( b \).

Before we continue our discussion, it is worth noting the definitions of comparisons for uncertainty with continuous uncertainty polls can be extended to support discrete polls.

### 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 \( x \) can be defined as follows:

\[
P(a > x) = \int_{x}^{\infty} a.F(v)F(x)dx = a.F(x) - a.F(\infty)
\]

\[
P(a = x) = 1 - P(a > x) = 1 - a.F(x)
\]

\[
P(a < x) = a.F(x)
\]

which can be treated as special cases for the definitions of uncertain 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 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 higher than a given threshold.

**Definition 8** Given an uncertainty comparator \( \epsilon_{u} \) (where \( \epsilon_{u} \) is any one of \( =e, =l-e, =l-e \)), a Probabilistic Join Query (PJQ) returns all tuples \((R_{i}, S_{j}), P(R_{i}=\epsilon_{u} S_{j})\) where \( i = 1, ..., m \), \( j = 1, ..., n \) and \( P(R_{i}=\epsilon_{u} S_{j}) > 0 \).

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 probability exceeds a user-defined probabilistic threshold. Using this extra constraint, we show in the subsequent sections that it is possible to evaluate probabilistic join 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 \( \epsilon_{u} \) (where \( \epsilon_{u} \) is any one of \( =e, =l-e, =l-e \)), a Probabilistic Threshold Join Query (PTJQ) returns all tuples \((R_{i}, S_{j})\) such that \( i = 1, ..., m \), \( j = 1, ..., n \) and \( P(R_{i}=\epsilon_{u} S_{j}) > \tau \), where \( \tau \in [0,1] \) is called the probabilistic threshold.

A PTJQ only returns join pairs that have probabilities higher than \( \tau \). 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 blocked 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. A well implemented in typical database systems, and so 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 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 \( e \) (i.e., pairs that satisfy \( R_i, x + e < S_j, y \) or \( S_j, x + e < R_i, y \)), since according to Definition 4, these tuples have zero chance of being paired up. Thus, any I/O-efficient overlapping join algorithms over intervals (e.g., [6]), can be used. For \( p > p \), we can immediately eliminate \( (R_i, S_j) \) if \( R_i, x < S_j, y \), 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: \( R_i, S_j \) tables containing common uncertainty attributes
\( R_i, * \) uncertainty join operator
\( p \) probability threshold of PTJQ
Output: \( (R_i, S_j) \) that satisfies \( P(R_i, S_j) > p \)

Begin
1. let \( A - a \ast A \) is the answer of PTJQ
2. let \( C - \{ (R_i, S_j) \} \) where \( (R_i, S_j) \) are results returned by an interval join algorithm over \( R_i, U \) and \( S_j, U \)
   (For \( = \) and \( \ast \), join over \( [R_i, x - c, R_i, x + c], [S_j, y - c, S_j, y + c] \))
3. \( P(R_i, S_j) \) in \( C \)
   i. if \( P(R_i, S_j) > p \) then \( A - A \cup (R_i, S_j) \)
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, \( P(R_i, S_j) \), may not be higher than \( p \). We could simply compute \( P(R_i, S_j) \) 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 \( [a, b] \) to find \( P(R_i, S_j) \). 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 subintervals exceeds \( p \), we can immediately conclude that \( (R_i, S_j) \) 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 \( a \cup b \neq \emptyset \). Let \( a_{x, z} \) be \( \max(0, a - c) \) and \( b_{x, z} \) be \( \min(c, b + c) \). Then \( P(a = b) \) is at most
\[
\min(f(u_{x, z}), f(v_{x, z})) = a.F(u_{x, z}) - b.F(v_{x, z}) - b.F(u_{x, z})
\]
Proof: Suppose \( a \cup b \) overlap at interval \( [a_{x, z}, b_{x, z}] \), from Equation 1 we have

\[
\int_{a_{x, z}}^{b_{x, z}} a.F(x) dx - \int_{a_{x, z}}^{b_{x, z}} b.F(x) dx \leq \int_{b_{x, z}}^{a_{x, z}} f(x) dx
\]
Similarly, we have \( P(a = b) \) is at most \( f(u_{x, z}) - b.F(v_{x, z}) - b.F(u_{x, z}) \).
Since \( P(a = b) \) is equal to \( P(a = b) \), \( P(a = b) \)
cannot be larger than the minimum of \( a \cdot F(u_{\text{a,b,c}}) - a \cdot F(u_{\text{a,b,c}}) \) and \( b \cdot F(u_{\text{a,b,c}}) - b \cdot F(u_{\text{a,b,c}}) \). 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 cells of \( R_i \), \( F(z) \) and \( S_j \), \( F(x) \), we can obtain in constant time the values of \( R_i \cdot F(u_{\text{a,b,c}}) - R_i \cdot F(u_{\text{a,b,c}}) \) and \( S_j \cdot F(u_{\text{a,b,c}}) - S_j \cdot F(u_{\text{a,b,c}}) \). If any of these two values is less than \( p \), we can immediately conclude from Lemma 1 that \( P(R_i, S_j) < p \), 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 \( a \cdot U \cap b \cdot U \neq \emptyset \). Let \( u_{\text{a,b,c}} \) be \( \max(a - b, b - a) \) and \( u_{\text{a,b,c}} \) be \( \min(a + b, a + b) \). Then

\[
1 - \min(a \cdot P(u_{\text{a,b,c}})) - a \cdot F(u_{\text{a,b,c}}) - b \cdot F(u_{\text{a,b,c}}) - F(u_{\text{a,b,c}}) \\
\]

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 \( R_i \cdot U \) and \( S_j \cdot U \) do not overlap, \((R_i, S_j)\) is an answer for \( p \), immediately.

**Theorem 3** Suppose \( a \) and \( b \) are uncertain-valued variables. Then.

1. If \( a \cdot l \leq b \cdot l < c \cdot l \), \( P(a > b) \geq 1 - a \cdot F(b \cdot l) \).
2. If \( a \cdot l \leq a \cdot b \leq a \cdot b \leq 1 \cdot a \cdot F(b \cdot l) \).

**Proof:** Let \( a \cdot l \leq b \cdot l \leq a \cdot b \), \( P(a > b) \) be \( F(a \cdot b) \). From Definition 7, \( a \cdot l > F(a \cdot b) \) is equal to \( a \cdot F(a \cdot b) + F(a \cdot b) \). Then, \( P(a > b) = 1 - (a \cdot b) \).

In particular, for the interval join operation, performed in Step 2, we 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 sets of uncertainty operators. Next we study how this page-level pruning can be realized in different join algorithms.

The discussion focuses on the equality (=) 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-merging and indexed-merging join and nested-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-valued 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. 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 our 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 $0 < x < 1$, 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 $P_a[B.l(x) < y < a.r] \leq x$.
- If $a.r > B.r(x)$, then $P_a[B.l(x) < y < a.r] \leq x$.

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 left (right) as much as possible. To better understand the concept of x-bound, let us take a look at Figure 4 which 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., $P_a[B.l(x) < y < a.r] \leq 0.1$ and $P_a[B.l(x) < y < a.r] \leq 0.3$. Similarly, $b$ cannot have a probability of over 0.3 of being outside the right-0.3-bound. Finally, all the uncertain intervals must be fully enclosed by the x-bound, which is also 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 we have a range query with 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 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 $V_j$ is in a tuple of the form $(l, r)$ for storing the left-$V_j$-bound and right-$V_j$-bound. The values of $V_j$ ($j = 1, \ldots, |V|$) are stored in an external table $W$, sorted in ascending order of $V_j$. Our join algorithms require $O$-bounds to be stored, with $W_j$ equal to 0, and $[V_1, V_2]$ representing the position of the p-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 4: O-bound, 0.1-bound and 0.3-bound. A range query $[l, u]$ with $p = 0.4$ is also shown.](image)

**Figure 4**

![Figure 5: Implementing x-bounds in a page.](image)

**Figure 5**

![Table](image)

*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 Algo-

---

1 For example, if an uncertain attribute has a size of 8 bytes for storing its uncertainty interval, it takes 8 bytes to specify the uniform uncertainty pdf and cdf. A 4KB page can store 504 items. Joining the contents of two pages then requires examining 256² = 65536 pairs.
Theorem 5 Given a range query \( Q \) with interval \([a, b]\) 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

\[
\begin{align*}
&|a, b| \quad \text{"Lower and upper bound of range query \( Q \)"} \\
&\gamma \quad \text{"Probability threshold of range query"} \\
&W \quad \text{"Page with table \( B \)"} \\
\end{align*}
\]

Output

FALSE: All intervals in \( B \) are guaranteed to fail \( Q \). TRUE otherwise.

(a) CheckLeft\((l, u, p, B, W)\)

1. for \( i = 1, \ldots, |W| \) do
   
   (i) if \( l < B.V, u \) and \( W \) then
      
      (a) return FALSE
   
2. return TRUE

(b) CheckRight\((l, u, p, B, W)\)

1. for \( i = 1, \ldots, |W| \) do
   
   (i) if \( B.V, u < W \) 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 \( \neg \text{left-}x\)-bounds for pruning. (b) CheckRight uses \( \neg \text{right-}x\)-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 pages cannot be joined. EquiJoin returns CHECK to indicate that the pages cannot be joined without further investigation.

EquiJoin\((B, S, W, p)\)

1. if \( \text{NOT}(\text{CheckLeft}(B, S, V, l - c, B.R, V, l + c, p, B.S, W))) \) or \( \text{NOT}(\text{CheckRight}(B, S, V, l - c, B.R, V, l + c, p, B.S, W))) \) then return PRUNE
2. if \( \text{NOT}(\text{CheckLeft}(B, S, V, l - c, B.R, V, r + c, p, B.S, W))) \) or \( \text{NOT}(\text{CheckRight}(B, S, V, l - c, B.R, V, r + c, p, B.S, W))) \) then return PRUNE
3. return CHECK

Input

\[
\begin{align*}
&B \quad \text{"Page with uncertainty bounds" from table \( B \)} \\
&S \quad \text{"Page with uncertainty bounds" from table \( S \)} \\
&W \quad \text{"Global table storing values of \( x \) for \( x \)-bounds"} \\
&\gamma \quad \text{"Resolution of \( \gamma \)"} \\
\end{align*}
\]

Output

(PRUNE) \( B.R; S \) or \( S.B; R \) is certain that \( P(B.R = S \gamma) < p \). (CHECK) otherwise.

EquiJoin\((B, S, W, p)\)

Using \( \text{CheckLeft}(B, S, V, l - c, B.R, V, l + c, p, B.S, W) \) and \( \text{CheckRight}(B, S, V, l - c, B.R, V, l + c, p, B.S, W) \) of Step 1, which exchanges the role of domain \( B.D \) and \( S.D \) the range query is now constructed by using the \( \neg \text{left-}x\)-bound of \( B \) and tested against the uncertain values in \( B \). Again, EquiJoin returns PRUNE if either \text{CheckLeft} or \text{CheckRight} returns 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 \text{CheckLeft}, returns FALSE on pages \( B \) and \( S \). The other three conditions use the same principles and their proofs are skipped. We begin with the following lemma.

Theorem 6 If \( \text{CheckLeft}(B, S) \) in EquiJoin returns FALSE, then for every uncertain value \( S_j \) in \( B \), its probability of satisfying the range query formed by \( B \) with probability higher than \( p \). CheckRight then returns PRUNE to indicate that these pages cannot be joined.

Proof: From Lemma 5, we know that no attributes in \( B \) exist that satisfy the range query formed by \( B \) with probability higher than \( p \). Further, any uncertainty interval \( R.B \) in \( B \) must be enclosed by \( [B.R, V, l - c, B.R, V, l + c] \), and therefore \( R.B \) satisfies the range query \( B \). According to Step 1 of CheckLeft there must be some \( q \) such that \( B.B, V, l + c, B.B, V, r + c \). Therefore, \( B.B, V, r + c \) is a \( B \) with probability higher than \( p \). Therefore, \( R.B \) satisfies the range query \( B \).
As shown in Figure 8, none of the uncertainty intervals in \( B_2 \) crosses the line \( B_2 V_4 \) with \( s \) fraction of more than \( H_c \). This implies no values in \( B_2 \) can satisfy \( [R_i l - c, R_i r + c] \) with probability higher than \( p \).

\[ [R_i l - c, R_i r + c] \]

Figure 8: Illustrating the correctness of EquiJoin.

For any \( R_i \) and \( S_j \) stored in pages \( B_1 \) and \( B_2 \), the intersection between \( [R_i l - c, R_i r + c] \) and \( [S_j l - c, S_j r + c] \) is given by \( [B_{r n, s_j}, B_{w n, s_j}] \), where \( B_{r n, s_j} = \max(R_i l - c, S_j l - c) \) and \( B_{w n, s_j} = \min(R_i r + c, S_j r + c) \). The following lemmas can be derived.

Theorem 7

If CheckLeft of Step 1 in EquiJoin returns FALSE, then

\[ S_j F(w_{u, s_j}) - S_j F(u_{w, s_j}) < p \]  

(3)

Proof: Recall from Equation 2 that if CheckRight is FALSE, according to Lemma 5 no uncertain value in \( B_1 \) satisfies the range query constructed by the O-bound of \( B_2 \) with probability higher than \( p \) for being greater than any \( S_j r \) in \( B_2 \). This means \( \min(R_i r + c, S_j r + c) \) is less than \( S_j r \), which in turn cannot be larger than \( S_j r + c \). This means \( \min(R_i r + c, S_j r + c) \) 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 return FALSE. From Lemma 1, we know that \( P(S_j \geq R_i) \leq S_j F(w_{u, s_j}) - S_j F(u_{w, s_j}) \), 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 FRAME, 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 \( B_1 \) and \( B_2 \) join with probability of \( p \) or more, and these pairs can be inserted in the answer without hesitation (Step 2). The final kind of answer, CHECK, is returned when neither the conditions in Step 1 nor the conditions 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 in \( B_2 \) satisfies the range query constructed by the O-bound of \( B_2 \) with probability higher than \( p \). Also, there exists some \( q \) such that \( B_2 V_1 \geq B_2 V_4 + q \).

Proof: Recall from Equation 2 that if CheckRight is FALSE, according to Lemma 5 no uncertain value in \( B_2 \) satisfies the range query constructed by the O-bound of \( B_2 \) with probability higher than \( p \).
Input
\( R_1 * \) Page with uncertainty bounds from table \( R_1 * \)
\( R_2 * \) Page with uncertainty bounds from table \( S_2 * \)
\( p * \) probability threshold of \( > \) join * 

Output
(i) \( \text{PRUNE} \) if \( R_2 \cdot Vj \cdot r \in R_2 \cdot Sj \) \( \Rightarrow \) it is certain that \( P(R_1 \cap S_j) \geq \alpha \)
(ii) \( \text{INCLUDE} \) if \( R_2 \cdot Sj \) \( \in R_2 \cdot Sj \) \( \Rightarrow \) it is certain that \( P(R_1 \cap S_j) > \alpha \)
(iii) \( \text{CHECK} \) otherwise.

\( \text{GTJoin} \) \( R_2 \cdot W_2 \cdot p \)
1. if \( \{ \text{NOT(CheckLeft}(R_2 \cdot Vj \cdot l, R_2 \cdot Vj \cdot r, p, R_2 \cdot W_2)) \} \) or 
\( \{ \text{NOT(CheckRight}(R_2 \cdot Vj \cdot l, R_2 \cdot Vj \cdot r, 1 - p, R_2 \cdot W_2)) \} \) then return \( \text{PRUNE} \)
2. if \( \{ \text{NOT(CheckLeft}(R_2 \cdot Vj \cdot l, R_2 \cdot Vj \cdot r, 1 - p, R_2 \cdot W_2)) \} \) or 
\( \{ \text{NOT(CheckRight}(R_2 \cdot Vj \cdot l, R_2 \cdot Vj \cdot r, p, R_2 \cdot W_2)) \} \) then return \( \text{INCLUDE} \)
3. return \( \text{CHECK} \)

Figure 9: Page Level Join for \( R_1 > S_j \)

and \( W_s < p \). Since the lower bound of any uncertainty interval \( S_j \) in \( B_2 \) is not less than \( R_2 \cdot Vj \cdot l, S_j \cdot l \geq B_2 \cdot Vj \cdot l \).

Figure 10(a) illustrates the situation \( ^7 \). We can see that the overlap region between any \( R_i \) and \( S_j \) \( \Rightarrow \) addresses a total probability of not more than \( W_s \) for \( R_i \), i.e., \( 1 - P(R_i \cap S_j) \leq W_s \) which is less than \( p \). This implies that the cumulative probability of \( R_i \) from the \( R_i \cdot l \) to \( S_j \cdot l \) is larger than \( 1 - p \), i.e.,
\[
R_i \cdot F(S_j \cdot l) > 1 - p \tag{1}
\]

Since \( R_i \cdot l \leq S_j \cdot l \leq R_j \cdot r \) according to Lemma 3.2, we have

\[
P(R_i \cap S_j) \leq 1 - R_i \cdot F(S_j \cdot l) \leq 1 - (1 - p) \quad \text{(by Equation 1)}
\]

Therefore Lemma 8 holds.

Another test criterion in Step 1 applies \( \text{CheckLeft} \), where the roles of \( B_2 \) and \( B_3 \) are switched, and the range query is formed by the \( p \)-bounds of \( B_2 \). 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 \( \text{CheckRight} \) and \( \text{CheckLeft} \) of Step 1 is to test whether \( P(R_i \cap S_j) > p \), and if so, "close away" \( B_2 \) and \( B_3 \). Step 2 performs the opposite: it establishes the conditions in which every pair of items in \( B_2 \) and \( B_3 \) can be placed in the answer. Specifically, Step 2 verifies the condition \( P(S_j \cap R_i) > 1 - p \), which can be easily achieved by modifying the parameters in Step 1. Since \( P(R_i \cap S_j) = 1 - P(S_j \cap R_i) \), if any of the two conditions in Step 2 are satisfied, we can conclude that \( P(R_i \cap S_j) > p \). \( \text{GTJoin} \) then returns \( \text{INCLUDE} \) to indicate that all combinations of \( (R_i, S_j) \) can be inserted to the answer without probing. Similar to \( \text{EquiJoin} \), \( \text{GTJoin} \) only needs to call four small checking routines.

Similar to \( \text{EquiJoin} \), \( \text{GTJoin} \) 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 retrieves data in units of pages. Whenever two data pages are compared in the join algorithms, uncertain tables can be read first, and with our pruning techniques, probing into actual values in the pages can be prevented. Of course, \( \text{EquiJoin} \) may not prevent the retrieval of intervals when \( \text{INCLUDE} \) is returned — however, it still improves performance because we can simply add the Cartesian product of the intervals from the two range-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 \( \text{EquiJoin} \) or \( \text{GTJoin} \). We denote the version of \( \text{BNLJ} \) where uncertainty tables are augmented as \( \text{Uncertainty-based Block-Nested-Loop Join} \) (\( \text{U-BNLJ} \) for short). We will compare the performance differences experimentally between these two join al-
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 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 Rectangular (MBR) that encloses all the uncertainty intervals stored in this 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 B-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 O-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 as place of an internal 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. The first table, uncertainty intervals are uniformly distributed in [0,10000]. The length of each interval is normally distributed with a mean $\mu = 5$ and deviation $\sigma = 1$. 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 ($N_{prob}$) 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 so uncertainty bounds are used effectivley.

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 positive 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 return more 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
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 \( N_{\text{pair}} \) 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 \( N_{\text{pair}} \) is low when \( p \) is low.

**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,
Table 1: Pruning Methods for Uncertainty Joins.

<table>
<thead>
<tr>
<th>Level</th>
<th>Savings</th>
<th>Applicability Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item</td>
<td>Computation</td>
<td>$x_i &lt; x_j$, $x_i &gt; x_j$</td>
</tr>
<tr>
<td>Page</td>
<td>Computation</td>
<td>$x_i &lt; x_j$, $x_i &gt; x_j$</td>
</tr>
<tr>
<td>Index</td>
<td>$V$ &amp; computation</td>
<td>$x_i &lt; x_j$, $x_i &gt; x_j$</td>
</tr>
</tbody>
</table>

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 serious 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